
Computer Science 141

Computing Hardware

Fall 2009

Harvard University

Instructor: Prof. David Brooks

dbrooks@eecs.harvard.edu

Today's Lecture

- Karnaugh Maps/Logic Minimization
- Time Response/Glitches

Sum-of-Products

- two-level canonical form of a function
 - standard form
 - unique algebraic signature (deterministically determined)
 - two alternative forms: SOPs and POSs
- Sum of Products (SOPs)
 - a.k.a. disjunctive normal form or minterm expansion

A	B	C	F	\bar{F}	minterm
0	0	0	0	1	$m_0 = \overline{ABC}$
0	0	1	0	1	$m_1 = \overline{AB}C$
0	1	0	0	1	$m_2 = \overline{A}B\bar{C}$
0	1	1	1	0	$m_3 = \overline{A}BC$
1	0	0	1	0	$m_4 = A\bar{B}\bar{C}$
1	0	1	1	0	$m_5 = A\bar{B}C$
1	1	0	1	0	$m_6 = AB\bar{C}$
1	1	1	1	0	$m_7 = ABC$

$$\begin{aligned}
 F(A,B,C) &= \overline{ABC} + \overline{AB}C + \overline{A}B\bar{C} + \overline{A}BC + A\bar{B}\bar{C} \\
 &= m_3 + m_4 + m_5 + m_6 + m_7 = \sum_{A,B,C} (3,4,5,6,7)
 \end{aligned}$$

Review: Product of Sums

- Product of Sums (POSs)
 - a.k.a. conjunctive normal form or maxterm expansion
 - construction is logical dual of SOP's construction

A	B	C	F	\bar{F}	maxterm
0	0	0	0	1	$M_0 = A + B + C$
0	0	1	0	1	$M_1 = A + B + \bar{C}$
0	1	0	0	1	$M_2 = A + \bar{B} + C$
0	1	1	1	0	$M_3 = A + \bar{B} + \bar{C}$
1	0	0	1	0	$M_4 = \bar{A} + B + C$
1	0	1	1	0	$M_5 = \bar{A} + B + \bar{C}$
1	1	0	1	0	$M_6 = \bar{A} + \bar{B} + C$
1	1	1	1	0	$M_7 = \bar{A} + \bar{B} + \bar{C}$

$$F(A,B,C) = (A + B + C)(A + B + \bar{C})(A + \bar{B} + C) = M_0 \bullet M_1 \bullet M_2 = \prod M(0,1,2)$$

$$\bar{F} = \prod M(3,4,5,6,7)$$

Can obtain POSs from applying DeMorgan's theorem to the SOPs of \bar{F} (and vice versa)

Why do logic minimization?

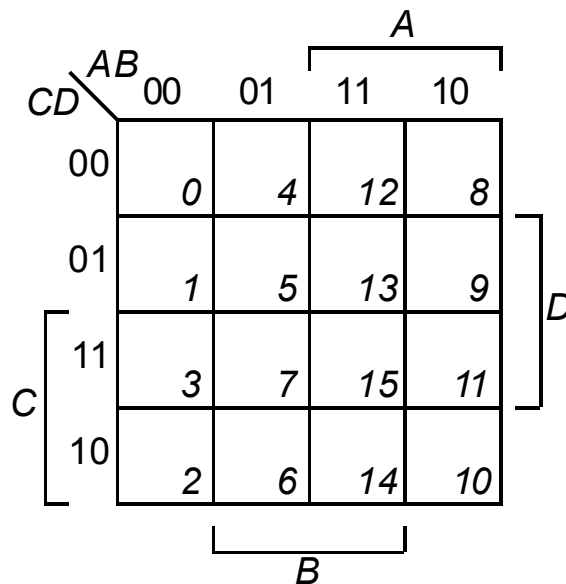
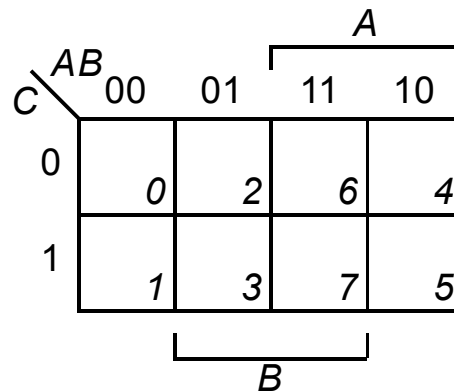
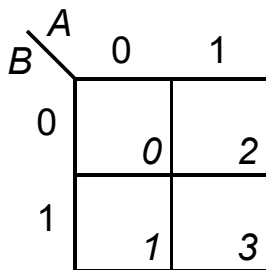
- Minimize amount of wiring in circuit (minimize no. of literals)
 - wiring takes space and is a difficult problem
 - physical gates have limited no. of inputs
- Minimize no. of gates in circuit
 - gates take area
 - minimal no. of gates \neq smallest wiring
- Minimize circuit propagation delay (minimize no. of gate levels)
 - faster is better (time is important measure of computer performance)
 - faster is definitely not smaller

Practical methods for boolean simplification

- All based on boolean algebra, but more systematic
- Methods reduce no. of literals and gates to implement function
- First: 2-level simplification, then multilevel methods
- Karnaugh map method (1953)
 - visual method for identifying boolean cubes
 - represent truth table of n-input variables as “cube” in
 - n-dimensional space
 - simplification via uniting theorem:
 - Uniting: $A \cdot B' + A \cdot B = A(B' + B) = A$

