Can we build any 2-input, single-1-output function using what we already know?
We can build (NOT, NOR, NAND) from CMOS gates, and combine them to build AND and OR.
Can we build arbitrary single-1-output functions from just these?

A	B	f
0	0	0
0	1	1
1	0	0
1	1	0

f is TRUE (= 1) exactly when

"(A is FALSE) AND (B is TRUE)" is a TRUE statement.

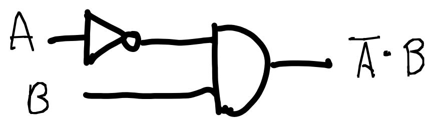
A	B	$\bar{A}$	$\bar{A}\cdot B = f$
0	0	1	0
0	1	1	1
1	0	0	0
1	1	0	0

TRUE

false

f is True

$$f = \bar{A}\cdot B \equiv \underbrace{m_1}_{\text{minterm}}$$

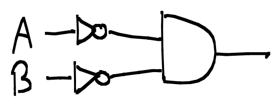


Our simple (single-1-output) functions are called "minterms". Our decomposition is called a "minterm expansion of f ".

Minterms

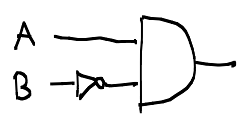
A	B	
0	0	1
0	1	0
1	0	0
1	1	0

$\longrightarrow \bar{A}\cdot\bar{B} \equiv m_0$



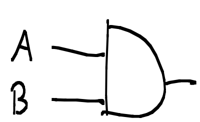
A	B	
0	0	0
0	1	0
1	0	1
1	1	0

$\longrightarrow A\cdot\bar{B} \equiv m_2$



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

$\longrightarrow A\cdot B \equiv m_3$



We now have all possible simple 2-input functions, ie., our minterms m0, m1, m2, and m3.

And we know how to build each from the 2-input gates we already have and NOT.

So, we can now build ANY 2-input function! Just OR these as needed.

Can we do the same for 3-input functions?

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

$\longrightarrow m_1 = \underbrace{\bar{A} \text{ and } \bar{B} \text{ and } C}_{\text{A 3-input function?}}$

We CAN build all 2-input functions.
BUT, WHAT is a 3-input function?
CAN we build from 2-input functions?

AND(X ,Y, Z)

is that the same as,

AND(AND(X, Y), Z) ???

A	B	A · B
0	0	0
0	1	0
1	0	0
1	1	1

(A · B)	C	(A · B) · C
0	0	0
0	1	0
1	0	0
1	1	1

Are these the same?

Do they have exactly the same output values for exactly the same input values? Check ALL input values.

B	C	B · C
0	0	0
0	1	0
1	0	0
1	1	1

A	(B · C)	A · (B · C)
0	0	0
0	1	0
1	0	0
1	1	1

A	B	C	(A · B) · C
0	0	0	?
0	0	1	?
0	1	0	?
0	1	1	?
1	0	0	?
1	0	1	?
1	1	0	?
1	1	1	?

SAME?

A	B	C	A · (B · C)
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Basic Algebraic Properties

equivalent functions

same output

1	P	1 · P
1	0	0
1	1	1

1	P	P
1	0	0
1	1	1

a logical constant = TRUE for all inputs.

Proof

P	P	P · P
0	0	0
1	1	1

P	P
0	0
1	1

$1 \cdot P = P$

$0 \cdot P = 0$

$P \cdot P = P$

$P \cdot \overline{P} = 0$

$A \cdot B = B \cdot A$

$(AB)C = A(BC)$

Proof

P	$\overline{P}$	$P \cdot \overline{P}$
0	1	0
1	0	0

always 0

Proof

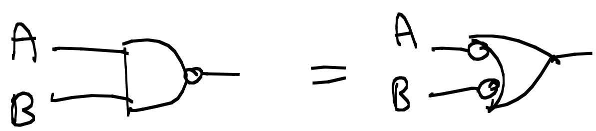
Handy tricks for (1) proving algebraic properties or doing algebraic operations, and (2) converting logic circuits from one type of logic gates to another. (Aside: can you prove Duality?)

