Design of Digital Systems II

Combinational Logic Design Principles

Moslem Amiri, Václav Přenosil

Embedded Systems Laboratory
Faculty of Informatics, Masaryk University
Brno, Czech Republic

amiri@mail.muni.cz
prenosil@fi.muni.cz

October, 2012

- **Combinational logic circuit**
  - A circuit whose outputs depend only on its current inputs
  - May contain an arbitrary number of logic gates and inverters but no feedback loops
    - A **feedback loop** is a signal path that allows output of a gate to propagate back to input of that same gate
    - Such a loop creates sequential circuit behavior
- Combinational circuit **analysis**
  - We start with a logic diagram and proceed to a formal description of function performed by that circuit, such as a truth table or a logic expression
- Combinational circuit **synthesis**
  - Reverse of analysis
  - Starting with a formal description and proceeding to a logic diagram

- **Boolean algebra**
  - A two-valued algebraic system
  - Is used to formulate propositions that are true or false, combine them to make new propositions, and determine truth or falsehood of the new propositions
- **Switching algebra**
  - Adaptation of Boolean algebra to analyze and describe behavior of circuits
  - A physical condition—voltage HIGH or LOW, capacitor charged or discharged, and so on—is represented by a variable X that can have one of two possible values, 0 or 1

Switching Algebra: Axioms

- **Axioms** or **postulates**
  - A minimal set of basic definitions that we assume to be true, from which all other information about system can be derived
  - The first two axioms of switching algebra embody digital abstraction

    $(A1) \quad X = 0 \quad \text{if } X \neq 1 \qquad (A1') \quad X = 1 \quad \text{if } X \neq 0$

    - The only difference between $A1$ and $A1'$ is interchange of 0 and 1
    - This is a characteristic of all axioms of switching algebra
    - This is basis of duality principle
  - If X denotes an inverter's input signal, X' denotes its output value

    $(A2) \quad \text{If } X = 0, \text{ then } X' = 1 \qquad (A2') \quad \text{If } X = 1, \text{ then } X' = 0$

  - AND and OR operations (AND has precedence)

    $(A3) \quad 0 \cdot 0 = 0 \qquad (A3') \quad 1 + 1 = 1$

    $(A4) \quad 1 \cdot 1 = 1 \qquad (A4') \quad 0 + 0 = 0$

    $(A5) \quad 0 \cdot 1 = 1 \cdot 0 = 0 \qquad (A5') \quad 1 + 0 = 0 + 1 = 1$

  - The five pairs of axioms, $A1$–$A5$ and $A1'$–$A5'$, completely define switching algebra
    - All other facts about system can be proved using these axioms as a staring point

Switching Algebra: Single-Variable Theorems

- Switching-algebra **theorems**
  - True statements that allow us to manipulate algebraic expressions to allow simpler analysis or more efficient synthesis of corresponding circuits

Table 1: Switching-algebra theorems with one variable.

(T1)	$X + 0 = X$	(T1′)	$X \cdot 1 = X$	(Identities)
(T2)	$X + 1 = 1$	(T2′)	$X \cdot 0 = 0$	(Null elements)
(T3)	$X + X = X$	(T3′)	$X \cdot X = X$	(Idempotency)
(T4)	$(X')' = X$			(Involution)
(T5)	$X + X' = 1$	(T5′)	$X \cdot X' = 0$	(Complements)

- **Perfect induction**
  - A technique to prove theorems in switching algebra
  - Since a variable can take on only 0 and 1, prove a theorem involving a single variable X by proving that it is true for both $X = 0$ and $X = 1$
  - E.g., to prove T1

$$[X = 0] \longrightarrow 0 + 0 = 0 \longrightarrow \text{true, according to axiom } A4'$$
$$[X = 1] \longrightarrow 1 + 0 = 1 \longrightarrow \text{true, according to axiom } A5'$$

Table 2: Switching-algebra theorems with two or three variables.