K-map

- tabular structure (effective up to about 6-input variables)
- maps organized so only 1 bit changes between any adjacent blocks
 - horizontally or vertically, not diagonally
 - map wraps around on itself
 - reduce by circling the largest possible on-set groups



examples

$$F = A\bar{B} + AB \\ = A$$

	A	0	1
B	0	0	1
1	0	0	1

$$F = \sum m(0,4,5,6,7) \\ = A + \bar{B}\bar{C}$$

			A		
	AB	00	01	11	10
C	0	1	0	1	1
1	0	0	1	1	
		B			

examples

$F = \sum m(0,2,3,5,6,7,8,10,11,14,15)$

CD \ AB		A			
		00	01	11	10
C	00	1	0	0	1
	01	0	1	0	0
	11	1	1	1	1
	10	1	1	1	1

B

D

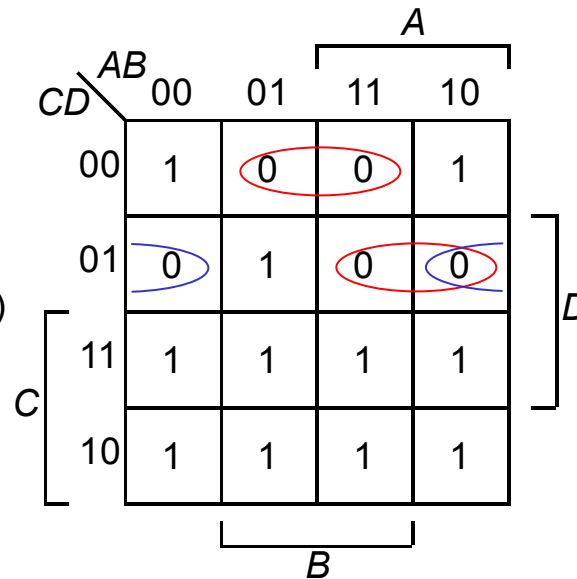
- 5- and 6-variable maps possible (see exercise in Wakerly)

examples

- Finding minimum POSs expressions

DeMorgan's 

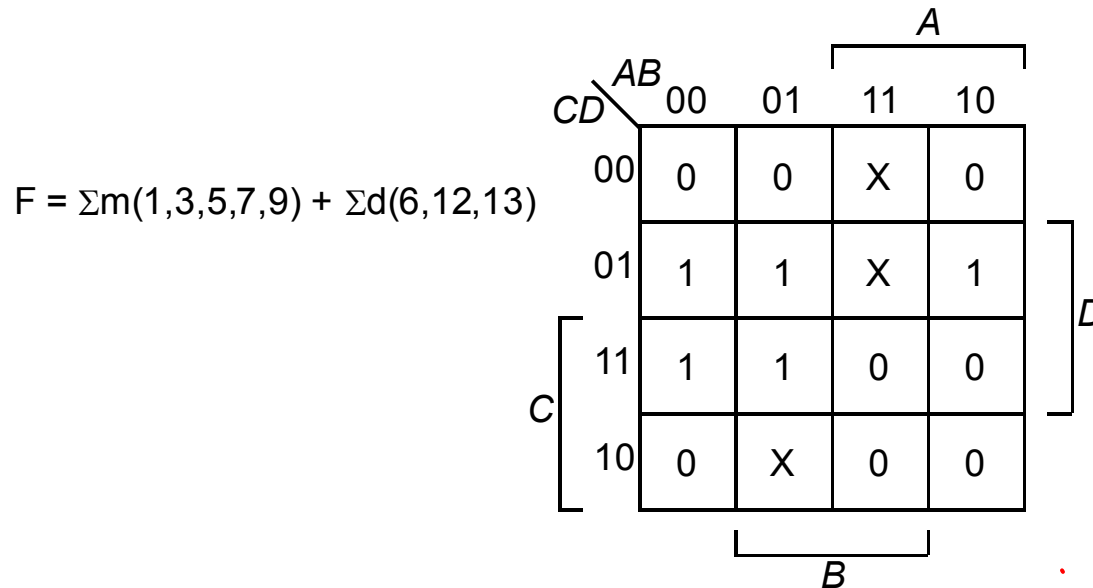
$$\begin{aligned}\bar{F} &= \sum m(1,4,9,12,13) \\ &= B\bar{C}\bar{D} + A\bar{C}D + \bar{B}\bar{C}D \\ F &= (\bar{B}+C+D)(\bar{A}+C+\bar{D})(B+C+\bar{D})\end{aligned}$$



A 4x4 Karnaugh map for variables A, B, C, and D. The columns are labeled AB (00, 01, 11, 10) and the rows are labeled CD (00, 01, 11, 10). The map contains 1s in all cells except for (01,00), (11,00), (01,01), and (11,01). Three groups are circled: a red group for A (cells (01,00), (11,00), (01,01), (11,01)), a blue group for B (cells (00,01), (01,01), (10,01), (11,01)), and a black group for D (cells (01,00), (01,01), (11,00), (11,01)).

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

dealing with don't care terms

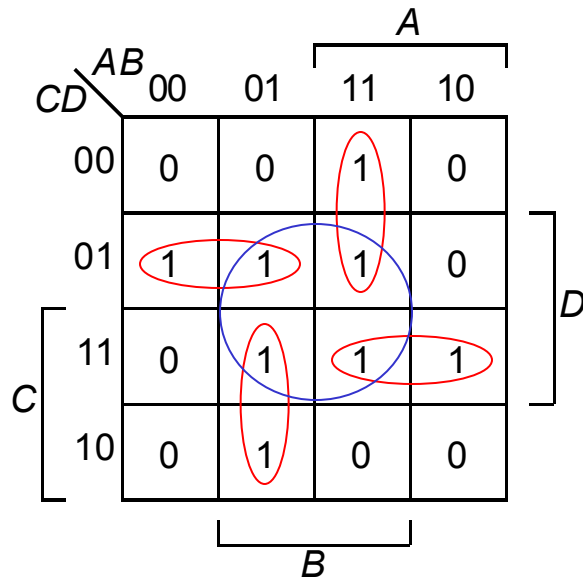


- don't care vs. don't know
 - don't care about behavior under these inputs (out-of-range)
 - don't know the value of this output (undefined value)

Terminology for 2-level simplification

- **implicant**
 - single element of the ON-set or any group of elements that can be combined together in a K-map
- **prime implicant**
 - implicant that cannot be combined with another implicant to eliminate a term
- **essential prime implicant**
 - if an element of the ON-set is covered by a single implicant, that implicant is essential

examples



- 5 prime implicants: $BD, ABC\bar{C}, ACD, \bar{A}BC, \bar{A}CD$
- 4 essential prime implicants: $ABC\bar{C}, ACD, \bar{A}BC, \bar{A}CD$

2-level simplification of SOPs expression

Algorithm

- consider each element of the ON-set in turn
- find “maximal” groupings of 1’s and X’s adjacent to element
- repeat Steps 1 and 2 till found all prime implicants
- find all essential prime implicants
- if any remaining 1’s not covered by essential prime implicants, select smallest number of prime implicants that cover remaining 1’s

algorithmic example

- Original k-map

CD \ AB		A			
		00	01	11	10
C	00	X	1	0	1
	01	0	1	1	1
	11	0	X	X	0
	10	0	1	0	1

- Find all primes

CD \ AB		A			
		00	01	11	10
C	00	X	1	0	1
	01	0	1	1	1
	11	0	X	X	0
	10	0	1	0	1

Example (conti.)

