

# Combinational logic

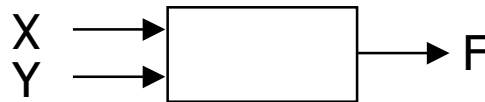
---

- ❑ Basic 2-input functions
  - NOT, AND, OR, NAND, NOR, XOR, . . .
  - **Laws of boolean logic**
- ❑ Theorems for logic simplification
  - Proofs
  - Applications
- ❑ Gate logic
  - networks of Boolean functions
  - time behavior
- ❑ Standard Representations
  - two-level canonical forms
  - incompletely Tools for Logic Simplification specified functions
  
  - Boolean cubes and Karnaugh maps
  - two-level simplification

# Possible logic functions of two variables

- There are 16 possible functions of 2 input variables:
  - in general, there are  $2^{(2^n)}$  functions of  $n$  inputs
    - 8 inputs =  $2^{2^8} = 2^{256} =$  **about a google**

Where do we start?



X	Y	16 possible functions (F0-F15)																
0	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
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	1

Labels for the 16 functions (F0-F15) from left to right:
   
 0, X and Y, X, Y, X xor Y, X or Y, X nor Y, not (X or Y), X = Y, xnor, not Y, not X, X nand Y, not (X and Y), 1

# An algebraic structure: A set of “natural laws”

□ A boolean algebraic structure consists of

- a set of elements (constants)  $B = \{0,1\}$
- binary operations  $\{ + , \bullet \}$
- and a unary operation  $\{ ' \}$
- such that the following **axioms** hold:

- |                  |   |   |
|------------------|---|---|
| 1. closure:      | $a + b$ is in $B$                             | $a \bullet b$ is in $B$                             |
| 2. commutative:  | $a + b = b + a$                               | $a \bullet b = b \bullet a$                         |
| 3. associative:  | $a + (b + c) = (a + b) + c$                   | $a \bullet (b \bullet c) = (a \bullet b) \bullet c$ |
| 4. Identity:     | $a + 0 = a$                                   | $a \bullet 1 = a$                                   |
|                  | $a + 1 = 1$                                   | $a \bullet 0 = 0$                                   |
| 5. distributive: | $a + (b \bullet c) = (a + b) \bullet (a + c)$ | $a \bullet (b + c) = (a \bullet b) + (a \bullet c)$ |
| 6. complement:   | $a + a' = 1$                                  | $a \bullet a' = 0$                                  |

**Defines AND, OR, NOT only ( $\bullet, +, '$ )**

**Notice, no definition of XOR,  $\oplus, *$ ...**

**Our question: Is this simple system powerful enough to build any digital system?**

# Completeness

- All 2-input functions can be implemented with AND,OR,NOT
  - There are few enough that we can test them all

**F1    F4    F9**  
**AND NOR    XNOR**

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

$$(X \bullet Y) + (X' \bullet Y') \equiv X = Y$$

Boolean expression that is true when the variables X and Y have the same value and false, otherwise

X, Y are Boolean algebra variables

- What about n-input functions?

# Proof by Construction

- Given a truth table w/ inputs  $I = \langle i_1, \dots, i_n \rangle$  and output  $O$  (we can do each output separately), here is a method to build  $O$ :

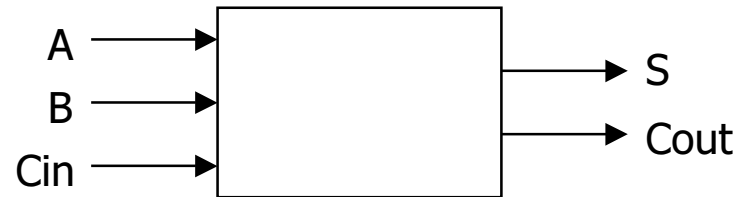
i1	i2	i3	i4	O
0	0	0	1	1
0	0	1	1	1
1	0	1	0	1
0	1	1	0	1
else				0

- Satisfy yourself that  $F \Leftrightarrow O$
- AND, OR, NOT are sufficient to implement any function

# An Example: The binary adder

## □ 1-bit binary adder

- inputs: A, B, Carry-in
- outputs: Sum, Carry-out



A	B	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$S = A' B' Cin + A' B Cin' + A B' Cin' + A B Cin$$

$$Cout = A' B Cin + A B' Cin + A B Cin' + A B Cin$$

Consider only carry:  $6 + 24 = 30$  transistors

**Can we do better than this?**

## Prove some Simplification Theorems

- ❑ Consensus Theorem:  $XY + YZ + X'Z = XY + X'Z$
- ❑ Let  $F = XY + YZ + X'Z$ , and Let  $G = XY + X'Z$
- ❑ Assume  $F \neq G$ , then try to find a contradiction
  - First prove  $G=1 \rightarrow F=1$ 
    - By Assoc.  $F = (XY+X'Z)+YZ$  so  $F = (G)+YZ$
    - $G=1$  so  $F = 1+YZ$
    - By Identity  $F = 1$
  - Then prove  $F=1 \rightarrow G=1$ 
    - By contradiction, assume  $F=1$  and  $G=0$
    - $F = G+YZ$  so  $F = (0+YZ)$
    - By Identity  $F = YZ$  so  $Y=1$  and  $Z=1$
    - So  $G = (X1 + X'1) = 0$
    - By Identity  $(X + X') = 0$ , contradicts Complement Axiom
- ❑ There are no values of XYZ such that  $F=1$  and  $G=0$  QED.
- ❑ This is called the consensus theorem and it is quite useful...