(T6)	$X + Y = Y + X$	(T6′)	$X \cdot Y = Y \cdot X$	(Commutativity)
(T7)	$(X + Y) + Z = X + (Y + Z)$	(T7′)	$(X \cdot Y) \cdot Z = X \cdot (Y \cdot Z)$	(Associativity)
(T8)	$X \cdot Y + X \cdot Z = X \cdot (Y + Z)$	(T8′)	$(X + Y) \cdot (X + Z) = X + Y \cdot Z$	(Distributivity)
(T9)	$X + X \cdot Y = X$	(T9′)	$X \cdot (X + Y) = X$	(Covering)
(T10)	$X \cdot Y + X \cdot Y' = X$	(T10′)	$(X + Y) \cdot (X + Y') = X$	(Combining)
(T11)	$X \cdot Y + X' \cdot Z + Y \cdot Z = X \cdot Y + X' \cdot Z$			(Consensus)
(T11′)	$(X + Y) \cdot (X' + Z) \cdot (Y + Z) = (X + Y) \cdot (X' + Z)$			

- Theorems in Tab. 2 are proved by perfect induction, by evaluating theorem statements for all possible combinations of X and Y (and Z)
- Proof of T9: $X + X \cdot Y = X \cdot 1 + X \cdot Y = X \cdot (1 + Y) = X \cdot 1 = X$
- In T11, if $Y \cdot Z = 1$, either $X \cdot Y$ or $X' \cdot Z$ must also be 1
  - Thus, $Y \cdot Z$ term is redundant
  - Consensus theorem is used to eliminate certain timing hazards
- It is possible to replace each variable in Tab. 2 with a logic expression

Switching Algebra: n-Variable Theorems

Table 3: Switching-algebra theorems with n variables.

(T12)	$X + X + \cdots + X = X$	(Generalized idempotency)
(T12′)	$X \cdot X \cdot \cdots \cdot X = X$	
(T13)	$(X_1 \cdot X_2 \cdot \cdots \cdot X_n)' = X_1' + X_2' + \cdots + X_n'$	(DeMorgan's theorems)
(T13′)	$(X_1 + X_2 + \cdots + X_n)' = X_1' \cdot X_2' \cdot \cdots \cdot X_n'$	
(T14)	$[F(X_1,X_2,\ldots,X_n,+,\cdot)]' = F(X_1',X_2',\ldots,X_n',\cdot,+)$	(Generalized DeMorgan's theorem)
(T15)	$F(X_1,X_2,\ldots,X_n) = X_1 \cdot F(1,X_2,\ldots,X_n) + X_1' \cdot F(0,X_2,\ldots,X_n)$	(Shannon's expansion theorems)
(T15′)	$F(X_1,X_2,\ldots,X_n) = [X_1 + F(0,X_2,\ldots,X_n)] \cdot [X_1' + F(1,X_2,\ldots,X_n)]$	

- Theorems in Tab. 3 are proved using **finite induction**
  1. *Basis step*: prove theorem is true for $n = 2$
  2. *Induction step*: if theorem is true for $n = i$, it is also true for $n = i + 1$
- Example: T12
  1. For $n = 2$, T12 = T3, therefore true
  2. If it is true for a logical sum of i X's, it is also true for a sum of $i+1$ X's

$$\underbrace{X + X + X + \cdots + X}_{i+1\ X\text{'s}} = X + (\underbrace{X + X + \cdots + X}_{i\ X\text{'s}}) = X + (\underbrace{X}_{\text{if T12 is true for } n=i}) \overset{\text{T3}}{=} X$$

Switching Algebra: *n*-Variable Theorems

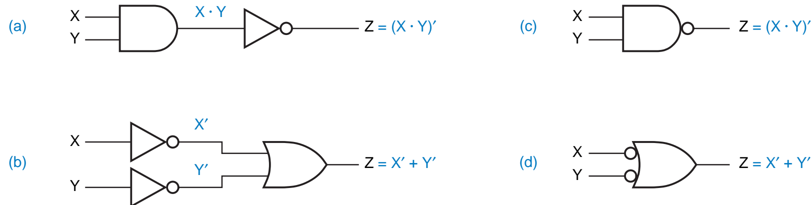


Figure 1: Equivalent circuits according to DeMorgan's theorem T13: (a) AND-NOT; (b) NOT-OR; (c) logic symbol for a NAND gate; (d) equivalent symbol for a NAND gate.

