
Computer Science 141

Computing Hardware

Fall 2009
Harvard University

Instructor: Prof. David Brooks

Outline

- HW1 out soon (due Wednesday next week)
- Binary number systems
- Boolean algebra

Electronics: What you should know

- Key points to understand
 - 1) Transistors are switches
 - 2) Switches can be used to form NAND/NOR gates
 - 3) Capacitance and Resistance == Delay!

Binary Numbers

- Number systems
 - problem: mapping infinite number systems onto finite representation for use by a computer
- two main classes of operations:
 - integer operations (fixed-point operations)
 - binary quantities
 - binary-coded decimal (BCD) quantities
 - floating-point operations

Representing Integers

- fixed number of representable integers
- based on machine word size
 - e.g. if word size = 32 bits $\Rightarrow 2^{32} = 4$ billion representable integer values
 - *overflow* if we try to represent integer outside of range

Representing Integers

- 2 types of integers
 - unsigned integers
 - string of bits interpreted as positive binary number
 - used to address memory
 - e.g. $0111_2 = 7_{10}$
 $1100_2 = 12_{10}$
 - signed integers
 - split encoding space among positive and negative numbers
 - 4 ways to interpret string of binary digits, 1 is most common

Representing a signed integer

- sign and magnitude

- high-order bit is sign bit, rest of bits are magnitude

$$3_{10} = 0011_2$$

$$-3_{10} = 1011_2$$

- 2 representations of zero

$$0_{10} = 0000_2$$

$$-0_{10} = 1000_2$$

- addition of numbers with same sign simple

$$3_{10} = 0011_2$$

$$-3_{10} = 1011_2$$

$$+ 4_{10} = 0100_2$$

$$+ -4_{10} = 1100_2$$

$$\hline 7_{10} = 0111_2$$

$$\hline -7_{10} = 1111_2$$

- addition of numbers with different sign hard

$$3_{10} = 0011_2$$

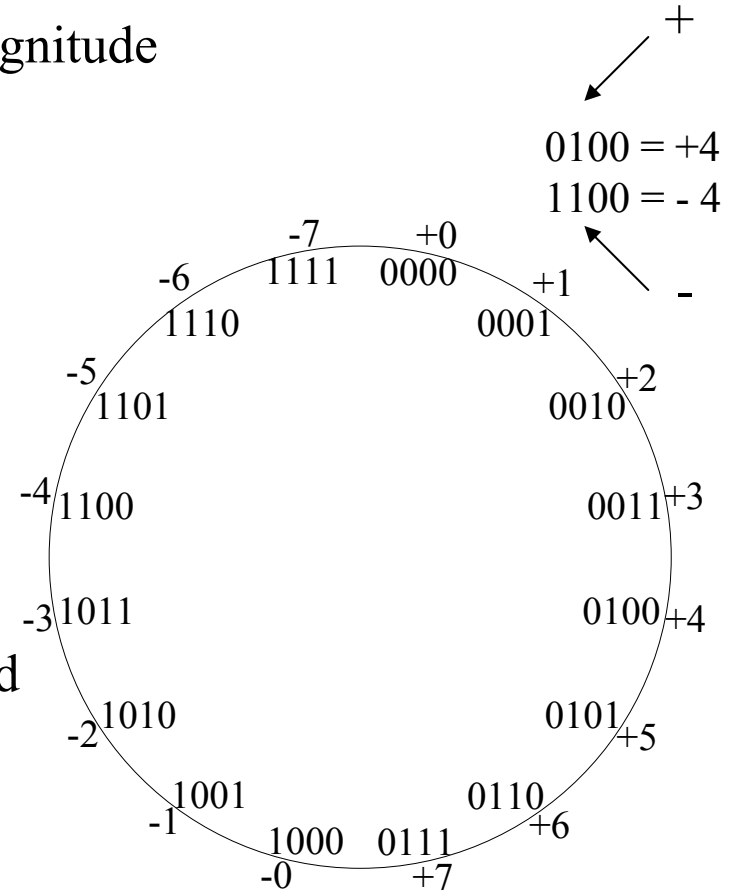
$$-3_{10} = 1011_2$$

$$+ -4_{10} = 1100_2$$

$$+ 4_{10} = 0100_2$$

$$\hline -1_{10} = 1001_2$$

$$\hline 1_{10} = 0001_2$$



⇒ need subtractor and comparator!