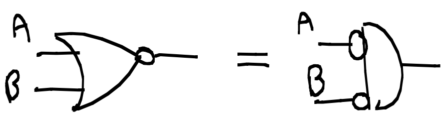
DeMorgan's Laws

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

A	B	$A\cdot B$	$\overline{AB}$
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

$\overline{A}$	$\overline{B}$	$\overline{A}+\overline{B}$
1	1	1
1	0	1
0	1	1
0	0	0





$$\overline{A+B} = \overline{A}\cdot\overline{B}$$ DUAL

Duality

$\cdot \rightarrow +$
$+ \rightarrow \cdot$
$1 \rightarrow 0$
$0 \rightarrow 1$

starting with an identity, gives an identity.

$1\cdot P = P$
$0+P = P$
duality

$0\cdot P = 0$
$1+P = 1$
duality

$(A+B)C = A\cdot C + B\cdot C$
$(A\cdot B)+C = (A+C)\cdot(B+C)$
dual

Build any k-input function? Use k-ary minterm expansion.

Build k-ary minterms from 2-input gates we already have?

A	B	C	f(A,B,C)	m_0	m_3	m_4	m_6
0	0	0	1	1	0	0	0
0	0	1	0	0	0	0	0
0	1	0	0	0	0	0	0
0	1	1	1	0	1	0	0
1	0	0	1	0	0	1	0
1	0	1	0	0	0	0	0
1	1	0	1	0	0	0	1
1	1	1	0	0	0	0	0

$$f = \underbrace{m_0 + m_3 + m_4 + m_6}_{\text{minterm expansion}}$$

(prove '+' associative)

(Recall: $P + 0 = P$)

$\Rightarrow m_0 + 0 + 0 + 0 = m_0$

any row is similar, 1 non-zero OR'd w/ zeroes.

What sort of functions are these minterms?

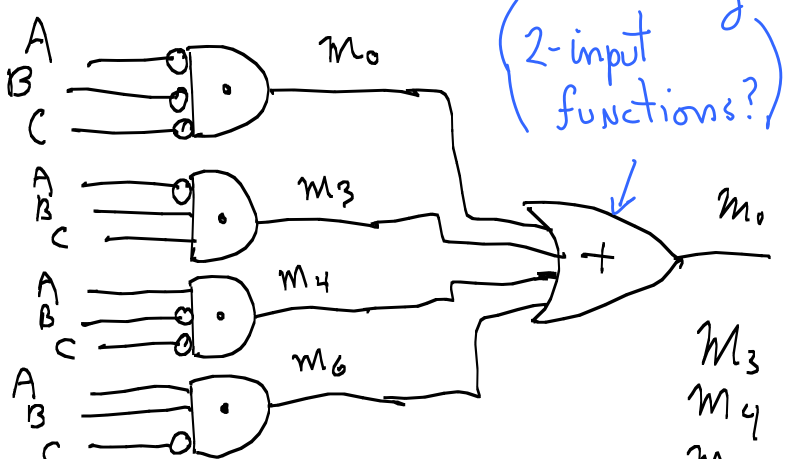
if A is F and B is F and C is F

A	B	C	m_0
0	0	0	1
0	0	1	0
0	1	0	0
⋮	⋮	⋮	⋮
1	1	1	0

$$m_0(A,B,C) = \underbrace{\bar A \cdot \bar B \cdot \bar C}$$

$$\begin{aligned}
\bar 0 \cdot \bar 0 \cdot \bar 0 &= 1\\
\bar 0 \cdot \bar 0 \cdot \bar T &= 0\\
\bar 0 \cdot \bar T \cdot \bar 0 &= 0\\
\bar 0 \cdot \bar T \cdot \bar T &= 0\\
\bar T \cdot \bar 0 \cdot \bar 0 &= 0\\
\bar T \cdot \bar 0 \cdot \bar T &= 0\\
\bar T \cdot \bar T \cdot \bar 0 &= 0\\
\bar T \cdot \bar T \cdot \bar T &= 0
\end{aligned}$$

yes, it matches, i.e. $\bar A \bar B \bar C$ really is correct expression for m_0.