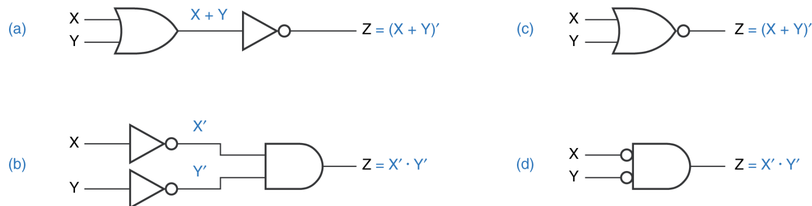


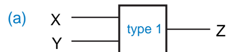
Figure 2: Equivalent circuits according to DeMorgan's theorem T13′: (a) OR-NOT; (b) NOT-AND; (c) logic symbol for a NOR gate; (d) equivalent symbol for a NOR gate.

- **Principle of duality**
  - Any theorem or identity in switching algebra remains true if 0 and 1 are swapped and $\cdot$ and $+$ are swapped throughout
  - This is true because duals of all axioms are true, so duals of all switching-algebra theorems can be proved using duals of axioms
- **Dual of a logic expression**

$$F^D(X_1, X_2, \ldots, X_n, +, \cdot, ') = F(X_1, X_2, \ldots, X_n, \cdot, +, ')$$

- Generalized DeMorgan's theorem

$$\begin{aligned}[F(X_1, X_2, \ldots, X_n, +, \cdot)]' &= F(X_1', X_2', \ldots, X_n', \cdot, +) \\ &= F^D(X_1', X_2', \ldots, X_n', +, \cdot)\end{aligned}$$

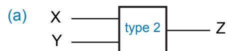


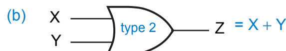
X	Y	Z
LOW	LOW	LOW
LOW	HIGH	LOW
HIGH	LOW	LOW
HIGH	HIGH	HIGH

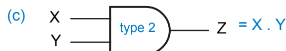
X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

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

Figure 3: A "type-1" logic gate: (a) electrical function table; (b) logic function table and symbol with positive logic; (c) logic function table and symbol with negative logic.







X	Y	Z
LOW	LOW	LOW
LOW	HIGH	HIGH
HIGH	LOW	HIGH
HIGH	HIGH	HIGH

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

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

Figure 4: A "type-2" logic gate: (a) electrical function table; (b) logic function table and symbol with positive logic; (c) logic function table and symbol with negative logic.

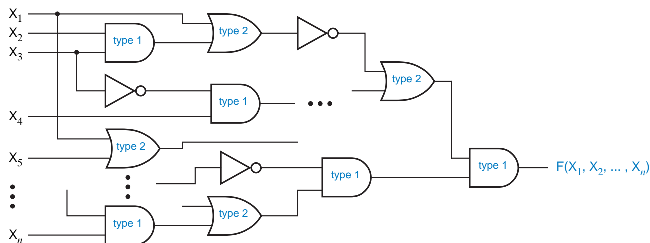


Figure 5: Circuit for a logic function using inverters and type-1 and type-2 gates under a positive-logic convention.

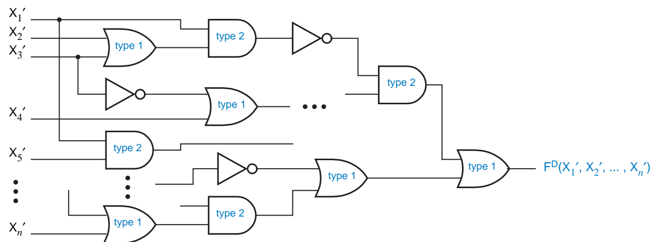


Figure 6: Negative-logic interpretation of the previous circuit.