# Useful Theorems based on Axioms

7. idempotency:

$$X + X = X$$

$$X \cdot X = X$$

8. involution:

$$(X')' = X$$

9. uniting:

$$X \cdot Y + X \cdot Y' = X$$

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

10. absorption:

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

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

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

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

11. factoring:

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

$$X \cdot Z + X' \cdot Y$$

$$X \cdot Y + X' \cdot Z =$$

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

12. consensus:

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

$$X \cdot Y + X' \cdot Z$$

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

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

13. de Morgan's:

$$(X + Y + \dots)' = X' \cdot Y' \cdot \dots$$

$$(X \cdot Y \cdot \dots)' = X' + Y' + \dots$$

14. generalized de Morgan's:

$$f'(X_1, X_2, \dots, X_n, 0, 1, +, \cdot) = f(X_1', X_2', \dots, X_n', 1, 0, \cdot, +)$$



# DeMorgan's Theorem

□  $F = (a + b)$

➤ By DeMorgan's Theorem:  $F' = (a' b')$

<b>a b</b>	<b>F</b>	<b>a'b'</b>	<b>F'</b>
<b>00</b>	<b>0</b>	<b>11</b>	<b>1</b>
<b>01</b>	<b>1</b>	<b>10</b>	<b>0</b>
<b>10</b>	<b>1</b>	<b>01</b>	<b>0</b>
<b>11</b>	<b>1</b>	<b>00</b>	<b>0</b>

□ Why DeMorgan's is important

➤ Convert and-or to nand-nand logic

- $F = (a + b)$  so  $F = ((a+b)')'$
- $F = ((a'b'))'$
- $F = (a'b')'$  which is (a' nand b')

# Duality Theorem

## □ Duality

- a dual of a Boolean expression is derived by replacing
  - by +, + by •, 0 by 1, and 1 by 0, and leaving variables unchanged
- any theorem that can be proven is thus also proven for its dual!
- a meta-theorem (a theorem about theorems)

## □ Or

$$T(X_1, X_2, \dots, X_n, 0, 1, +, \bullet) \Leftrightarrow T(X_1, X_2, \dots, X_n, 1, 0, \bullet, +)$$
$$[(a \bullet a') = 0] \Leftrightarrow [(a + a') = 1]$$

## □ Different than deMorgan's Law

- this is a statement about theorems
- this is not a way to manipulate (re-write) expressions

# Duals of Useful Theorems

7. idempotency:

$$X + X = X$$

$$X \cdot X = X$$

8. involution:

$$(X')' = X$$

9. uniting:

$$X \cdot Y + X \cdot Y' = X$$

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

10. absorption:

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

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

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

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

11. factoring:

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

$$X \cdot Z + X' \cdot Y$$

$$X \cdot Y + X' \cdot Z =$$

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

12. consensus:

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

$$X \cdot Y + X' \cdot Z$$

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

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

13. de Morgan's:

$$(X + Y + \dots)' = X' \cdot Y' \cdot \dots$$

$$(X \cdot Y \cdot \dots)' = X' + Y' + \dots$$

14. generalized de Morgan's:

$$f'(X_1, X_2, \dots, X_n, 0, 1, +, \cdot) = f(X_1', X_2', \dots, X_n', 1, 0, \cdot, +)$$

# Proof by deduction

□ Using the axioms of Boolean algebra:

➤ e.g., prove the theorem:  $X \cdot Y + X \cdot Y' = X$

distributivity (8)	$X \cdot Y + X \cdot Y'$	=	$X \cdot (Y + Y')$
complementarity (5)	$X \cdot (Y + Y')$	=	$X \cdot (1)$
identity (1D)	$X \cdot (1)$	=	$X \Rightarrow$

➤ e.g., prove the theorem:  $X + X \cdot Y = X$

identity (1D)	$X + X \cdot Y$	=	$X \cdot 1 + X \cdot Y$
distributivity (8)	$X \cdot 1 + X \cdot Y$	=	$X \cdot (1 + Y)$
identity (2)	$X \cdot (1 + Y)$	=	$X \cdot (1)$
identity (1D)	$X \cdot (1)$	=	$X \Rightarrow$

# Proof by enumeration (show all cases)

□ Use complete truth table to show all cases:

➤ e.g., de Morgan's:

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

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

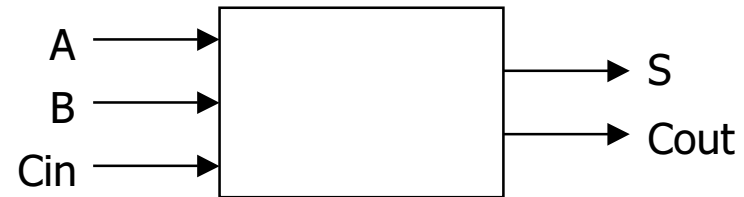
$(X \cdot Y)' = X' + Y'$   
NAND is equivalent to OR  
with inputs complemented

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

# Back to our Problem

## □ 1-bit binary adder

- inputs: A, B, Carry-in
- outputs: Sum, Carry-out



A	B	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$S = A' B' C_{in} + A' B C_{in}' + A B' C_{in}' + A B C_{in}$$

$$C_{out} = A' B C_{in} + A B' C_{in} + A B C_{in}' + A B C_{in}$$

In CMOS → 42 transistors!!

**Can we do better than this?**

# Apply the theorems to simplify expressions

- The theorems of Boolean algebra can simplify Boolean expressions
  - e.g., full adder's carry-out function (same rules apply to any function)