Build using (2-input functions?)

$$m_0 + m_3 + m_4 + m_6 = f$$

$$m_3 = \bar A B C$$
$$m_4 = A \bar B \bar C$$
$$m_6 = A B \bar C$$

Complete Logic set

(OR AND NOT)

MAXTERM EXPANSION

A	B	C	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

6 minterms, but only 2 zeroes?

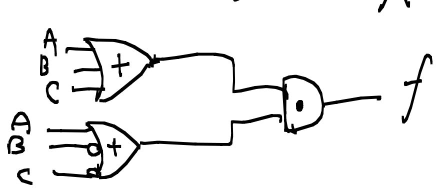
$\bar f$	
1	→ m_0
0	
0	
1	→ m_3
0	
0	
0	
0	

2 minterms

$$\bar f = m_0 + m_3 = \bar A \bar B \bar C + \bar A B C$$

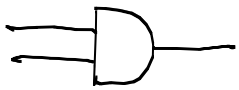
$$f = \bar{\bar f} = \overline{\bar A \bar B \bar C + \bar A B C} \xrightarrow{\text{de Morgan}} \left(\overline{\bar A \bar B \bar C}\right)\left(\overline{\bar A B C}\right)$$

$$= (A + B + C)(A + \bar B + \bar C)$$

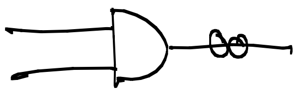


Pushing Bubbles

implement using NOR logic



= f



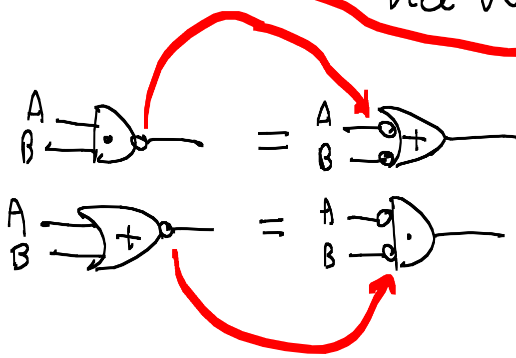
= $\bar{\bar{f}}$ = f

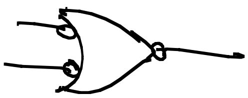
Bubble goes from output side to all inputs, or vice versa

De Morgan's Laws

$\overline{AB} = \bar{A} + \bar{B} \Rightarrow$

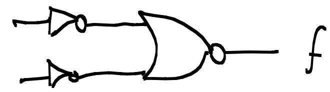
$\overline{A+B} = \bar{A}\cdot\bar{B} \Rightarrow$



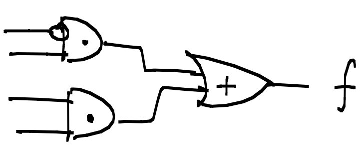


= f

equivalent circuit in NOR logic

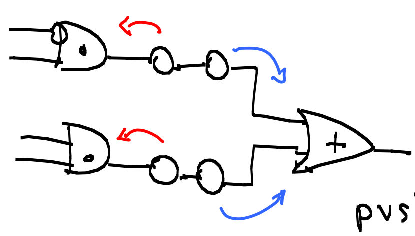


AND-OR ➔ NAND-NAND



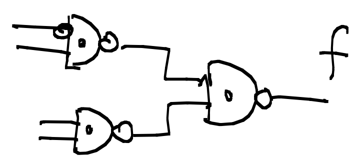
➔

Add bubble pairs

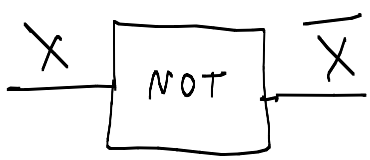


push bubbles

➔

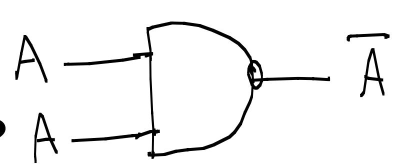


NAND—NAND is complete logic family, if we have NOT.