ones compliment

- complement bits in positive number to generate negative

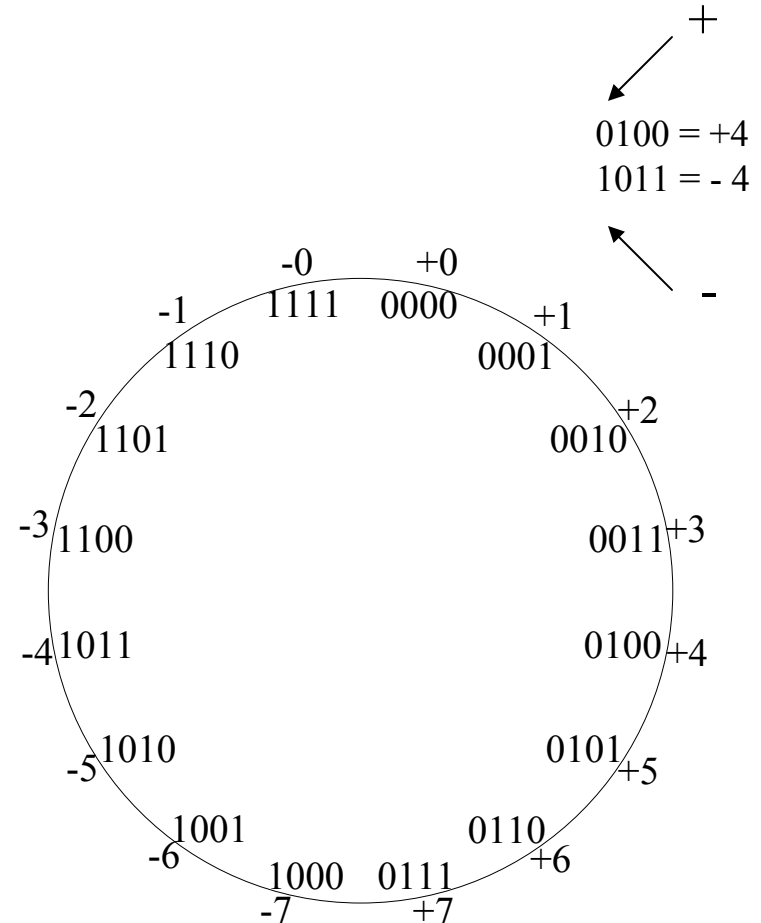
$$3_{10} = 0011_2 \qquad -3_{10} = 1100_2$$

- msb still represents “sign bit”
- still 2 representations of zero

$$0_{10} = 0000_2 \qquad -0_{10} = 1111_2$$

- addition can require end-around

$3_{10} = 0011_2$	$-3_{10} = 1100_2$
$+ 4_{10} = 0100_2$	$+ -4_{10} = 1011_2$
$7_{10} = 0111_2$	$1\ 0111_2$
	$+ \qquad\qquad 1_2$
	$-7_{10} = 0\ 1000_2$



⇒ end-around carry required to compensate for 2nd zero!