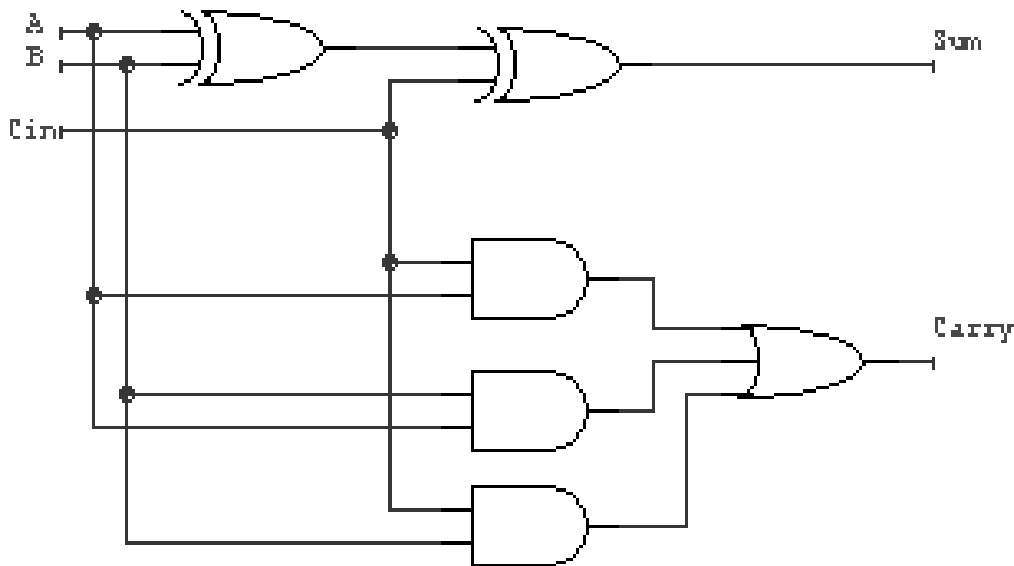
$$\begin{aligned} \text{Cout} &= A' B \text{Cin} + A B' \text{Cin} + A B \text{Cin}' + A B \text{Cin} \\ &= A' B \text{Cin} + A B' \text{Cin} + A B \text{Cin}' + A B \text{Cin} + A B \text{Cin} \\ &= A' (B \text{Cin}) + A (B \text{Cin}) + A B' \text{Cin} + A B \text{Cin}' + A B \text{Cin} \\ &= B \text{Cin} + A B' \text{Cin} + A B \text{Cin}' + A B \text{Cin} + A B \text{Cin} \\ &= B \text{Cin} + B'(A\text{Cin}) + B(A\text{Cin}) + (A B)\text{Cin}' + (A B)\text{Cin} \\ &= B \text{Cin} + A \text{Cin} + A B \end{aligned}$$

From 30 to 18 transistors

## Apply the theorems to simplify expressions

$$\begin{aligned}\text{Sum} &= A' B' \text{Cin} + A' B \text{Cin}' + A B' \text{Cin}' + A B \text{Cin} \\ &= \text{Cin}' (A'B + AB') + \text{Cin}(AB + A'B') \\ &= \text{Cin}'(A \oplus B) + \text{Cin}(A \oplus B)' \\ &= \text{Cin} \oplus (A \oplus B)\end{aligned}$$

But we have we saved any area for sum??



**Useful for  
The next  
homework**

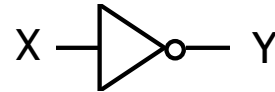


# Try it for MUTEX

RegA	ReqB	s1	s0	s1*	s0*
0	0	0	0	0	0
0	1	0	0	0	1
1	0	0	0	1	0
1	1	0	0	1	0
0	0	1	0	0	0
0	1	1	0	0	1
1	0	1	0	1	0
1	1	1	0	1	0
0	0	0	1	0	0
0	1	0	1	0	1
1	0	0	1	1	0
1	1	0	1	0	1
0	0	1	1	0	0
0	1	1	1	0	1
1	0	1	1	1	0
1	1	1	1	1	0

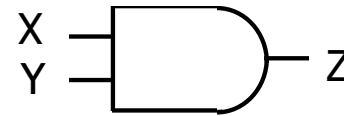
# From Boolean expressions to logic gates

□ NOT     $X'$      $X$      $\sim X$



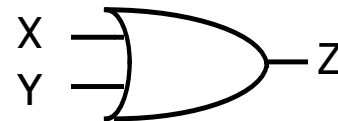
X	Y
0	1
1	0

□ AND     $X \cdot Y$      $XY$      $X \wedge Y$



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

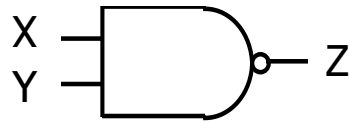
□ OR     $X + Y$      $X \vee Y$



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

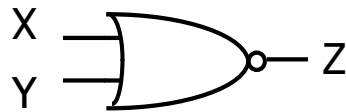
# From Boolean expressions to logic gates

□ NAND



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

□ NOR



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

$X \text{ xor } Y = X Y' + X' Y$   
 X or Y but not both  
 ("inequality", "difference")

□ XOR

$X \oplus Y$



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

$X \text{ xnor } Y = X Y + X' Y'$   
 X and Y are the same  
 ("equality", "coincidence")

□ XNOR

$X = Y$



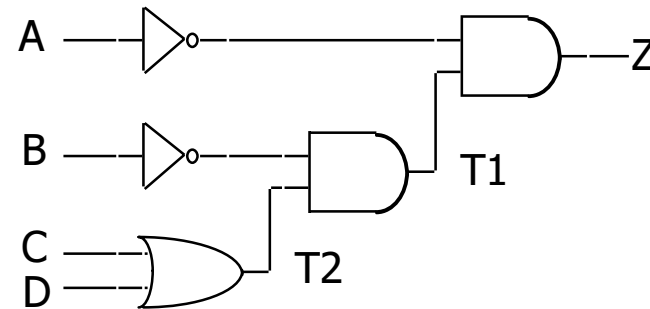
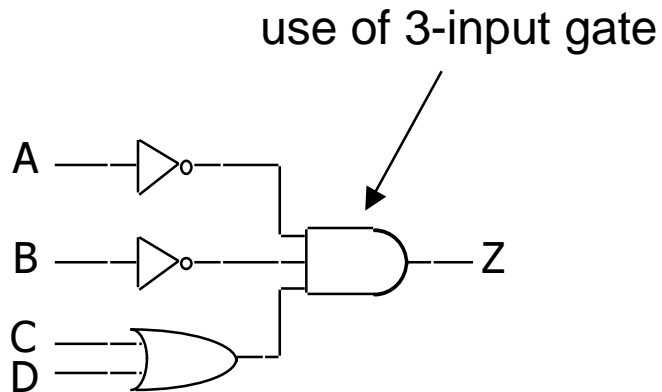
X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	1

