Lecture 6

- Logistics
  - HW2 due on Wednesday
  - Lab 2 this week
- Last lecture
  - Canonical forms
  - NAND and NOR
- Today's lecture
  - More NAND and NOR and pushing bubbles
  - Logic simplification: Visualization techniques
    - Boolean cubes
    - Karnaugh maps

The "WHY" slide

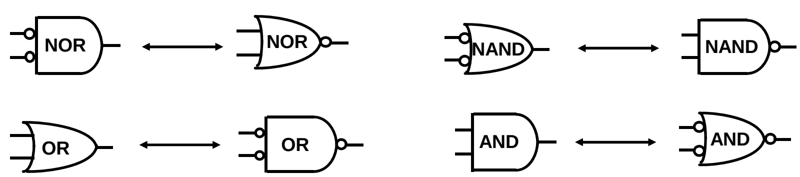
- Converting to use NAND and NOR
  - NAND and NOR are more efficient gates than AND or OR (and therefore more common). Your computer is built almost exclusively on NAND and NOR gates. It is good to knowhow to convert any logic circuits to a NAND/NOR circuit.
- Pushing bubbles
  - It is always good to remember logical/theoretical concepts visually. This is one way to remember the NAND/NOR conversion easily.
- Logic Simplification
  - If you are building a computer or a cool gadget, you want to optimize on size and efficiency. Having extra unnecessary operations/gates is not great. We teach nice techniques to allow logic simplifications.

NAND and NOR (logic gates)

- de Morgan's
  - Standard form: A'B' = (A + B)'   A' + B' = (AB)'
  - Inverted: A + B = (A'B')'   (AB) = (A' + B')'

  - AND with complemented inputs ≡ NOR
  - OR with complemented inputs ≡ NAND
  - OR ≡ NAND with complemented inputs
  - AND ≡ NOR with complemented inputs

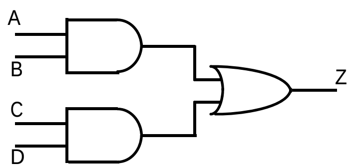
pushing the bubble

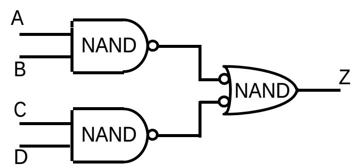


Converting to use NAND/NOR

- Introduce inversions ("bubbles")
  - Introduce bubbles in pairs
    - Conserve inversions
    - Do not alter logic function
- Example
  - AND/OR to NAND/NAND

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

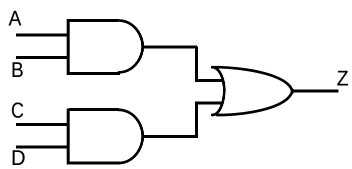


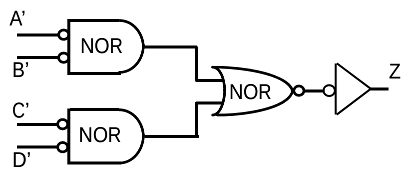


Converting to use NAND/NOR (con't)

◆ Example: AND/OR network to NOR/NOR

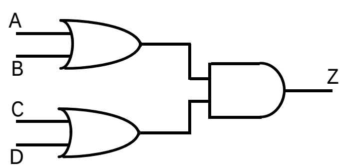
Z = AB+CD
= (A'+B')'+(C'+D')'
= [(A'+B')'+(C'+D')']''
= {[(A'+B')'+(C'+D')']'}'

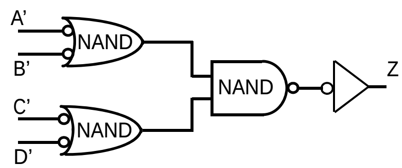




Converting to use NAND/NOR (con't)