Switching Algebra: Duality

- Figs. 5 and 6
  - Fig. 5 shows a circuit corresponding to expression $F(X_1, X_2, \ldots, X_n)$ following positive-logic convention
  - Circuit of Fig. 6 is that of Fig. 5 without change, but logic convention is changed from positive to negative
    - For every possible combination of input voltages (HIGH and LOW), the circuit still produces the same output voltage
    - But from point of view of switching algebra, output value—0 or 1—is opposite of what it was under positive-logic convention
    - Likewise, each input value is opposite of what it was
  - Therefore, for each possible input combination to circuit in Fig. 5, output is opposite of that produced by opposite input combination applied to circuit in Fig. 6

$$F(X_1, X_2, \ldots, X_n) = [F^D(X_1', X_2', \ldots, X_n')]'$$

By complementing both sides, we get generalized DeMorgan's theorem

$$[F(X_1, X_2, \ldots, X_n)]' = F^D(X_1', X_2', \ldots, X_n')$$

Standard Representations of Logic Functions

- The most basic representation of a logic function is **truth table**
  - It lists output of circuit for every possible input combination

Table 4: General truth table structure for a 3-variable logic function, $F(X, Y, Z)$.

Row	X	Y	Z	F
0	0	0	0	F(0,0,0)
1	0	0	1	F(0,0,1)
2	0	1	0	F(0,1,0)
3	0	1	1	F(0,1,1)
4	1	0	0	F(1,0,0)
5	1	0	1	F(1,0,1)
6	1	1	0	F(1,1,0)
7	1	1	1	F(1,1,1)

Table 5: Truth table for a particular 3-variable logic function, $F(X, Y, Z)$.

Row	X	Y	Z	F
0	0	0	0	1
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	1
5	1	0	1	0
6	1	1	0	1
7	1	1	1	1

Standard Representations of Logic Functions

- **Literal**
  - A variable or complement of a variable
  - Examples: $X,\ Y,\ X',\ Y'$
- **Product term**
  - A single literal or a logical product of two or more literals
  - Examples: $Z',\ W \cdot X \cdot Y,\ X \cdot Y' \cdot Z,\ W' \cdot Y' \cdot Z$
- **Sum-of-products expression**
  - A logical sum of product terms
  - Example: $Z' + W \cdot X \cdot Y + X \cdot Y' \cdot Z + W' \cdot Y' \cdot Z$
- **Sum term**
  - A single literal or a logical sum of two or more literals
  - Examples: $Z',\ W + X + Y,\ X + Y' + Z,\ W' + Y' + Z$
- **Product-of-sums expression**
  - A logical product of sum terms
  - Example: $Z' \cdot (W + X + Y) \cdot (X + Y' + Z) \cdot (W' + Y' + Z)$

- **Normal term**
  - A product or sum term in which no variable appears more than once
  - A nonnormal term can always be simplified to a constant or a normal term
  - Examples of nonnormal terms: $W \cdot X \cdot X \cdot Y',\ W + W + X' + Y,\ X \cdot X' \cdot Y$
  - Examples of normal terms: $W \cdot X \cdot Y',\ W + X' + Y$
- *n*-variable **minterm**
  - A normal product term with n literals
  - There are 2^n such product terms
  - Examples of 4-variable minterms: $W' \cdot X' \cdot Y' \cdot Z',\ W \cdot X \cdot Y' \cdot Z,\ W' \cdot X' \cdot Y \cdot Z'$
- *n*-variable **maxterm**
  - A normal sum term with n literals
  - There are 2^n such sum terms
  - Examples of 4-variable maxterms: $W' + X' + Y' + Z',\ W + X' + Y' + Z,\ W' + X' + Y + Z'$

- Correspondence between truth table and minterms and maxterms
  - A minterm is defined as a product term that is 1 in exactly one row of truth table
  - A maxterm is defined as a sum term that is 0 in exactly one row of truth table

Table 6: Minterms and maxterms for a 3-variable logic function, $F(X, Y, Z)$.

