Lecture 5

- Converting to use NAND and NOR
- Minimizing functions using Boolean cubes

Minimal set

- Can implement any logic function from NOT, NOR, and NAND
- In fact, can do it with only NORs and NANDs
  - NOT is just NAND or NOR with two identical inputs

X	**Y**	**X nor Y**
0	**0**	**1**
1	**1**	**0**

X	**Y**	**X nand Y**
0	**0**	**1**
1	**1**	**0**

Why NAND/NOR?

- NAND/NOR preferred for real hardware implementation
  - More efficient (less switches per gate)
- But how do we convert from the canonical forms that are expressed in AND/OR?

NAND/NOR truth tables

$(X + Y)' = X' \bullet Y'$
NOR is equivalent to AND with inputs complemented

X	Y	X'	Y'	(X + Y)'	X' • Y'
0	0	1	1	1	1
0	1	1	0	0	0
1	0	0	1	0	0
1	1	0	0	0	0

$(X \bullet Y)' = X' + Y'$
NAND is equivalent to OR with inputs complemented

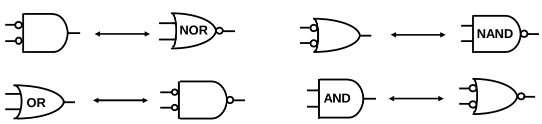
X	Y	X'	Y'	(X • Y)'	X' + Y'
0	0	1	1	1	1
0	1	1	0	1	1
1	0	0	1	1	1
1	1	0	0	0	0

Pushing the bubble

- Apply DeMorgan's

$A'B' = (A + B)'$ $\quad A' + B' = (AB)'$

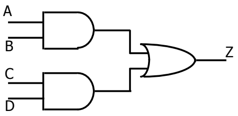
$A + B = (A'B')'$ $\quad (AB) = (A' + B')'$

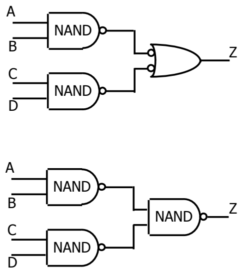


Converting to NAND/NOR

- Introduce inversions ("bubbles")
  - Introduce bubbles in pairs
    - Conserve inversions
    - Do not alter logic function

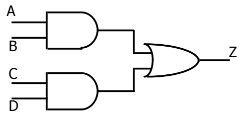
AND/OR to NAND/NAND

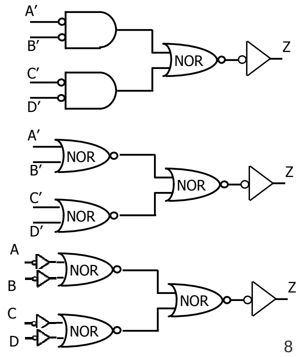




Z = AB + CD
= (A'+B')'+(C'+D')'
= [(A'+B')(C'+D')]'
= [(AB)'(CD)']'

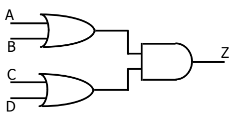
AND/OR to NOR/NOR

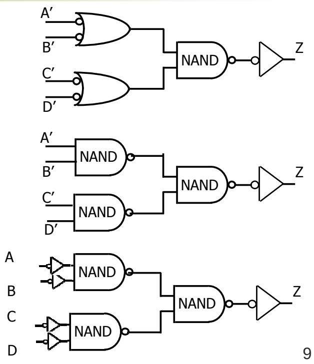




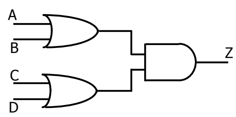
Z = AB + CD
= (A'+B')'+(C'+D')'
= [(A'+B')'+(C'+D')']"
= {[(A'+B')'+(C'+D')']'}'

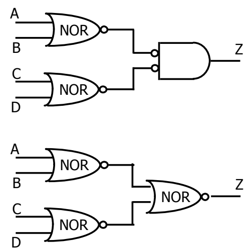
OR/AND to NAND/NAND





OR/AND to NOR/NOR





Goal: Logic minimization

- Algebraic simplification
  - Not a systematic procedure
  - Hard to know when we reached the minimum
- Computer-aided design tools
  - Require very long computation times (NP hard)
  - Heuristic methods employed–"educated guesses"

Goal: Logic minimization

- Visualization methods are useful
  - Our brain is good at figuring "simple" things out
  - Many real-world problems are solvable by hand

Key tool: Uniting Theorem

- Uniting theorem: A(B'+B) = A
- The approach:
  - Find where some variables don't change (the A's above) and others do (the B's above)
  - Eliminate the changing variables (the B's)

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

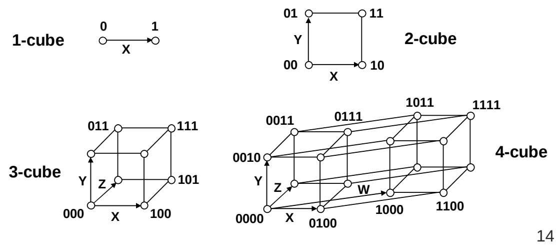
A has the same value in both "on-set" rows ⇒ keep A

B has a different value in the both rows ⇒ eliminate B

F = A'B'+A'B = A'(B+B') = A'

Boolean cubes

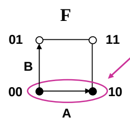
- Visualization tool for the uniting theorem
  - n input variables = n-dimensional "cube"



Mapping to Boolean cubes

- ON set = solid nodes
- OFF set = empty nodes

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



Look for on-set adjacent to each other

Sub-cube (a line) comprises two nodes

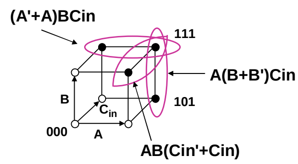
A varies within the sub-cube; B does not

This sub-cube represents B'

Example: Full adder carry-out

- Binary full-adder carry-out logic
  - On-set is covered by the OR of three 1-D subcubes

A	B	Cin	Cout
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

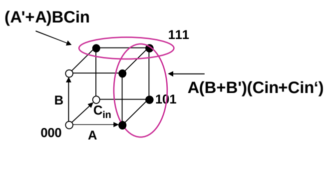


Cout = BCin+AB+ACin

Example

- Changed one bit from the previous function
  - On-set is covered by the OR of one 2-D subcube and one 3-D subcube

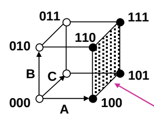
A	B	Cin	D
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1



D = BCin+A

M-D cubes in N-D space

- In a 3-cube (three variables):
  - A 0-cube (a single node) yields a term in 3 literals
  - A 1-cube (a line of two nodes) yields a term in 2 literals
  - A 2-cube (a plane of four nodes) yields a term in 1 literal
  - A 3-cube (a cube of eight nodes) yields a constant term "1"



F(A,B,C) = $\sum m(4,5,6,7)$

On-set forms a square (a 2-D cube)

A is asserted (true) and unchanging
B and C vary

This sub-cube represents the literal A