Twos compliment

- complement bits and add 1 to generate negative

$$3_{10} = 0011_2$$

$$-3_{10} = (1100_2 + 1) = 1101_2$$

- msb still represents “sign bit”

- only 1 representations of zero

$$0_{10} = 0000_2$$

$$-8_{10} = 1000_2$$

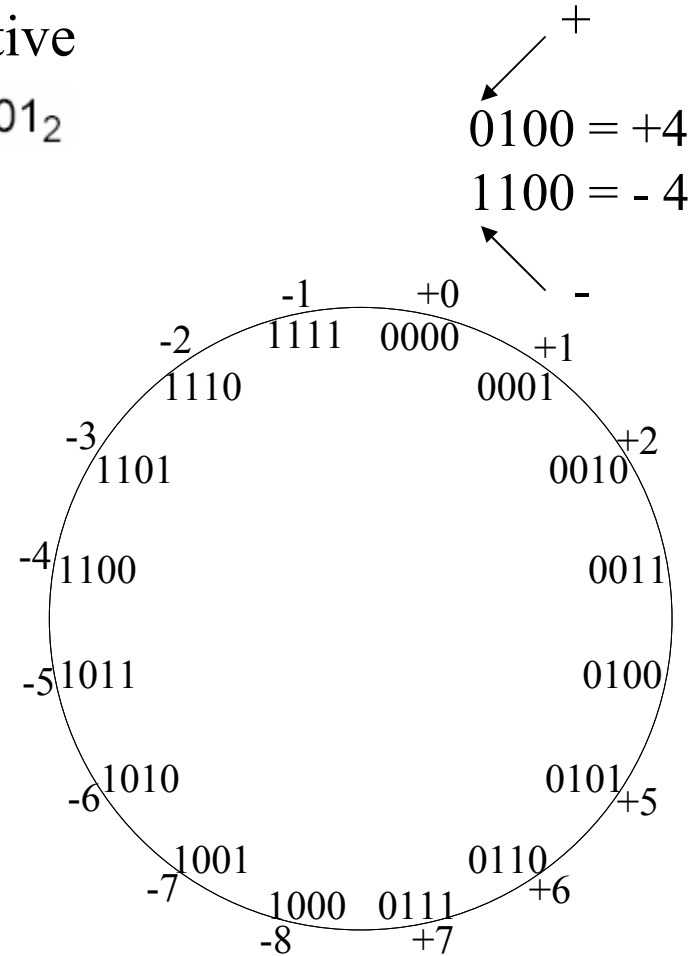
- 1 more negative number than positive!

- addition always simple

$$\begin{array}{r} 3_{10} = 0011_2 \\ + 4_{10} = 0100_2 \\ \hline 7_{10} = 0111_2 \end{array}$$

$$\begin{array}{r} -3_{10} = 1101_2 \\ + -4_{10} = 1100_2 \\ \hline -7_{10} = 1\ 1001_2 \end{array}$$

⇒ carry out ignored (more later)!

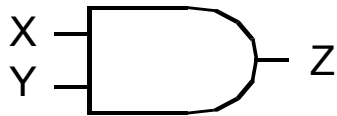


Boolean Algebra

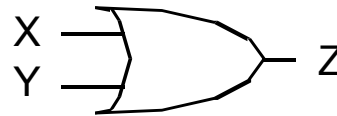
- Basis of logic minimization
- Why do logic minimization?
 - minimize amount of wiring in circuit (minimize no. of literals)
 - wiring takes space and is a difficult problem
 - physical gates have limited no. of inputs
 - minimize no. of gates in circuit
 - gates take area
 - minimal no. of gates \neq 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

Boolean Algebra

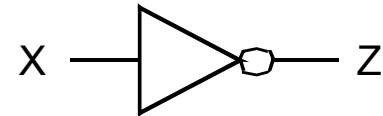
- Fundamentals
 - 2 elements $\{0,1\}$, 2 binary operators $(\cdot,+)$, 1 unary operator (')
 - all logic functions can be implemented in terms of AND (\cdot) , OR $(+)$, and NOT (') gates