- Find all essential primes and minimum covering of remaining 1's

$CD \backslash AB$		A			
		00	01	11	10
C	00	X	1	0	1
	01	0	1	1	1
	11	0	X	X	0
	10	0	1	0	1

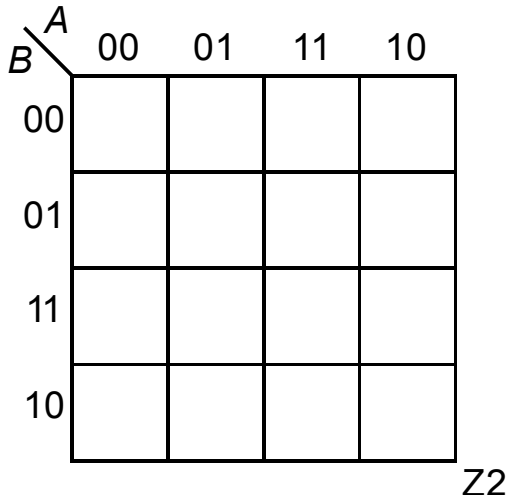
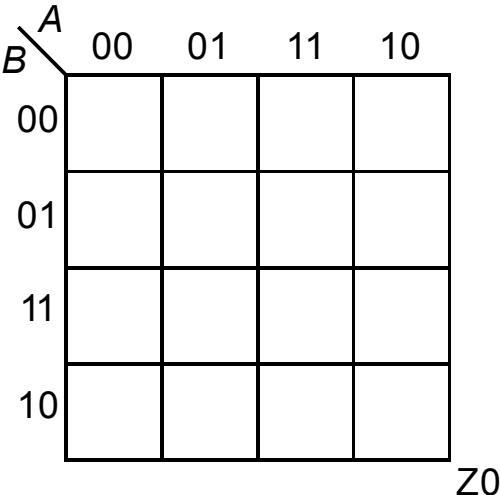
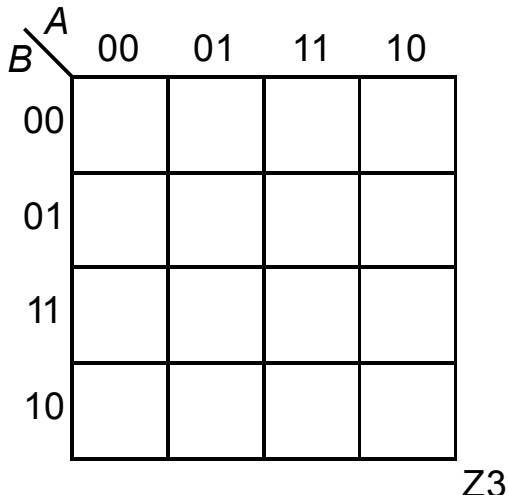
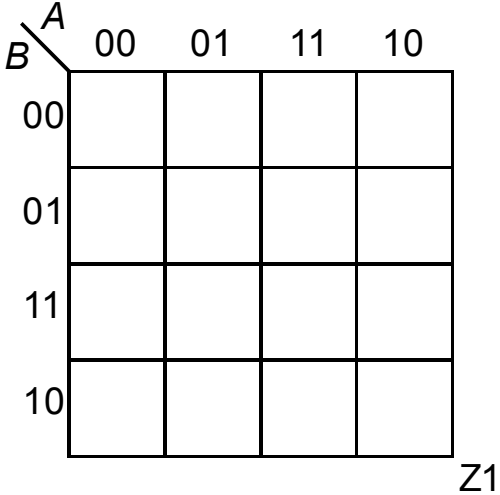
Diagram illustrating a 4x4 Karnaugh map for variables A, B, C, and D. The map shows the following values:

- Row 00 (C=0): (A=0, B=0) is X; (A=0, B=1) is 1; (A=1, B=1) is 0; (A=1, B=0) is 1.
- Row 01 (C=1): (A=0, B=0) is 0; (A=0, B=1) is 1; (A=1, B=1) is 1; (A=1, B=0) is 1.
- Row 11 (C=2): (A=0, B=0) is 0; (A=0, B=1) is X; (A=1, B=1) is X; (A=1, B=0) is 0.
- Row 10 (C=3): (A=0, B=0) is 0; (A=0, B=1) is 1; (A=1, B=1) is 0; (A=1, B=0) is 1.

Red circles highlight the 1's at (0,1), (1,1), (3,1) and (0,3), (3,3). A blue circle highlights the 1's at (1,2) and (1,3). Brackets labeled A, B, and C indicate the column, row, and column indices respectively.

design example: two-bit multiplier

A0	A1	B0	B1	Z3	Z2	Z1	Z0
0	0	0	0				
0	0	0	1				
0	0	1	0				
0	0	1	1				
0	1	0	0				
0	1	0	1				
0	1	1	0				
0	1	1	1				
1	0	0	0				
1	0	0	1				
1	0	1	0				
1	0	1	1				
1	1	0	0				
1	1	0	1				
1	1	1	0				
1	1	1	1				



Two-Bit Multiplier Design

$$Z_0 = A_0 B_0$$

$$Z_1 = \bar{A}_1 A_0 B_1 + A_0 B_1 \bar{B}_0 + A_1 \bar{B}_1 B_0 + A_1 \bar{A}_0 B_0$$