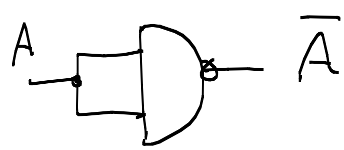


is there a NAND circuit for this function?

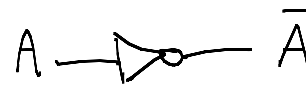
A	$\cancel{B}$ A	NAND
0	0	1
~~0~~	~~1~~	~~1~~
~~1~~	~~0~~	~~1~~
1	1	0



=

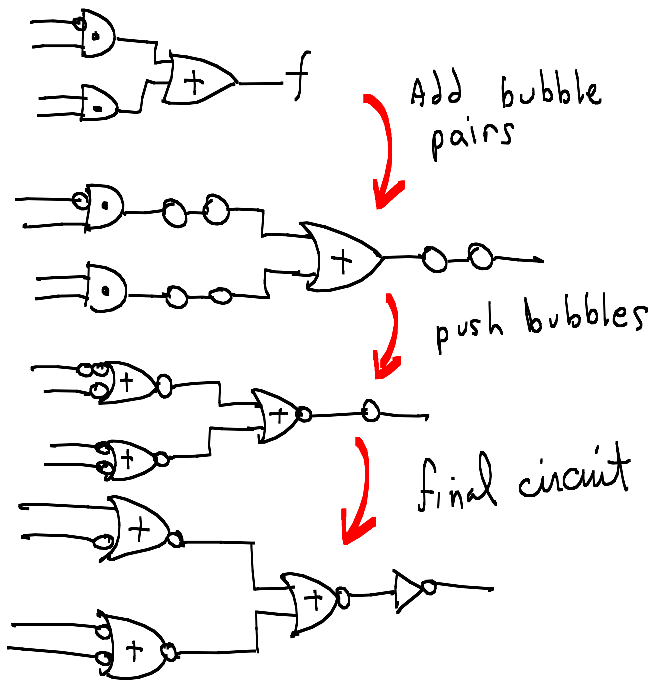


=

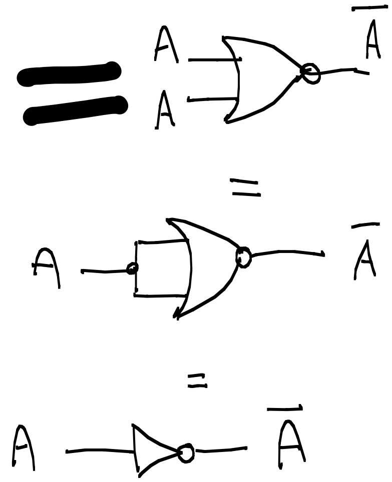


NOR-NOR is complete, if we have NOT.