X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

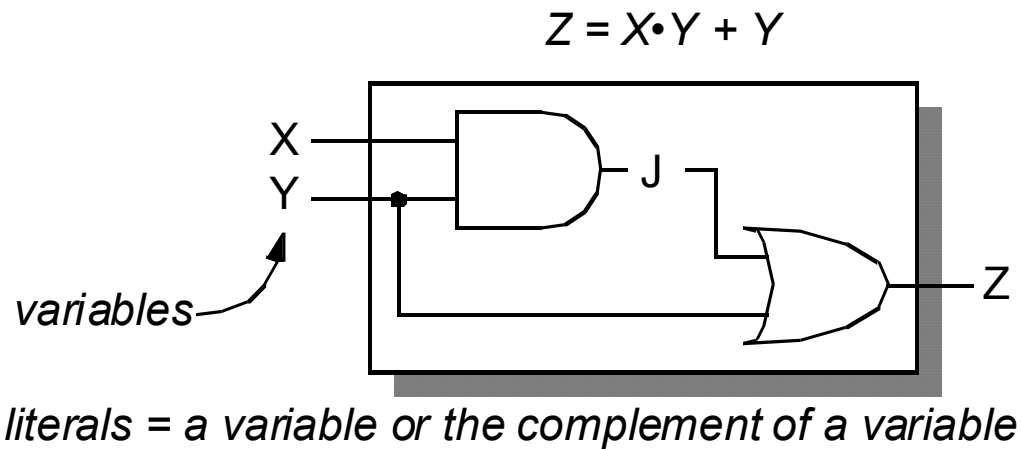


X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1



X	Z
0	1
1	0

- Terminology



Boolean operations

- overall, 16 functions of 2 variables

		<i>Function</i>															
X	Y	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

$F_0 = 0$ $F_1 = X \bullet Y$ $F_3 = X$ $F_5 = Y$ $F_7 = X + Y$
 $F_A = \bar{Y}$ $F_F = 1$ $F_C = \bar{X}$ $F_8 = \overline{X + Y}$ $F_E = \overline{X \bullet Y}$
 $F_6 = X \oplus Y$ $F_9 = \overline{X \oplus Y}$ *NOR* *NAND*
XOR *XNOR*

Boolean operations

- remaining 4 functions based on boolean operator called implication (not commonly found as primitives in digital circuits)
- NAND/NOR – each electrically common and functionally complete
- all functions not equivalent in implementation speed
- Wakerly describes a large number of laws/theorems, we'll discuss just a few of the basic, important Laws and Theorems of Boolean Algebra

Laws and Theorems of Boolean Algebra

- identities:

$$X + 0 = X \quad X \cdot 1 = X$$

$$X + 1 = 1 \quad X \cdot 0 = 0$$

- idempotence law:

$$X + X = X \quad X \cdot X = X$$

- complements law:

$$X + X' = 1 \quad X \cdot X' = 0$$

- commutative law:

$$X + Y = Y + X \quad X \cdot Y = Y \cdot X$$

- associative law:

$$(X + Y) + Z = X + (Y + Z) = X + Y + Z$$

$$(X \cdot Y) \cdot Z = X \cdot (Y \cdot Z) = X \cdot Y \cdot Z$$

Note: associative law allows us to build gates with >2 inputs

Laws and Theorems of Boolean Algebra

- distributive law:

$$X \cdot (Y+Z) = (X \cdot Y) + (X \cdot Z)$$

$$X + (Y \cdot Z) = (X+Y) \cdot (X+Z)$$

Note: distributive law is *not* the same as it is for integer ops

- duality – every boolean function has a dual

- algorithm:

1. change ANDs to ORs and vice versa
2. change logic 0's to logic 1's and vice versa
3. leave literals unchanged

- $\{f(X_1, X_2, \dots, X_n, 0, 1, +, \cdot)\}^D = f(X_1, X_2, \dots, X_n, 1, 0, \cdot, +)$

- if expression is true, then dual is true

Proving Distributive Law

$$X \cdot (Y+Z) = (X \cdot Y) + (X \cdot Z)$$

X	Y	Z	(Y+Z)	$X \cdot (Y+Z)$	$(X \cdot Y)$	$(X \cdot Z)$	$(X \cdot Y) + (X \cdot Z)$
0	0	0					
0	0	1					
0	1	0					
0	1	1					
1	0	0					
1	0	1					
1	1	0					
1	1	1					