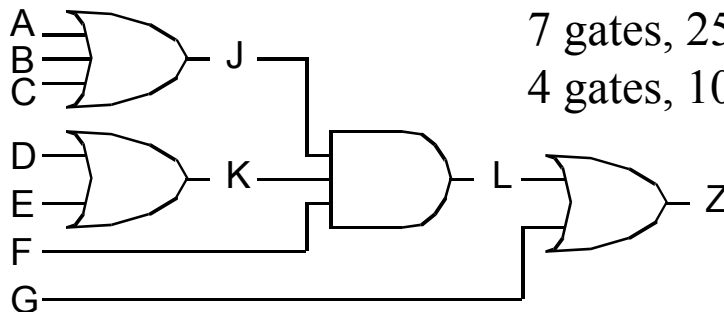
$$Z_2 = A_1 \bar{A}_0 B_1 + A_1 B_1 \bar{B}_0$$

$$Z_3 = A_1 A_0 B_1 B_0$$

Multilevel logic

- possible to further reduce resources needed by some 2-level functions if we factor out common subexpressions (factored form)
- more gate efficient at the cost of longer propagation delays
- Example:

$$\begin{aligned} Z &= ADF + AEF + BDF + BEF + CDF + CEF + G \\ &= (AD + AE + BD + BE + CD + CE)F + G \\ &= [(A+B+C)D + (A+B+C)E]F + G \\ &= (A+B+C)(D+E)F + G \\ &= (J)(K)F + G = L + G \end{aligned}$$



7 gates, 25 wires, and 2 gate delays for 2-level logic vs.
4 gates, 10 wires, and 3 gate delays for factored logic

CAD tools for 2-level boolean minimization

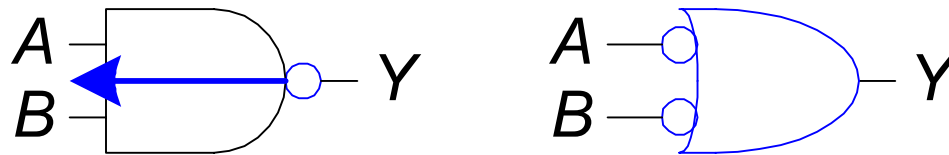
- Karnaugh map method hard to automate
- Quine-McCluskey method (1955)
 - systematic procedure for
 1. generating all prime implicants
 2. extracting minimum set of primes covering on-set
 - problems:
 1. number of prime implicants grows as $3^n/n$
 2. finding a minimum cover is NP-completeHence only works for about a dozen inputs
- Espresso (UC Berkeley program)
 - set of heuristics
 - finds minimal solution fast
 - does NOT guarantee minimum solutionBut works well for *dozens* of inputs

CAD tools for multilevel logic synthesis

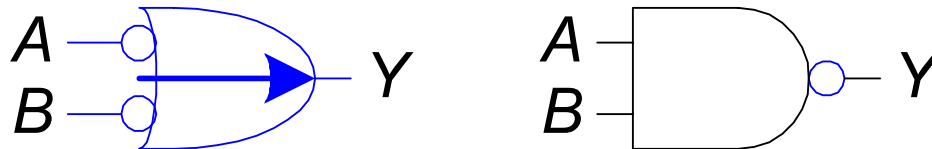
- not easy to define optimality for multilevel networks
- key idea is to identify common boolean subexpressions
- MisII (UC Berkeley program)
 - powerful tool to assist in multilevel synthesis
 - uses espresso as subroutine

Bubble Pushing

- Pushing bubbles backward (from the output) or forward (from the inputs) changes the body of the gate from AND to OR or vice versa.
- Pushing a bubble from the output back to the inputs puts bubbles on all gate inputs.

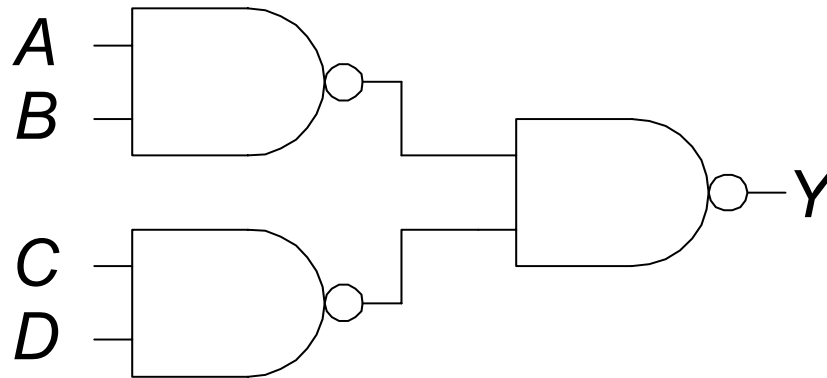


- Pushing bubbles on *all* gate inputs forward toward the output puts a bubble on the output and changes the gate body.



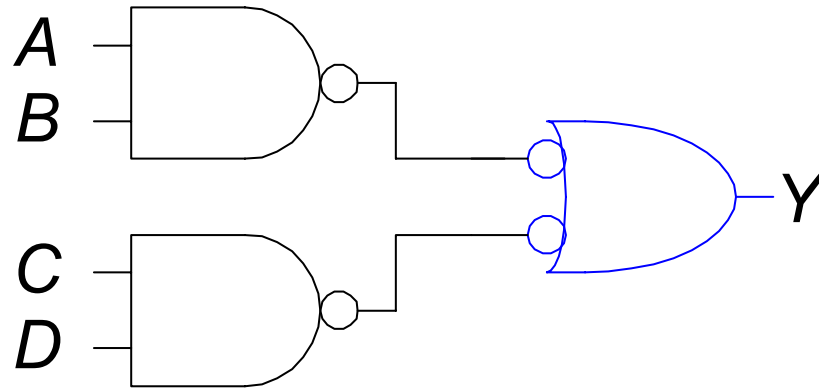
Bubble Pushing

- What is the Boolean expression for this circuit?