AND-OR ➜ NOR-NOR



A	~~B~~ A	NOR
0	0	1
~~0~~	~~1~~	~~0~~
1	1	0

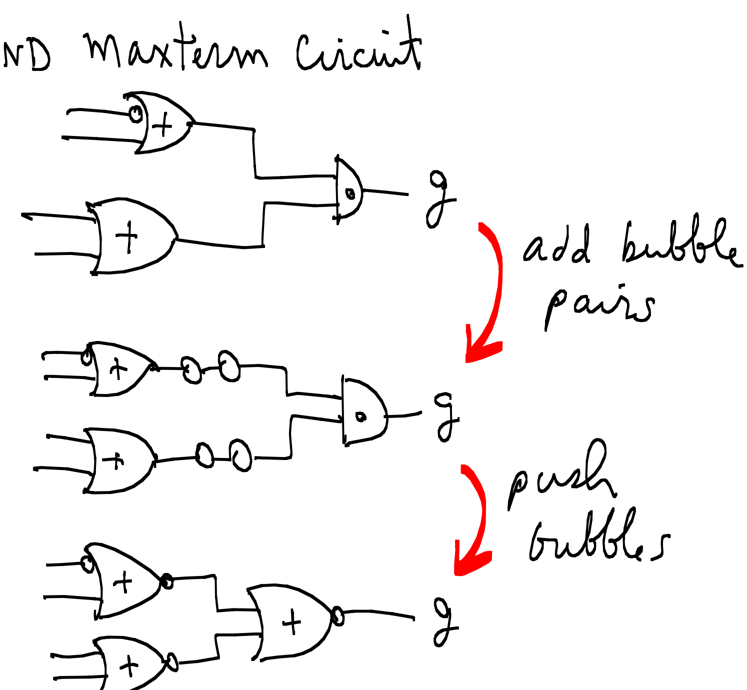


We can push bubbles to get a NOR-NOR circuit from an AND-OR circuit.
But, is there an easier way?

Is there an easier way to Get a NOR-NOR circuit?

yes, use a maxterm expansion

OR-AND Maxterm Circuit



NOR-NOR maxterm circuit

Karnaugh Maps are truth tables that allow us to find simpler logic expressions (and circuits). This is something that can also be done algebraically, but is usually harder that way. Is there a general procedure for minimizing circuits? (Is it computable?)

f: $A \backslash B$	0	1
0	m_0	m_1
1	m_2	m_3

=

A	B	f
0	0	m_0
0	1	m_1
1	0	m_2
1	1	m_3

Suppose $f = m_0 + m_1 + m_2$

f: $AB \backslash C$	0	1
00	1	1
01	1	0
11	0	0
10	0	0

*

this is $m_0 + m_1 = \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C = \bar{A}\bar{B}(C+\bar{C}) = \bar{A}\bar{B}$

this is $m_0 + m_2 = \bar{A}\bar{C}$

$\rightarrow f = \bar{A}\bar{B} + \bar{A}\bar{C}$

* Gray code

g: $AB \backslash CD$	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_8	m_9	m_{11}	m_{10}
10	m_{12}	m_{13}	m_{15}	m_{14}

$B=1$

$A=1$

$C=1$

$D=1$

if all entries are 1's,

term $= m_5 + m_7 + m_9 + m_{11}$

$= (B=1)\cdot(D=1)$

$= BD$

Find a collection of terms that:
1. covers all ones
2. has the fewest terms possible.

Term = a region where one or more variables are "don't cares": the output does not change when the variables change value, those variables make no difference. The values of the remaining variables define the term.

So, what can we say about mult-bit symbols as output from our Boolean functions? Deal with each bit of output symbol individually.

	f	$f_1 f_0$
00	$\longrightarrow$	01
01	$\longrightarrow$	10
10	$\longrightarrow$	11
11	$\longrightarrow$	00

$\Longrightarrow$

		f_1
0	0	0
0	1	1
1	0	1
1	1	0

		f_0
0	0	1
0	1	0
1	0	1
1	1	0

are these 2-bit symbols or 2 1-bit symbols?
Could be either:

$f: \{0,1\} \times \{0,1\} \longrightarrow \{00, 01, 10, 11\}$

4 symbols

2 symbols

2 symbols

$\{A, B\} \times \{\Theta, \phi\} \longrightarrow \{@, \&, *, \$\}$

$S_1 \times S_2 \longrightarrow S_3$

OR

$f: \{00, 01, 10, 11\} \longrightarrow \{00, 01, 10, 11\}$

4 symbols

4 symbols

$\{@, \&, *, \$\} \longrightarrow \{@, \&, *, \$\}$

$S_3 \longrightarrow S_3$

	g	
000	$\longrightarrow$	00
001	$\longrightarrow$	01
010	$\longrightarrow$	01
011	$\longrightarrow$	10
100	$\longrightarrow$	01
101	$\longrightarrow$	10
110	$\longrightarrow$	10
111	$\longrightarrow$	11

In general, we can now handle any functions with m-bit input and n-bit output. We can implement in hardware:

===> ANY finite symbol set of size 2^k by using {0, 1}^k multi-bit symbols
===> ANY finite combination of finite sets
===> ANY functions mapping from any combination to any combination