Proving Distributive and Covering Theorems

$$(X+Y) \cdot (X+Z) = X + (Y \cdot Z)$$

Proving Covering Theorem

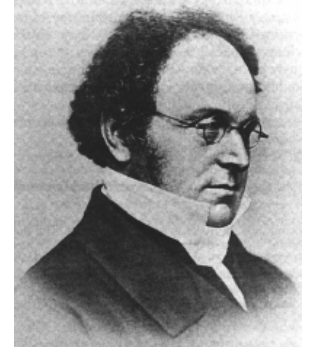
$$X + X \cdot Z = X$$

DeMorgan's Theorem

- DeMorgan's theorem: procedure for complementing a complex function.

$$(1) \quad \overline{(X + Y)} = \bar{X} \bullet \bar{Y}$$

$$(2) \quad \overline{X \bullet Y} = \bar{X} + \bar{Y}$$



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

General Form of DeMorgan's

- Generalizes to n variables

$$(1) \overline{(X + Y + Z + \dots)} = \bar{X} \bullet \bar{Y} \bullet \bar{Z} \dots$$

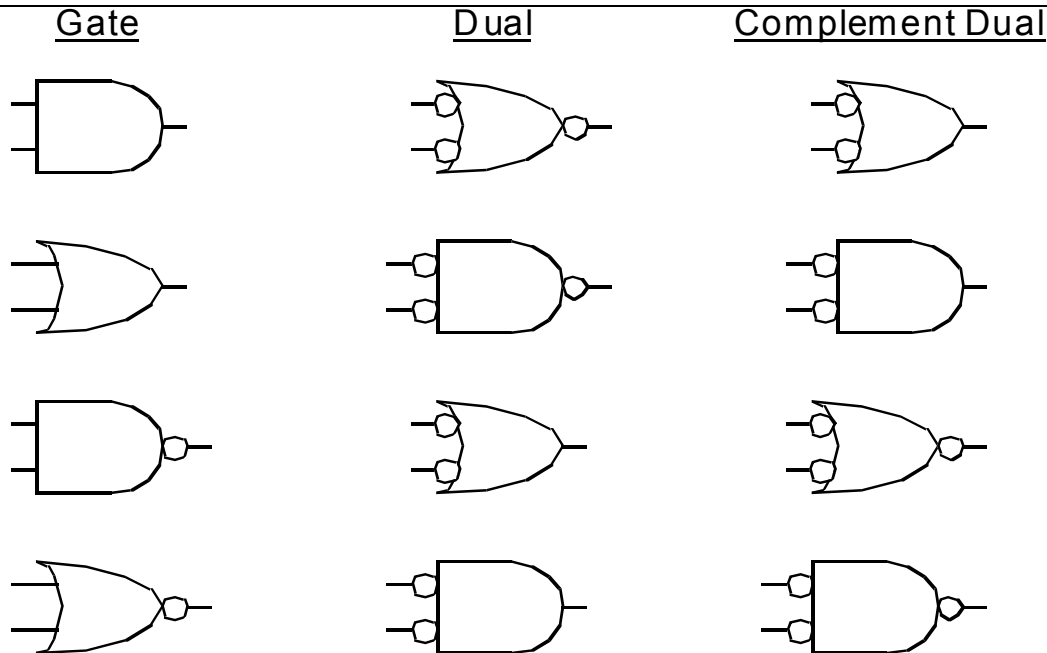
$$(2) \overline{\bar{X} \bullet \bar{Y} \bullet \bar{Z} \dots} = \bar{\bar{X}} + \bar{\bar{Y}} + \bar{\bar{Z}} + \dots$$

– algorithm:

1. change ANDs to ORs and vice versa
2. change logic 0's to logic 1's and vice versa
3. complement each literal

- $\{f(X_1, X_2, \dots, X_n, 0, 1, +, \bullet)\}' = f(X_1', X_2', \dots, X_n', 1, 0, \bullet, +)$
- duality theorem \neq DeMorgan's theorem

compliments and duals



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

- pushing bubbles – we'll see this trick often...