Bubble Pushing

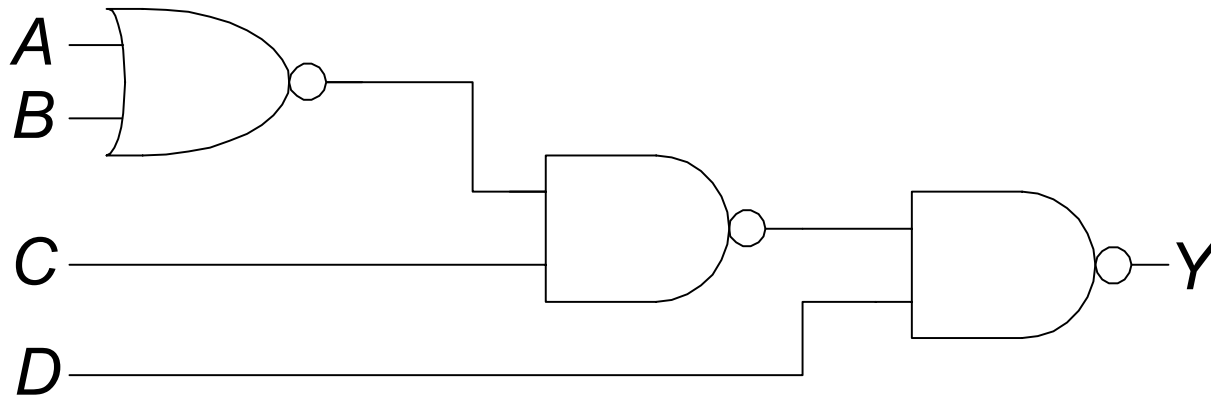
- What is the Boolean expression for this circuit?



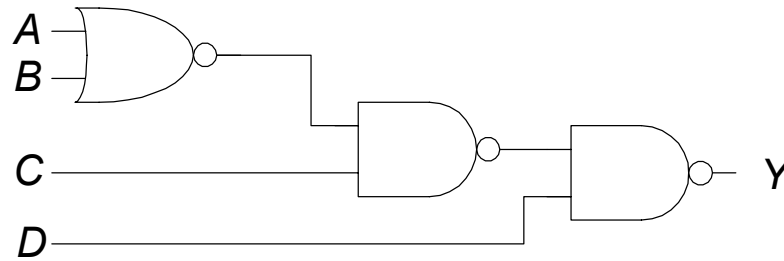
$$Y = AB + CD$$

Bubble Pushing Rules

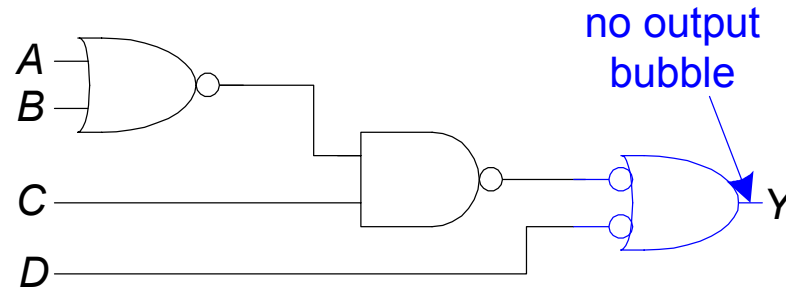
- Begin at the output of the circuit and work toward the inputs.
- Push any bubbles on the final output back toward the inputs.
- Draw each gate in a form so that bubbles cancel.



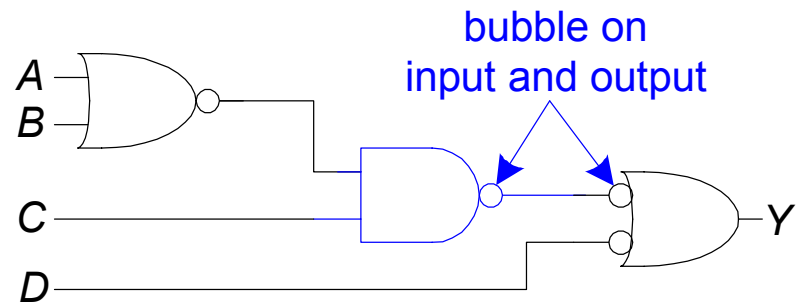
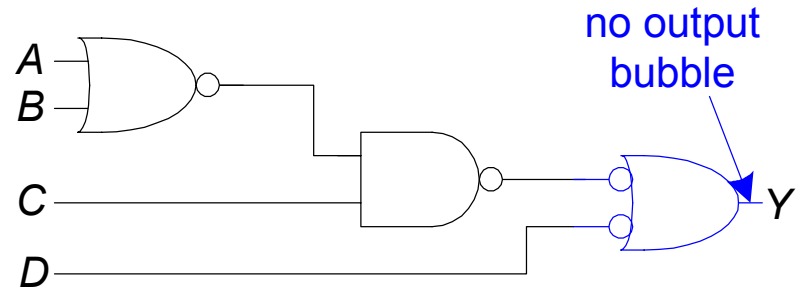
Bubble Pushing Example



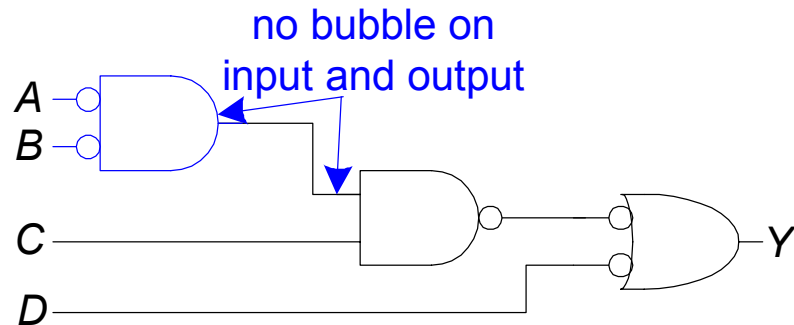
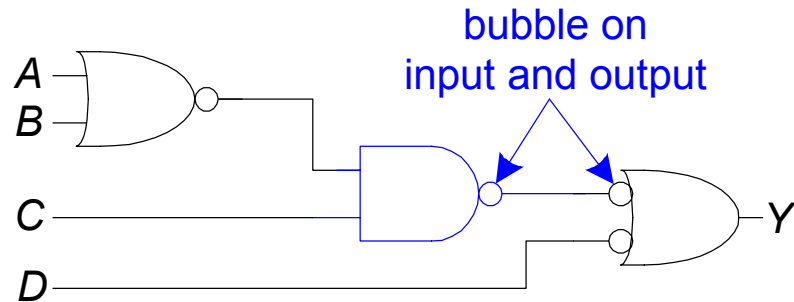
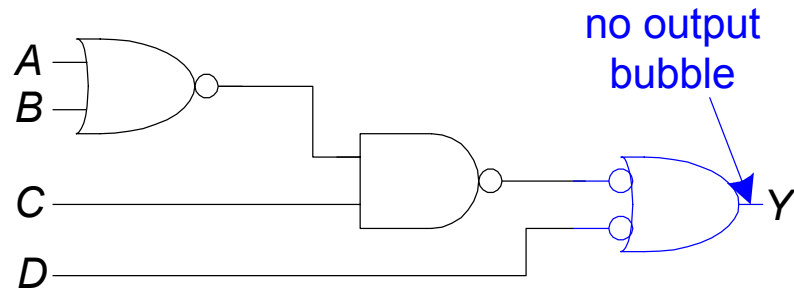
Bubble Pushing Example