Row	X	Y	Z	F	***Minterm***	***Maxterm***
0	0	0	0	F(0,0,0)	$X' \cdot Y' \cdot Z'$	$X + Y + Z$
1	0	0	1	F(0,0,1)	$X' \cdot Y' \cdot Z$	$X + Y + Z'$
2	0	1	0	F(0,1,0)	$X' \cdot Y \cdot Z'$	$X + Y' + Z$
3	0	1	1	F(0,1,1)	$X' \cdot Y \cdot Z$	$X + Y' + Z'$
4	1	0	0	F(1,0,0)	$X \cdot Y' \cdot Z'$	$X' + Y + Z$
5	1	0	1	F(1,0,1)	$X \cdot Y' \cdot Z$	$X' + Y + Z'$
6	1	1	0	F(1,1,0)	$X \cdot Y \cdot Z'$	$X' + Y' + Z$
7	1	1	1	F(1,1,1)	$X \cdot Y \cdot Z$	$X' + Y' + Z'$

Standard Representations of Logic Functions

- **Minterm i**
  - The minterm corresponding to row i of truth table
  - A variable is complemented if corresponding bit in binary is 0
  - Example: row 5 $\longrightarrow$ binary: 101 $\longrightarrow$ minterm 5: $X \cdot Y' \cdot Z$
- **Maxterm i**
  - The maxterm corresponding to row i of truth table
  - A variable is complemented if corresponding bit in binary is 1
  - Example: row 5 $\longrightarrow$ binary: 101 $\longrightarrow$ maxterm 5: $X' + Y + Z'$
- **Canonical sum** of a logic function
  - A sum of minterms corresponding to truth-table rows (input combinations) for which the function produces a 1 output
- **Canonical product** of a logic function
  - A product of maxterms corresponding to input combinations for which the function produces a 0 output

Standard Representations of Logic Functions

- In Tab. 5
  - Canonical sum

$$F = \sum_{X,Y,Z}(0,3,4,6,7)$$
$$= X' \cdot Y' \cdot Z' + X' \cdot Y \cdot Z + X \cdot Y' \cdot Z' + X \cdot Y \cdot Z' + X \cdot Y \cdot Z$$

Notation $\sum_{X,Y,Z}(0,3,4,6,7)$ is a **minterm list** or **on-set**

  - Canonical product

$$F = \prod_{X,Y,Z}(1,2,5) = (X + Y + Z') \cdot (X + Y' + Z) \cdot (X' + Y + Z')$$

Notation $\prod_{X,Y,Z}(1,2,5)$ is a **maxterm list** or **off-set**

Row	X	Y	Z	F
0	0	0	0	1
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	1
5	1	0	1	0
6	1	1	0	1
7	1	1	1	1

Standard Representations of Logic Functions

- Conversion between a minterm list and a maxterm list
  - For a function of n variables, possible minterm and maxterm numbers are in the set $\{0, 1, \ldots, 2^n - 1\}$
  - To switch between list types, take the set complement
  - Example

$$\sum_{A,B,C}(0,1,2,3) = \prod_{A,B,C}(4,5,6,7)$$

$$\sum_{X,Y}(1) = \prod_{X,Y}(0,2,3)$$

$$\sum_{W,X,Y,Z}(0,1,2,3,5,7,11,13) = \prod_{W,X,Y,Z}(4,6,8,9,10,12,14,15)$$

- Each of these representations specifies exactly the same information
  1. A truth table
  2. An algebraic sum of minterms, the canonical sum
  3. A minterm list using $\sum$ notation
  4. An algebraic product of maxterms, the canonical product
  5. A maxterm list using $\prod$ notation

Combinational-Circuit Analysis

- We analyze a combinational logic circuit by obtaining a formal description of its logic function
- Operations possible after obtaining a formal description
  - Determining behavior of logic circuit for various input combinations
  - Manipulating an algebraic description to suggest different circuit structures
  - Transforming an algebraic description into a standard form corresponding to an available circuit structure
    - E.g., a sum-of-products expression corresponds directly to circuit structure used in PLAs, and a truth table corresponds to lookup memory used in most FPGAs
  - Using an algebraic description of circuit's functional behavior in analysis of a larger system that includes the circuit