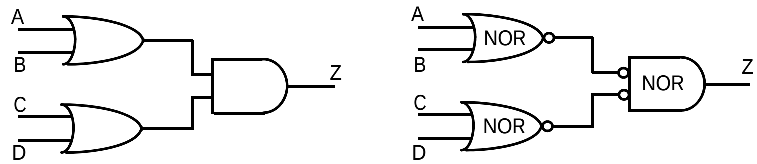
◆ Example: OR/AND to NAND/NAND



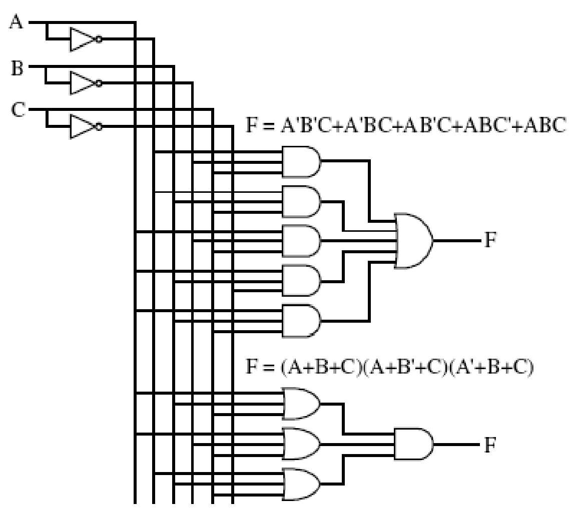


Converting to use NAND/NOR(con't)

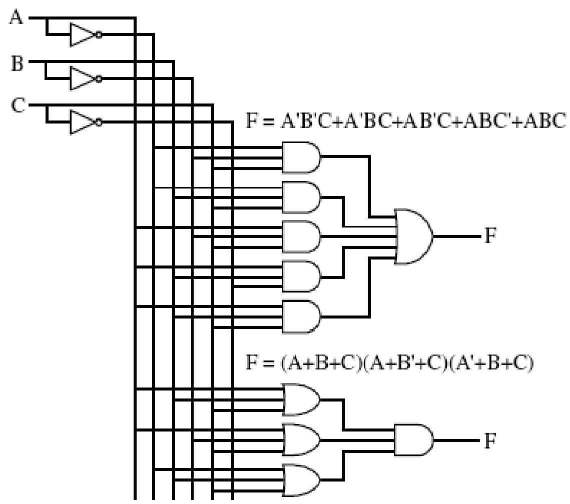
- Example: OR/AND to NOR/NOR



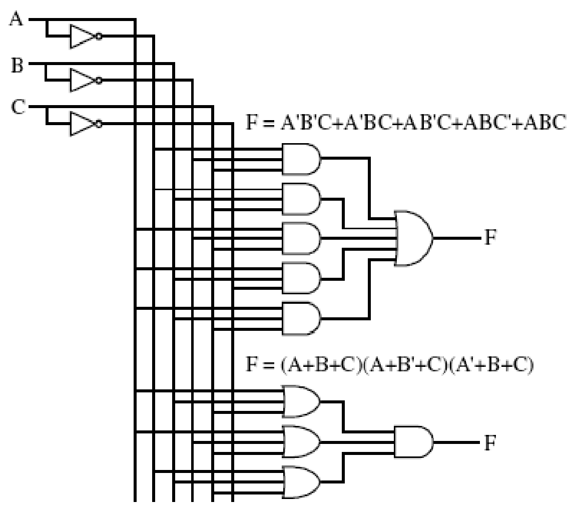
Example of bubble pushing: before pushing



Example of bubble pushing: NAND/NAND



Example of bubble pushing: NOR/NOR



Goal: Minimize two-level logic expression

- Algebraic simplification
  - not an 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"
- Visualization methods are useful
  - our brain is good at figuring things out over computers
  - many real-world problems are solvable by hand

Key tool: The Uniting Theorem

- The uniting theorem → A(B'+B) = A
- The approach:
  - Find 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

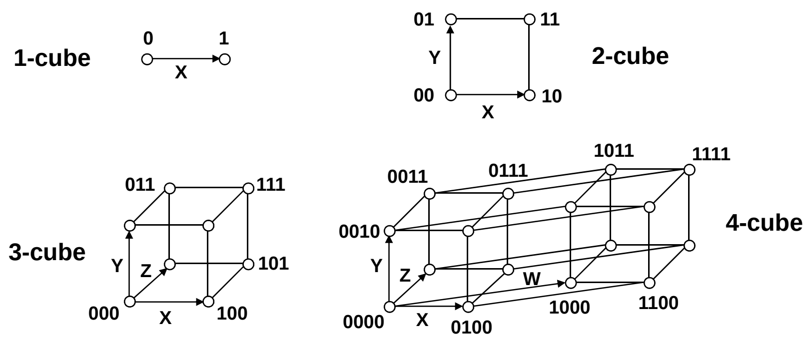
A has the same value in both "on-set" rows
⇒ keep A