Bubble Pushing Example



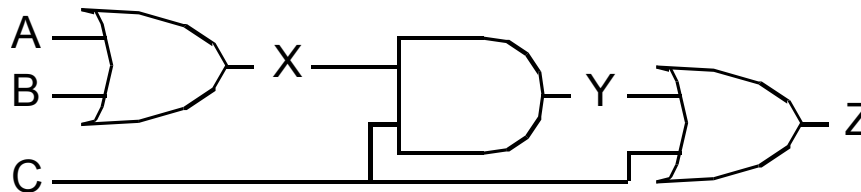
Bubble Pushing Example



$$Y = \overline{A} \overline{B} C + \overline{D}$$

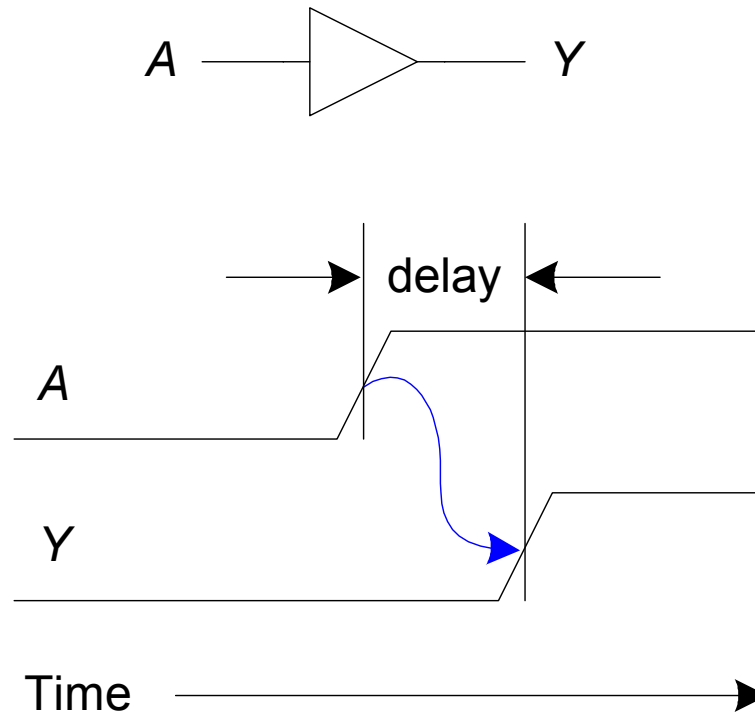
Time Response in Combinational Circuits

- Circuit analysis so far concerned only with steady state behavior
 - apply input pattern, observe output pattern (given enough time)
- Dynamic behavior important too
 - propagation delay – varies depending upon
 - path of change through circuit
 - direction of change within gate



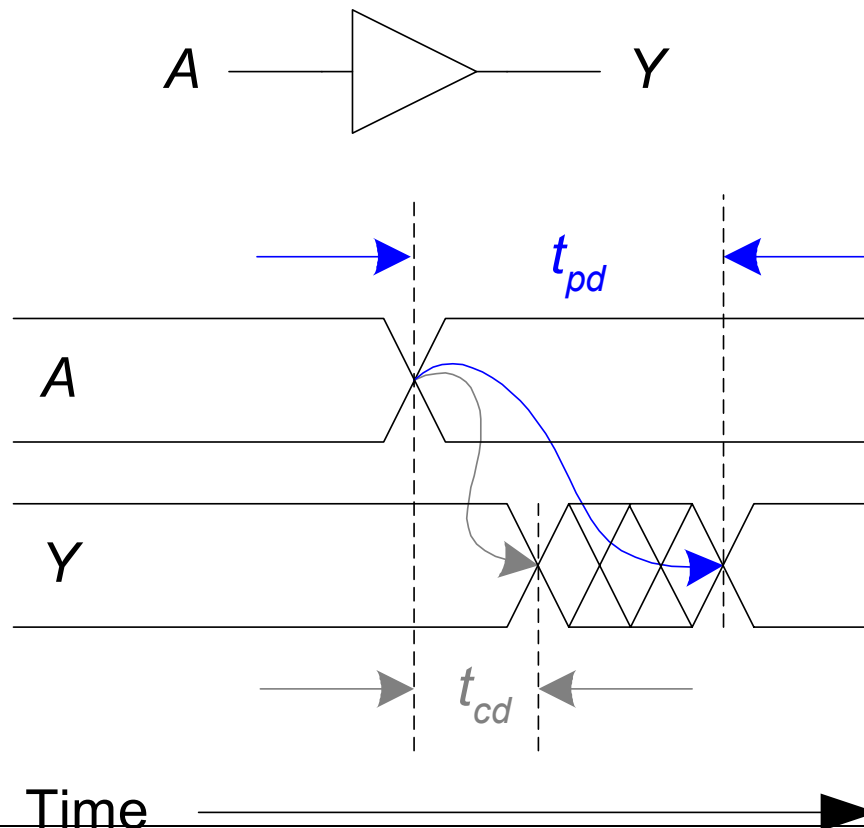
Timing

- Delay between input change and output changing
- How to build fast circuits?