Combinational-Circuit Analysis

- The most basic functional description is truth table
  - Obtain truth table of an n-input circuit by working the way through all 2^n input combinations
  - For each input combination, determine all of gate outputs produced by that input, propagating information from circuit inputs to outputs
  - Truth table is written by transcribing output sequence of final gate

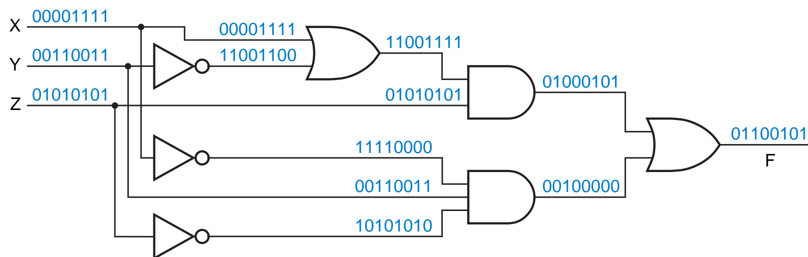


Figure 7: Gate outputs created by all input combinations.

Table 7: Truth table for the logic circuit of Fig. 7.

Row	X	Y	Z	F
0	0	0	0	0
1	0	0	1	1
2	0	1	0	1
3	0	1	1	0
4	1	0	0	0
5	1	0	1	1
6	1	1	0	0
7	1	1	1	1

- The number of input combinations of a logic circuit grows exponentially with the number of inputs
  - Instead of exhaustive approach, we normally use an algebraic approach
  - Complexity of algebraic approach is linearly proportional to size of circuit

- Algebraic approach
  - Build up a parenthesized logic expression corresponding to logic operators and structure of circuit
  - Start at circuit inputs and propagate expressions through gates toward output
  - Simplify expressions while going, or defer all algebraic manipulations until an output expression is obtained

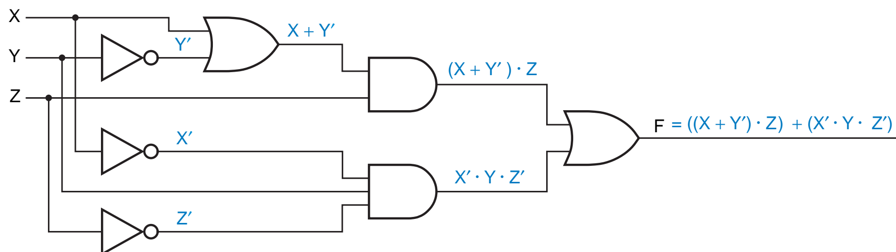


Figure 8: Logic expressions for signal lines.

- In Fig. 8, a sum of products is obtained by "multiplying out" output function

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

This new expression corresponds to a different circuit for the same logic function, as shown in Fig. 9

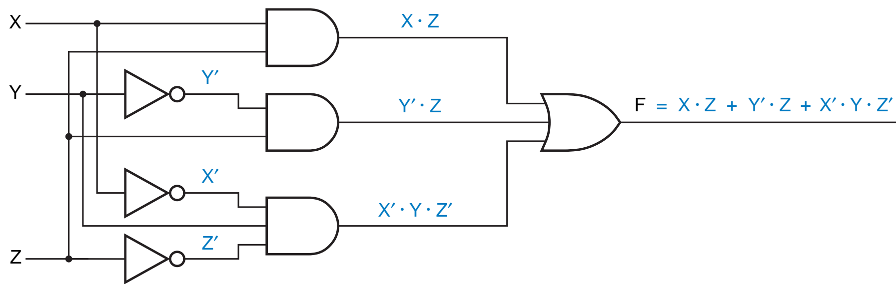


Figure 9: Two-level AND-OR circuit.