More terminology

- 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	$m_7 = ABC$

$$F(A,B,C) = \overline{ABC} + \overline{AB}C + \overline{A}BC + A\bar{B}\bar{C} + ABC$$

$$= m_3 + m_4 + m_5 + m_6 + m_7 = \sum_{A,B,C} (3,4,5,6,7)$$

- unique, but not simplest in terms of resources or delay

$$\longrightarrow \overline{F(A,B,C)} = \sum_{A,B,C} (0,1,2)$$

Reducing F

- simplifying:

$$\begin{aligned}F(A,B,C) &= \overline{A}BC + A\overline{B}\overline{C} + A\overline{B}C + A\overline{B}\overline{C} + ABC \\ &= m_3 + m_4 + m_5 + m_6 + m_7 = \sum_{A,B,C} (3,4,5,6,7) \\ &= \overline{A}BC + AB(\overline{C} + C) + AB(\overline{C} + C) \\ &= \overline{A}BC + AB(1) + AB(1) \\ &= \overline{A}BC + A(\overline{B} + B) \\ &= \overline{A}BC + A(1) \\ &= BC + A \\ &= A + BC\end{aligned}$$

product terms



Simplifying Theorem

$$X + X'Y = X + Y$$

Recall distributive theorem

$$X + YZ = (X + Y)(X + Z)$$

Product of Sums

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

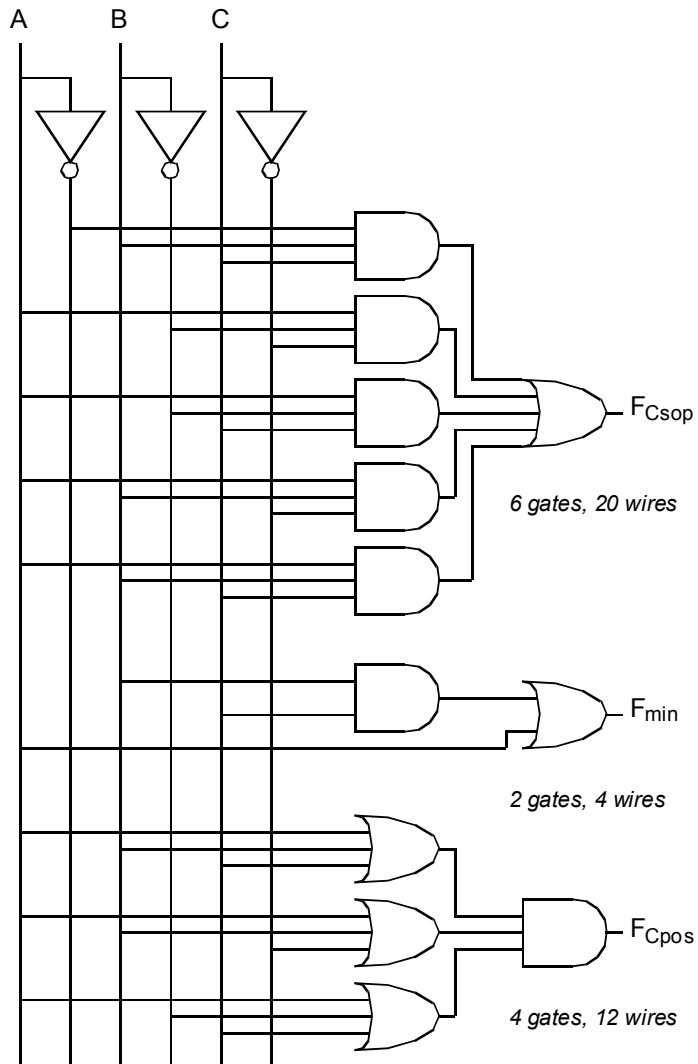
A	B	C	F	\bar{F}	maxterm
0	0	0	0	1	$m_0 = A + B + C$
0	0	1	0	1	$m_1 = A + B + \bar{C}$
0	1	0	0	1	$m_2 = A + \bar{B} + C$
0	1	1	1	0	$m_3 = A + \bar{B} + \bar{C}$
1	0	0	1	0	$m_4 = \bar{A} + B + C$
1	0	1	1	0	$m_5 = \bar{A} + B + \bar{C}$
1	1	0	1	0	$m_6 = \bar{A} + \bar{B} + C$
1	1	1	1	0	$m_7 = \bar{A} + \bar{B} + \bar{C}$