Propagation & Contamination Delay

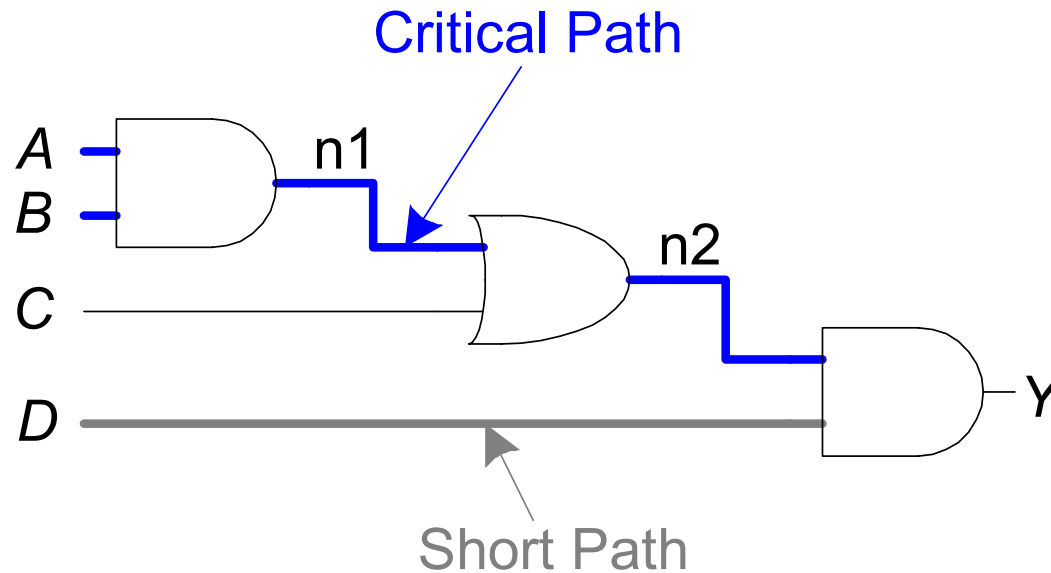
- Propagation delay: $t_{pd} = \max$ delay from input to output
- Contamination delay: $t_{cd} = \min$ delay from input to output



Propagation & Contamination Delay

- Delay is caused by
 - Capacitance and resistance in a circuit
 - Speed of light limitation
- Reasons why t_{pd} and t_{cd} may be different:
 - Different rising and falling delays
 - Multiple inputs and outputs, some of which are faster than others
 - Circuits slow down when hot and speed up when cold

Critical (Long) and Short Paths



Critical (Long) Path: $t_{pd} = 2t_{pd_AND} + t_{pd_OR}$

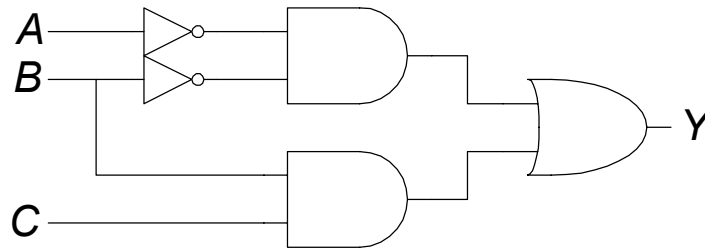
Short Path: $t_{cd} = t_{cd_AND}$

Glitches

- Glitch: when a single input change causes multiple output changes
- Glitches don't cause problems because of synchronous design conventions (which we'll talk about in a bit)
- But it's important to recognize a glitch when you see one in timing diagrams

Glitch Example

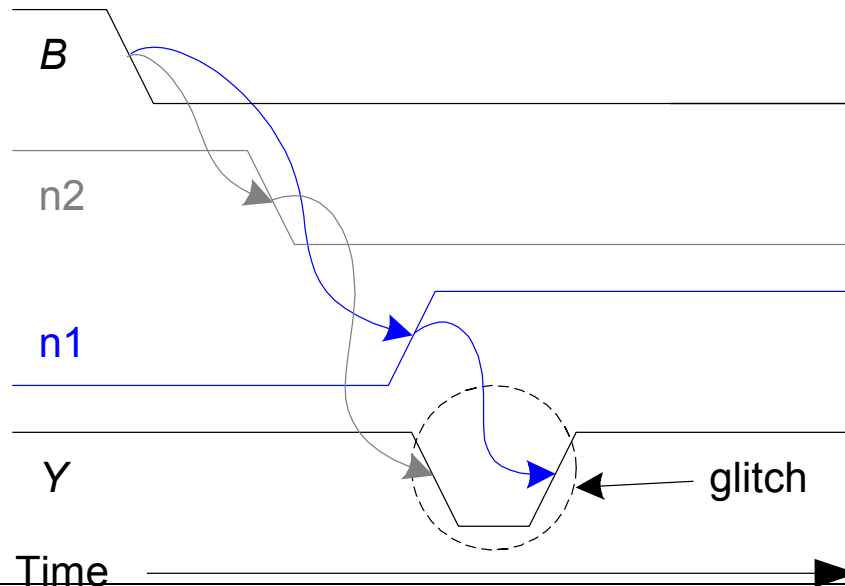
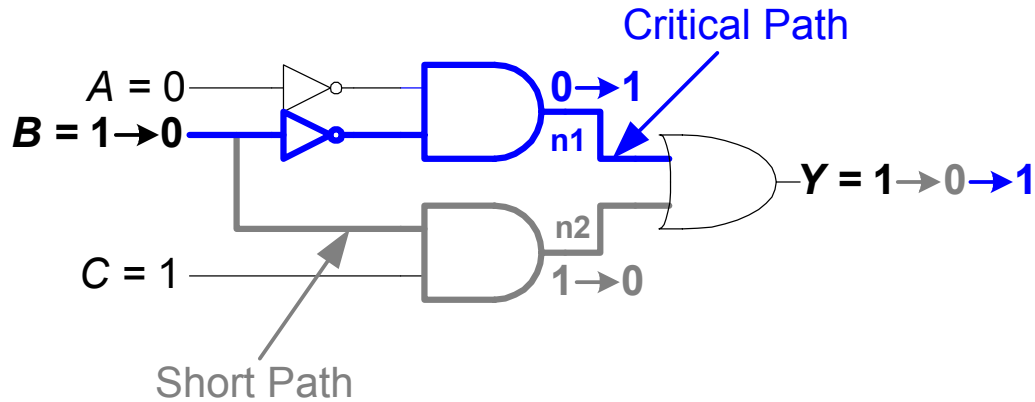
- What happens when $A = 0$, $C = 1$, B falls?