# From Boolean expressions to logic gates (cont'd)

□ More than one way to map expressions to gates

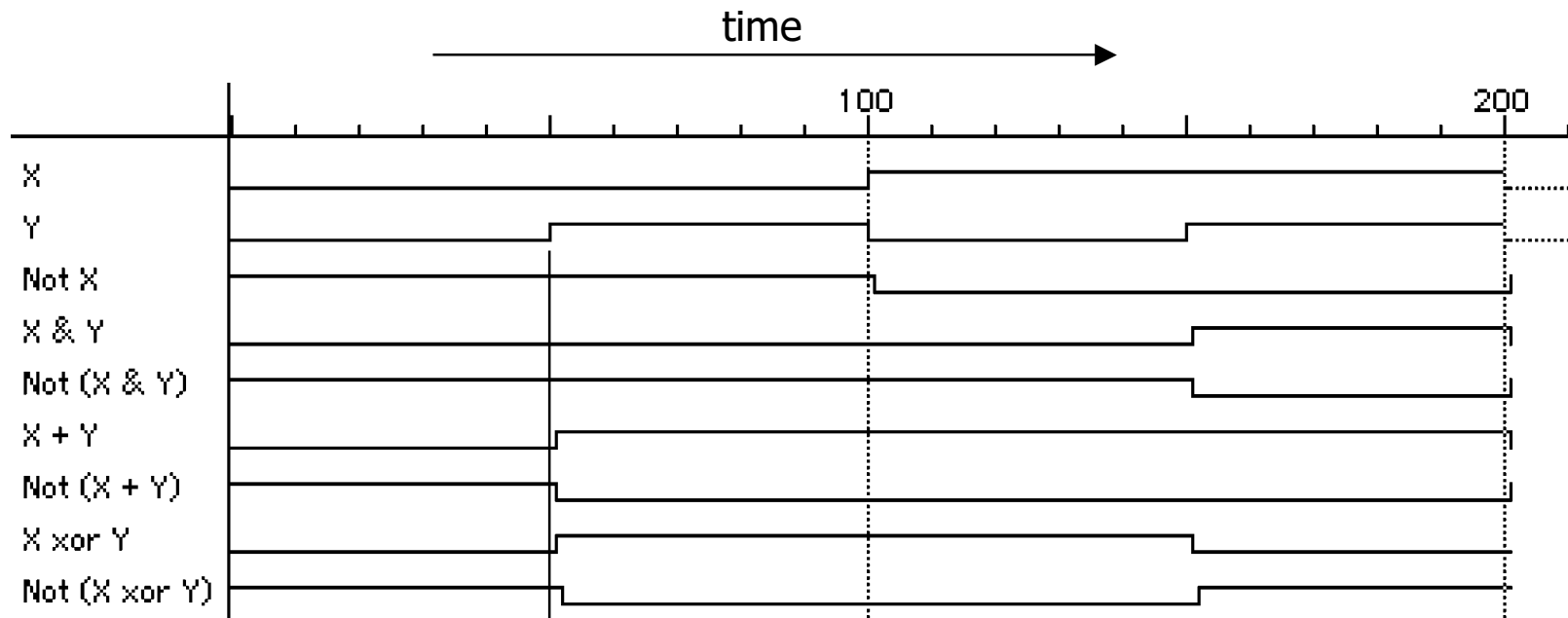
➤ e.g.,  $Z = A' \cdot B' \cdot (C + D) = (A' \cdot (B' \cdot (C + D)))$

$$\frac{\overline{T2}}{T1}$$



# Waveform view of logic functions

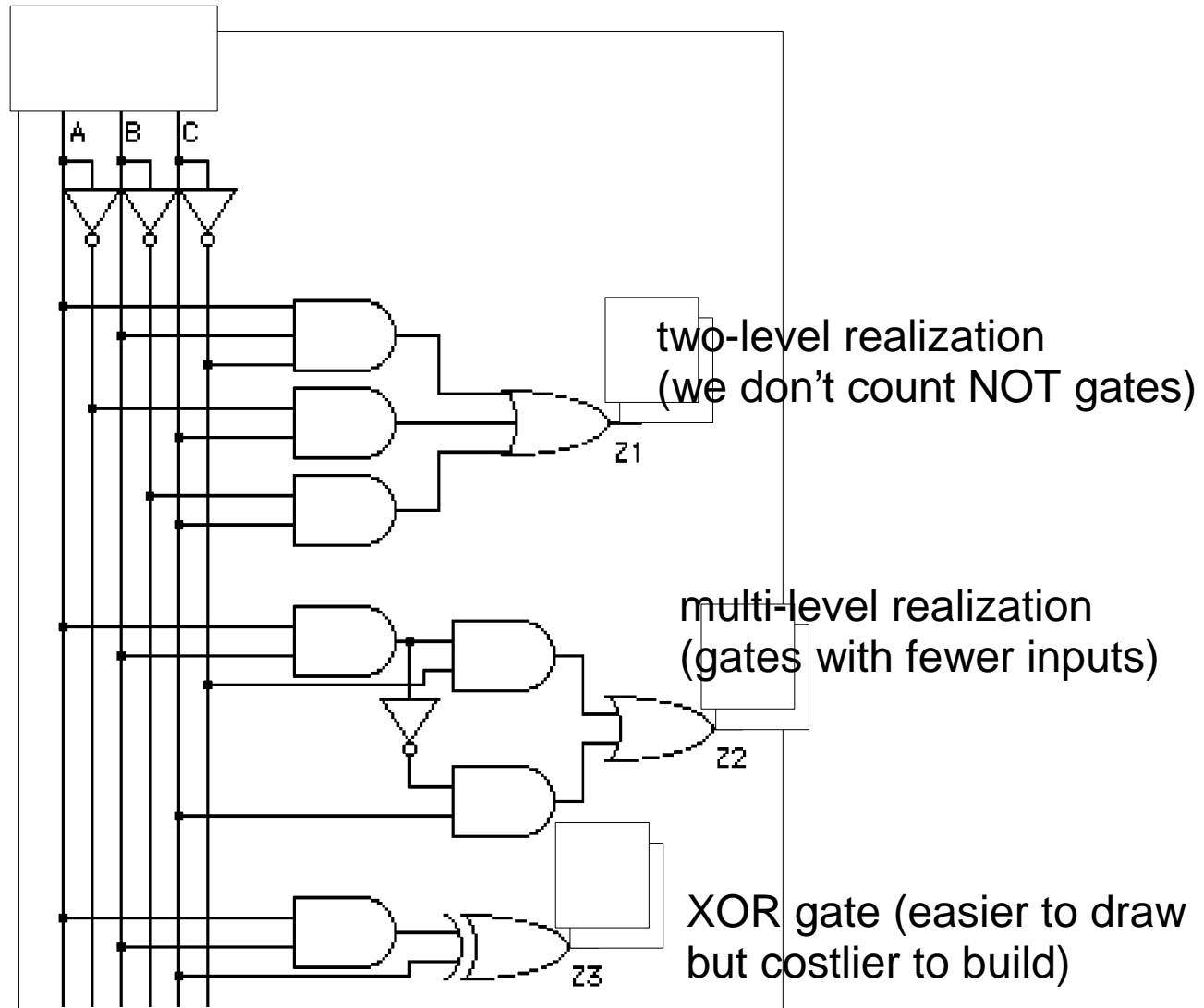
- Just a sideways truth table
  - but note how edges don't line up exactly
  - it takes time for a gate to switch its output!