$$F(A,B,C) = (A + B + C)(A + B + \bar{C})(A + \bar{B} + C) = m_0 \bullet m_1 \bullet m_2 = \prod M(0,1,2)$$

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

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

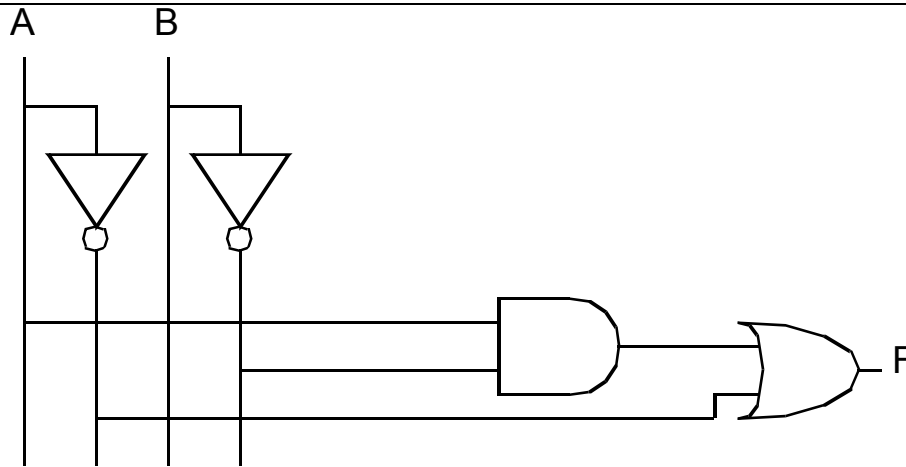
Gate-level implementation



- **NOTE:**

- gate count does not include inverters
- wire count does not include output

Combinational Circuit Analysis



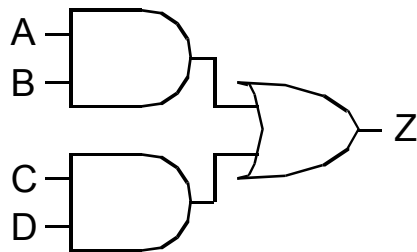
- Exhaustive Enumeration (quickly becomes intractable)
- Algebraic Manipulation

Combinational Circuit Synthesis

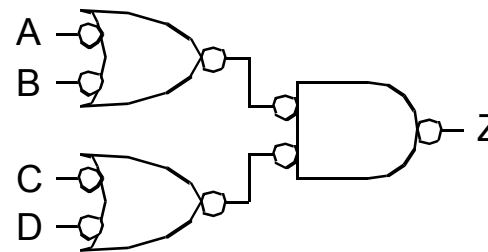
- What occurs after formalization of the circuit description?
- Includes
 - circuit manipulation—pushing bubbles (next page)
 - circuit minimization plays a large role

Conversion for technology-dependent gates

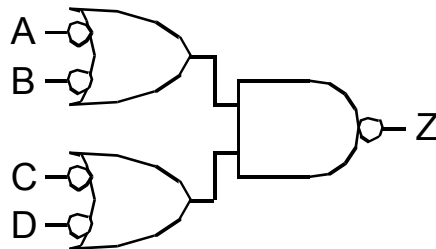
- electronically easier to build inverting gates
- need to convert AND/OR networks into NAND/NAND or NOR/NOR networks
- NAND/NAND networks



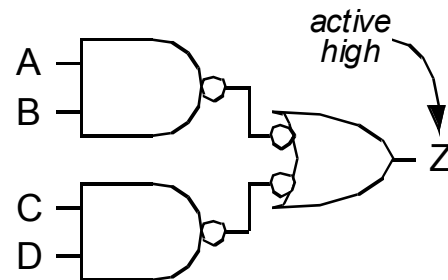
(i) $Z = A \cdot B + C \cdot D$



(ii) Replace gates with duals.