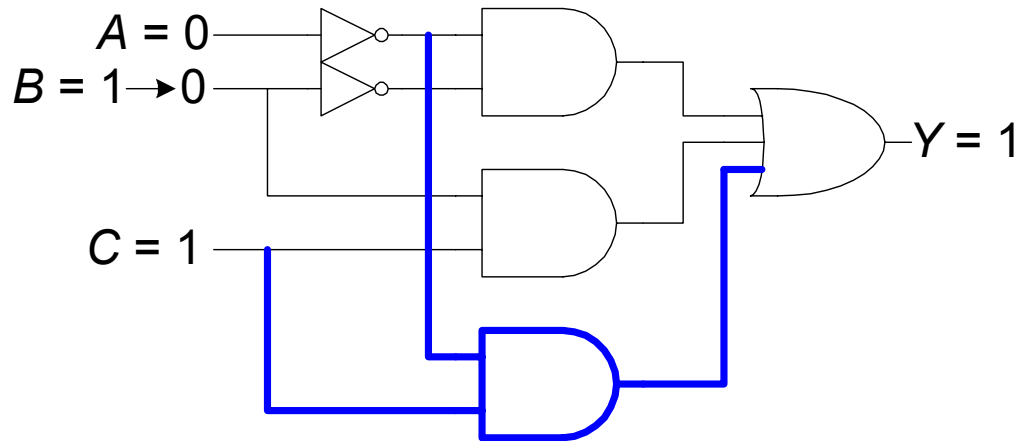
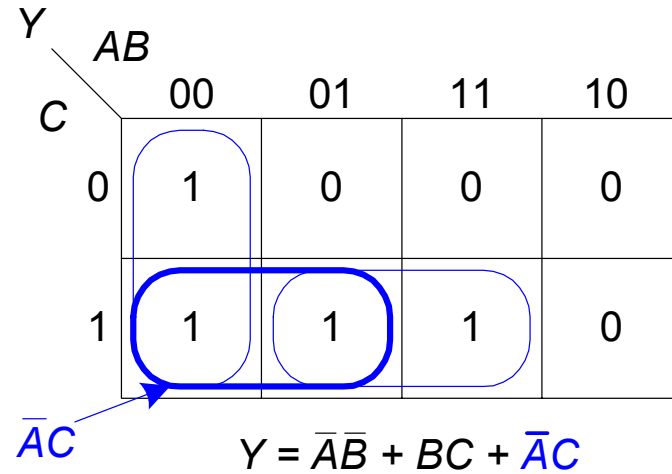
Y		AB			
		00	01	11	10
C	0	1	0	0	0
	1	1	1	1	0

$$Y = \bar{A}\bar{B} + BC$$

Glitch Example (cont.)



Fixing the Glitch



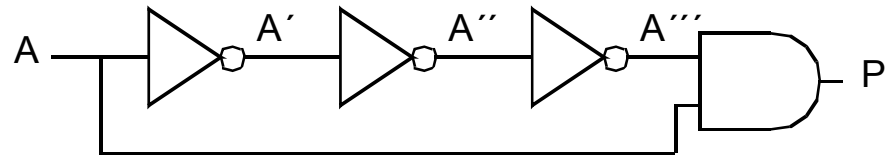
Why Understand Glitches?

- Glitches don't cause problems because of synchronous design conventions (Chapter 3)
- But it's important to recognize a glitch when you see one in simulations or on an oscilloscope
- Can't get rid of all glitches – simultaneous transitions on multiple inputs can also cause glitches

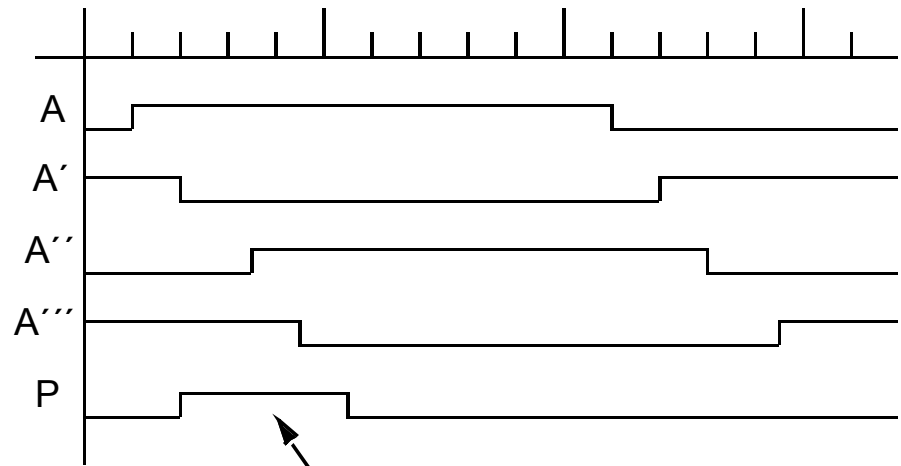
Useful circuits employing delay

1. Pulse shaping circuit

- statically boring, dynamically useful
- Takes advantage of asymmetry in delay paths
- length of pulse depends on number of inverters
- circuit not as interesting if even number of inverters



$$P = A \cdot A''' = A \cdot A' = 0$$

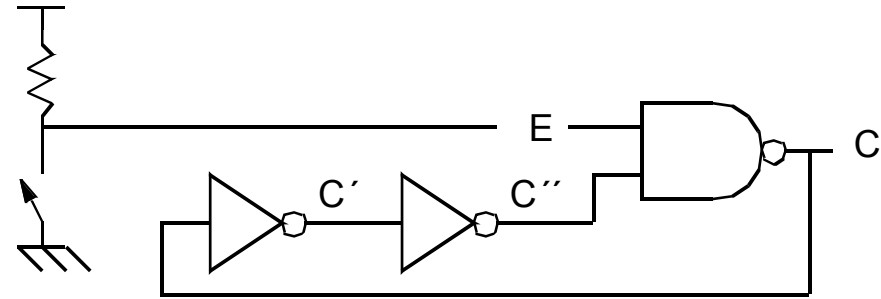


P goes high for 3 gate delays!

Useful circuits employing delay

2. Oscillator

- odd number of inverters in a loop produces a continually-running pulse generator



- add ENABLE pin to reset oscillator