change in Y takes time to "propagate" through gates

# Choosing different realizations of a function

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



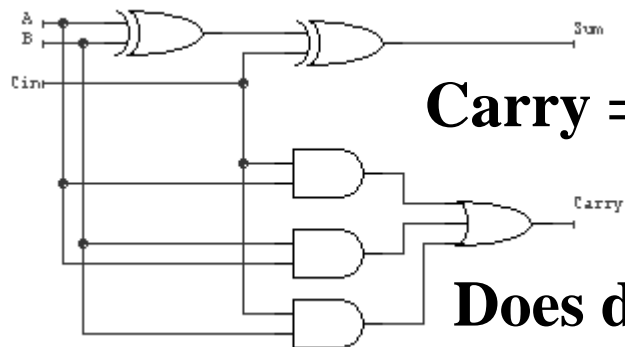
# Which realization is best?

## ❑ Reduce number of inputs

- literal: input variable (complemented or not)
  - can approximate cost of logic gate as 2 transistors per literal
  - why not count inverters?
- fewer literals means less transistors
  - smaller circuits
- fewer inputs implies faster gates
  - gates are smaller and thus also faster
- fan-ins (# of gate inputs) are limited in some technologies

## ❑ Reduce number of gates

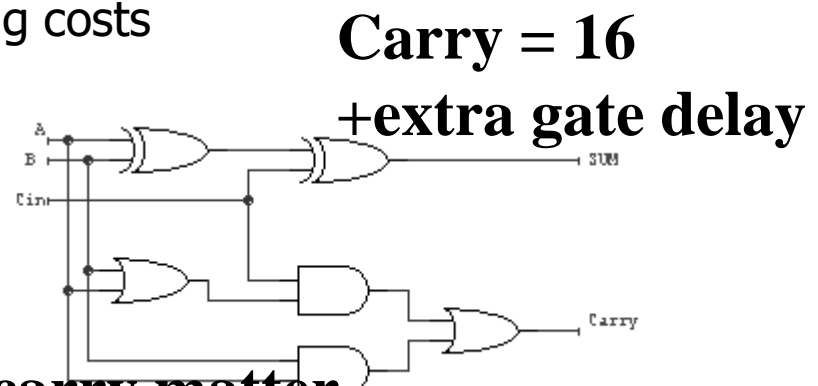
- fewer gates (and the packages they come in) means smaller circuits
  - directly influences manufacturing costs



**Carry = 18**

**Does delay of carry matter**

**since Sum is slow??**



**Carry = 16**

**+extra gate delay**

## Which is the best realization? (cont'd)

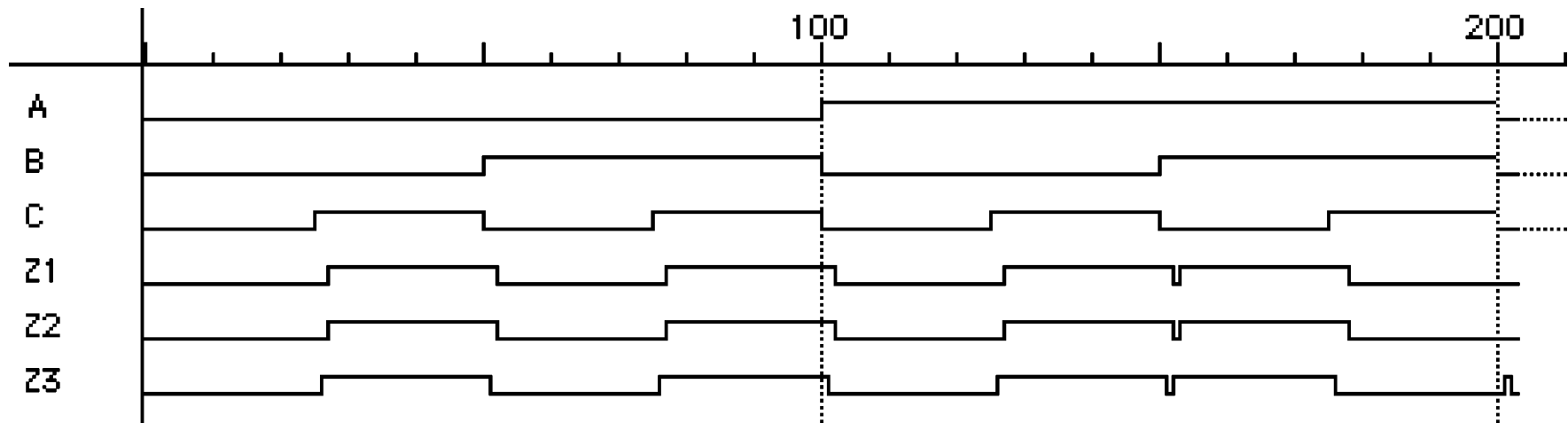
---

- ❑ Reduce number of levels of gates
  - fewer level of gates implies reduced signal propagation delays
  - minimum delay configuration typically requires more gates
    - wider, less deep circuits
- ❑ How do we explore tradeoffs between increased circuit delay and size?
  - automated tools to generate different solutions
  - logic minimization: reduce number of gates and complexity
  - logic optimization: reduction while trading off against delay