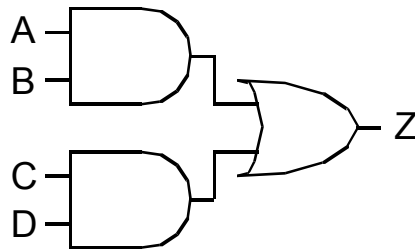
(iii) Drop extra inversions.
NAND/NAND circuit.



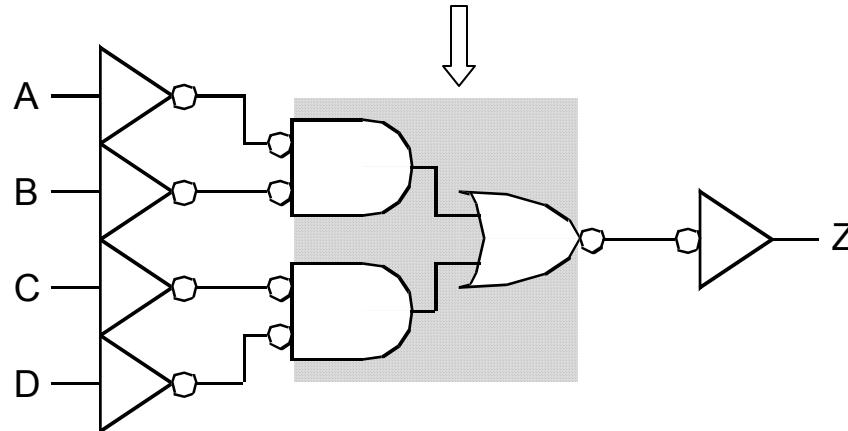
(iv) Push bubbles thru gate.
Match polarities.

Conversion for technology-dependent gates

- NOR/NOR networks
 - simple replacement not logically equivalent (OR-AND)
 - add inverter at input and at output
- Similar conversions for OR/AND networks



$$(i) Z = A \cdot B + C \cdot D$$



Algebraic Boolean Simplification

- Three major approaches to apply laws/theorems of boolean algebra to simplify the function
- Grouping

$$A + AB + BC$$

$$A(1 + B) + BC$$

$$1 + B = 1$$

$$A + BC$$

Algebraic Boolean Simplification

- Multiplication of redundant variables
 - Can apply identity function without changing logic function
 - Sometimes making *adding* terms to the function makes it easier to simplify

$$\begin{aligned}AB + A\bar{C} + BC &= AB(C + \bar{C}) + A\bar{C} + BC \\ &= ABC + AB\bar{C} + A\bar{C} + BC \\ &= BC(1 + A) + A\bar{C}(1 + B) \\ &= BC + A\bar{C}\end{aligned}$$

Algebraic Boolean Simplification

- Application of DeMorgan's Theorem
 - Expressions with multiple inversions can *sometimes* be greatly simplified with DeMorgan's Theorem

$$\begin{aligned}\overline{\overline{ABC} + \overline{ACD} + \overline{BC}} &= \overline{(\overline{A} + B + \overline{C}) + (\overline{A} + \overline{C} + \overline{D}) + \overline{BC}} \\ &= \overline{(\overline{A} + B + \overline{C} + \overline{D}) + \overline{BC}} \\ &= \overline{(\overline{A} + B + \overline{C} + \overline{D})} \\ &= \overline{A} \overline{B} C D\end{aligned}$$

Algebraic Boolean Simplification

- Simplification using algebraic rules is
 - ad hoc approach
 - When do you stop trying to apply rules?
 - When should you make the expression more complicated?
 - not guaranteed to find simpler expression
- Next Time:
 - Develop a more systematic approach! (towards automated tools)