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

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

Boolean cubes

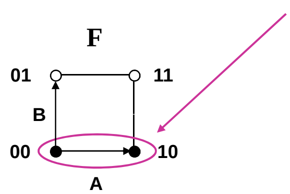
- Visualization tool for the uniting theorem
  - n input variables = n-dimensional "cube"



Mapping truth tables onto 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

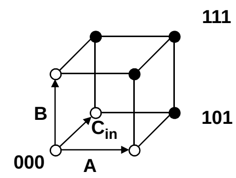
A varies within the sub-cube; B does not

This sub-cube represents B'

Example using Boolean cube

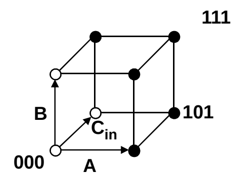
- Binary full-adder carry-out logic

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



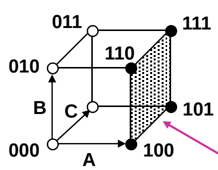
Another example using Boolean cube

A	B	Cin	Cout
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



M-dimensional cubes in n-dimensional 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

Karnaugh maps (K-map)

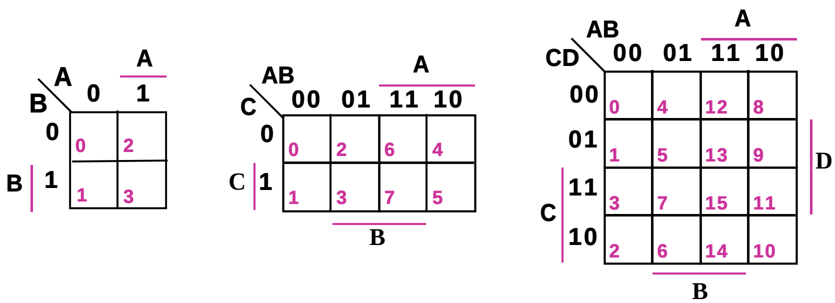
- Flat representation of Boolean cubes
  - Easy to use for 2– 4 dimensions
  - Hard for 4 – 6 dimensions
  - Virtually impossible for 6+ dimensions
    - Use CAD tools
- Help visualize adjacencies
  - On-set elements that have one variable changing are adjacent

	A	B	F
0	0	0	1
1	0	1	0
2	1	0	1
3	1	1	0

B \ A	0	1
0	$_0$ 1	$_2$ 1
1	$_1$ 0	$_3$ 0

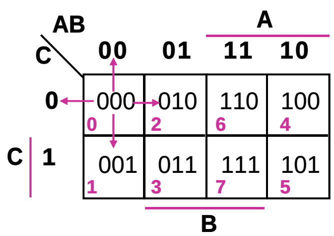
2, 3, and 4 dimensional K-maps

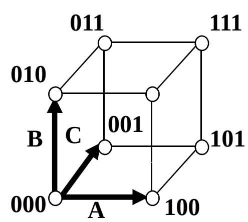
- Uses Gray code: Only one bit changes between cells
  - Example: 00 → 01 → 11 → 10



Adjacencies

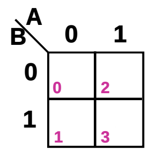
- Wrap–around at edges
  - First column to last column
  - Top row to bottom row





K-map minimization example: 2 variables

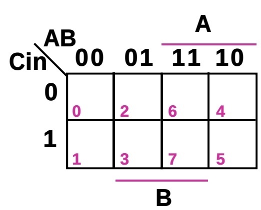
	A	B	F
0	0	0	1
1	0	1	0
2	1	0	1
3	1	1	0



F =

K-map minimization example: 3 variables

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 =

One more example: 3 variables

A	B	Cin	Cout
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