# Are all realizations equivalent?

- ❑ Under the same input stimuli, the three alternative implementations have almost the same waveform behavior
  - delays are different
  - glitches (hazards) may arise
  - variations due to differences in number of gate levels and structure
- ❑ The three implementations are functionally equivalent



# Implementing Boolean functions

---

- ❑ Technology independent
  - canonical forms
  - two-level forms
  - multi-level forms
  
- ❑ Technology choices
  - packages of a few gates
  - regular logic
  - two-level programmable logic
  - multi-level programmable logic
  - ASIC Cell Libraries

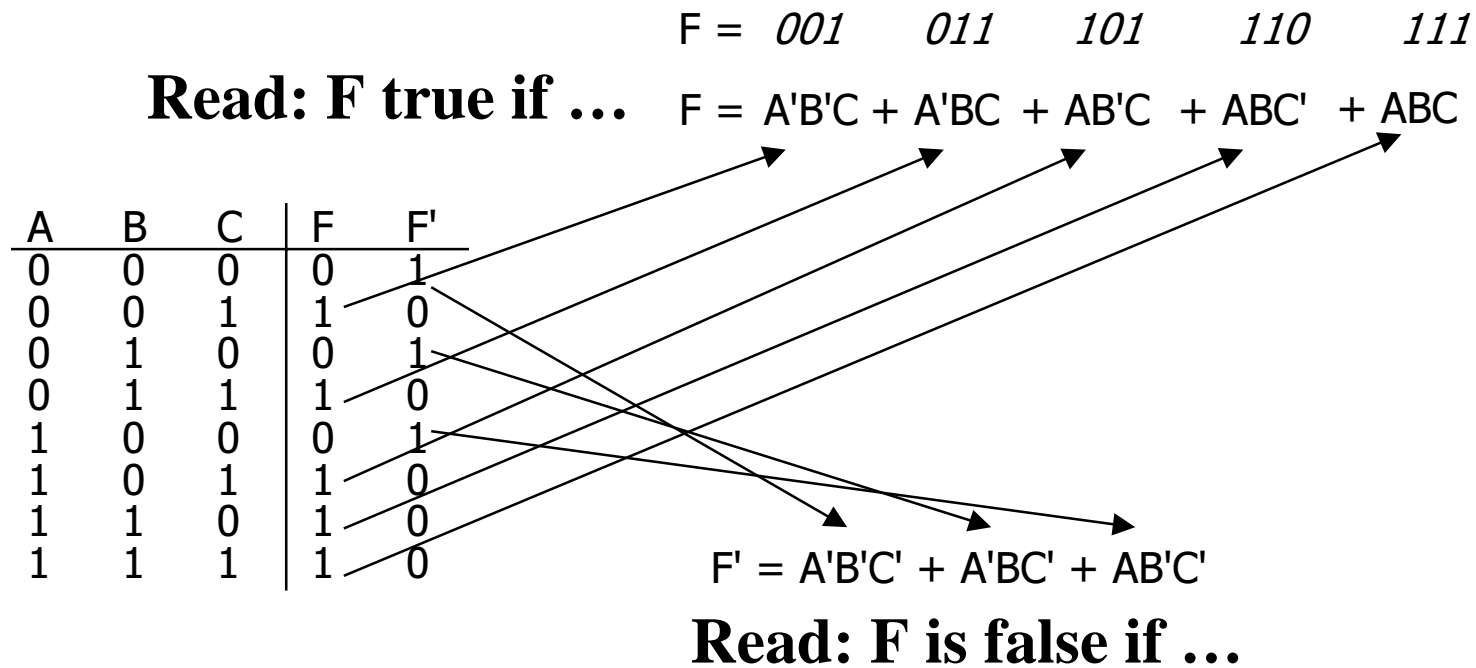
# Canonical forms

---

- ❑ Truth table is the unique signature of a Boolean function
- ❑ Many alternative gate realizations may have the same truth table
- ❑ Canonical forms
  - standard forms for a Boolean expression
  - provides a unique algebraic signature for a truth table

# Sum-of-products canonical forms

- ❑ Also known as disjunctive normal form
- ❑ Also known as minterm expansion




# Sum-of-products canonical form (cont'd)

□ Product term (or minterm)

- ANDed product of literals – input combination for which output is true
- each variable appears exactly once, in true or inverted form (but not both)

A	B	C	minterms	
0	0	0	A'B'C'	m0
0	0	1	A'B'C	m1
0	1	0	A'BC'	m2
0	1	1	A'BC	m3
1	0	0	AB'C'	m4
1	0	1	AB'C	m5
1	1	0	ABC'	m6
1	1	1	ABC	m7

short-hand notation for minterms of 3 variables



F in canonical form:

$$\begin{aligned}
 F(A, B, C) &= \Sigma m(1,3,5,6,7) \\
 &= m1 + m3 + m5 + m6 + m7 \\
 &= A'B'C + A'BC + AB'C + ABC' + ABC
 \end{aligned}$$

canonical form  $\neq$  minimal form

$$\begin{aligned}
 F(A, B, C) &= A'B'C + A'BC + AB'C + ABC + ABC' \\
 &= (A'B' + A'B + AB' + AB)C + ABC' \\
 &= ((A' + A)(B' + B))C + ABC' \\
 &= C + ABC' \\
 &= ABC' + C \\
 &= AB + C
 \end{aligned}$$

# Product-of-sums canonical form

- Also known as conjunctive normal form
- Also known as maxterm expansion

**Read: F is not false if...**  $F = (A + B + C) (A + B' + C) (A' + B + C)$

A	B	C	F	F'
0	0	0	0	1
0	0	1	1	0
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	0

**Read: F is not true if...**

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

# Product-of-sums canonical form (cont'd)

## □ Sum term (or maxterm)

- ORed sum of literals – input combination for which output is false
- each variable appears exactly once, in true or inverted form (but not both)

A	B	C	maxterms
0	0	0	A+B+C    M0
0	0	1	A+B+C'    M1
0	1	0	A+B'+C    M2
0	1	1	A+B'+C'    M3
1	0	0	A'+B+C    M4
1	0	1	A'+B+C'    M5
1	1	0	A'+B'+C    M6
1	1	1	A'+B'+C'    M7

short-hand notation for maxterms of 3 variables

F in canonical form:

$$\begin{aligned}
 F(A, B, C) &= \Pi M(0,2,4) \\
 &= M0 \cdot M2 \cdot M4 \\
 &= (A + B + C) (A + B' + C) (A' + B + C)
 \end{aligned}$$

canonical form  $\neq$  minimal form

$$\begin{aligned}
 F(A, B, C) &= (A + B + C) (A + B' + C) (A' + B + C) \\
 &= (A + B + C) (A + B' + C) \\
 &\quad (A + B + C) (A' + B + C) \\
 &= (A + C) (B + C)
 \end{aligned}$$

# S-o-P, P-o-S, and de Morgan's theorem

## □ Sum-of-products

➤  $F' = A'B'C' + A'BC' + AB'C'$

## □ Apply de Morgan's

➤  $(F')' = (A'B'C' + A'BC' + AB'C')'$

➤  $F = (A + B + C) (A + B' + C) (A' + B + C)$

## □ Product-of-sums

➤  $F' = (A + B + C') (A + B' + C') (A' + B + C') (A' + B' + C) (A' + B' + C')$

## □ Apply de Morgan's

➤  $(F')' = ( (A + B + C')(A + B' + C')(A' + B + C')(A' + B' + C)(A' + B' + C') )'$

➤  $F = A'B'C + A'BC + AB'C + ABC' + ABC$

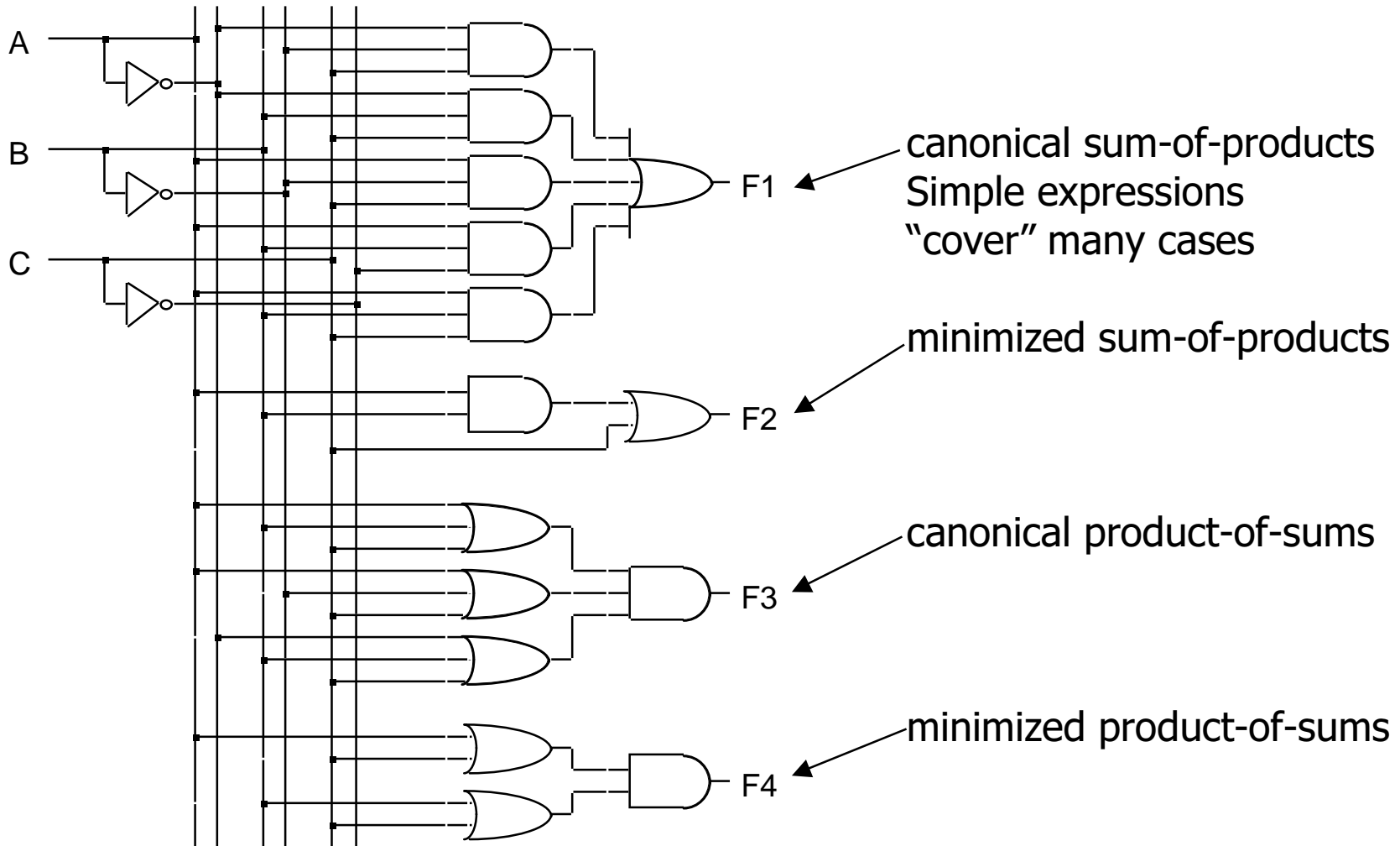
**Why go through all this?**

**Sometimes the intuitively easy solution gives you F'!!**



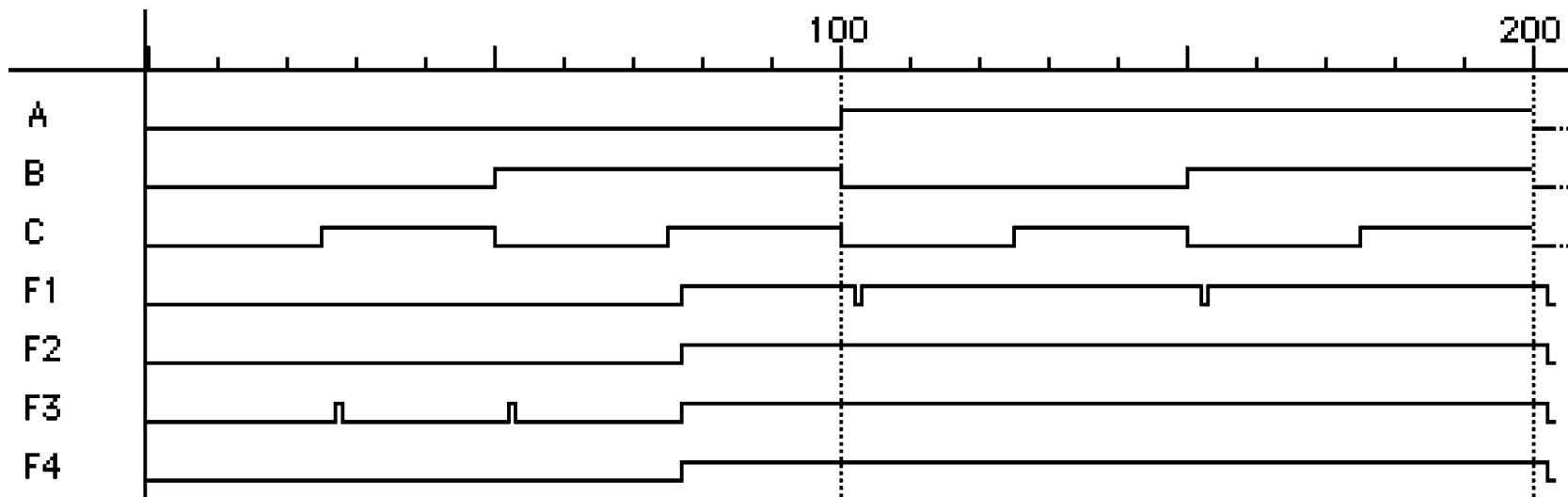
# Comparison of forms

$$F = AB + C$$



# Waveforms for the four alternatives

- Waveforms are essentially identical
  - except for timing hazards (glitches)
  - delays almost identical (modeled as a delay per level, not type of gate or number of inputs to gate)



# Mapping between canonical forms

## □ Minterm to maxterm conversion

- use maxterms whose indices do not appear in minterm expansion
- e.g.,  $F(A,B,C) = \Sigma m(1,3,5,6,7) = \Pi M(0,2,4)$

## □ Maxterm to minterm conversion

- use minterms whose indices do not appear in maxterm expansion
- e.g.,  $F(A,B,C) = \Pi M(0,2,4) = \Sigma m(1,3,5,6,7)$

## □ Minterm expansion of F to minterm expansion of F'

- use minterms whose indices do not appear
- e.g.,  $F(A,B,C) = \Sigma m(1,3,5,6,7)$  so  $F'(A,B,C) = \Sigma m(0,2,4)$

## □ Maxterm expansion of F to maxterm expansion of F'

- use maxterms whose indices do not appear
- e.g.,  $F(A,B,C) = \Pi M(0,2,4)$  so  $F'(A,B,C) = \Pi M(1,3,5,6,7)$

# Incompletely specified functions

- Example: binary coded decimal increment by 1
  - BCD digits encode the decimal digits 0 – 9 in the bit patterns 0000 – 1001

A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	0
0	0	1	0	0	0	1	1
0	0	1	1	0	1	0	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	1	1
0	1	1	1	1	0	0	0
1	0	0	0	1	0	0	1
1	0	0	1	0	0	0	0
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

off-set of W

on-set of W

don't care (DC) set of W

these inputs patterns should never be encountered in practice – **"don't care"** about associated output values, can be exploited in minimization

# Notation for incompletely specified functions

## □ Don't cares and canonical forms

- so far, only represented on-set
- also represent don't-care-set
- need two of the three sets (on-set, off-set, dc-set)

## □ Canonical representations of the BCD increment by 1 function:

- $Z = m_0 + m_2 + m_4 + m_6 + m_8 + d_{10} + d_{11} + d_{12} + d_{13} + d_{14} + d_{15}$
- $Z = \Sigma [ m(0,2,4,6,8) + d(10,11,12,13,14,15) ]$
- $Z = M_1 \cdot M_3 \cdot M_5 \cdot M_7 \cdot M_9 \cdot D_{10} \cdot D_{11} \cdot D_{12} \cdot D_{13} \cdot D_{14} \cdot D_{15}$
- $Z = \Pi [ M(1,3,5,7,9) \cdot D(10,11,12,13,14,15) ]$

# Simplification of two-level combinational logic

- ❑ Finding a minimal sum of products or product of sums realization
  - exploit don't care information in the process
- ❑ Algebraic simplification
  - not an algorithmic/systematic procedure
  - how do you know when the minimum realization has been found?
- ❑ Computer-aided design tools
  - precise solutions require very long computation times, especially for functions with many inputs ( $> 10$ )
  - heuristic methods employed – "educated guesses" to reduce amount of computation and yield good if not best solutions
- ❑ Hand methods still relevant
  - to understand automatic tools and their strengths and weaknesses
  - ability to check results (on small examples)
- ❑ Next: Non-algebraic methods for simplifying 2-level logic