- Similarly, a product of sums is obtained by "adding out" output function of Fig. 8

$$
\begin{aligned}
F &= ((X + Y') \cdot Z) + (X' \cdot Y \cdot Z') \\
&= (X + Y' + X') \cdot (X + Y' + Y) \cdot (X + Y' + Z') \cdot (Z + X') \cdot (Z + Y) \cdot (Z + Z') \\
&= 1 \cdot 1 \cdot (X + Y' + Z') \cdot (X' + Z) \cdot (Y + Z) \cdot 1 \\
&= (X + Y' + Z') \cdot (X' + Z) \cdot (Y + Z)
\end{aligned}
$$

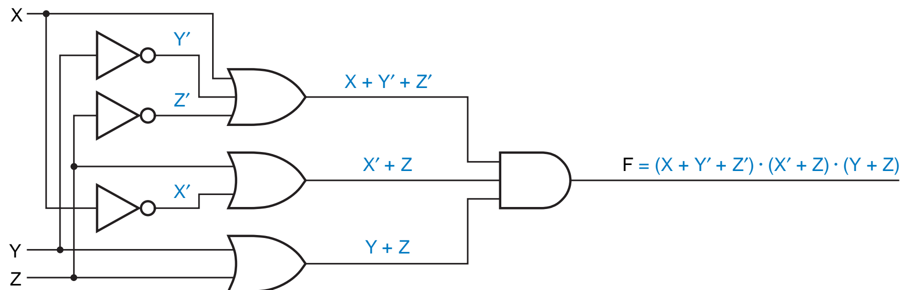


Figure 10: Two-level OR-AND circuit.

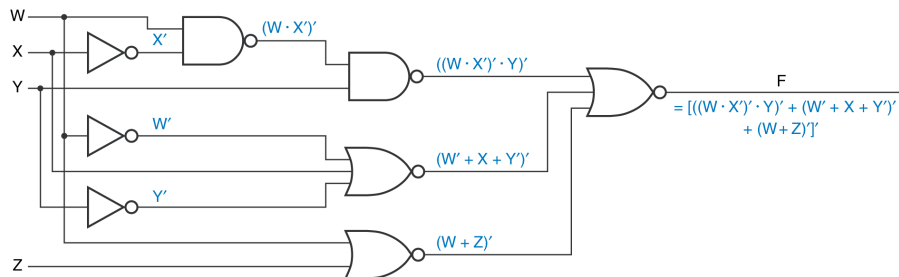


Figure 11: Algebraic analysis of a logic circuit with NAND and NOR gates.

$$
\begin{aligned}
F &= [((W \cdot X')' \cdot Y)' + (W' + X + Y')' + (W + Z)']' \\
&= ((W' + X)' + Y')' \cdot (W \cdot X' \cdot Y)' \cdot (W' \cdot Z')' \\
&= ((W \cdot X')' \cdot Y) \cdot (W' + X + Y') \cdot (W + Z) \\
&= ((W' + X) \cdot Y) \cdot (W' + X + Y') \cdot (W + Z)
\end{aligned}
$$

- DeMorgan's theorem can be applied *graphically* to simplify algebraic analysis
  - We can cancel out some of inversions
  - In Fig. 12, this manipulation leads us to a simplified output expression directly

$$F = ((W' + X) \cdot Y) \cdot (W' + X + Y') \cdot (W + Z) \quad (1)$$

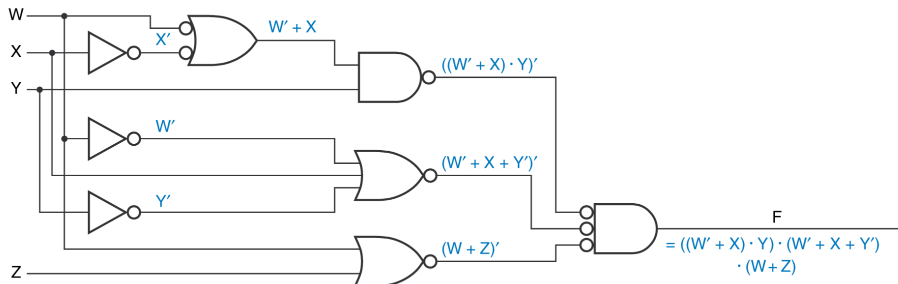


Figure 12: Algebraic analysis of the circuit in Fig. 11 after substituting some NAND and NOR symbols.

- When we simplify a logic expression, we get an expression corresponding to a different physical circuit
  - E.g., simplified expression (1) corresponds to circuit of Fig. 13

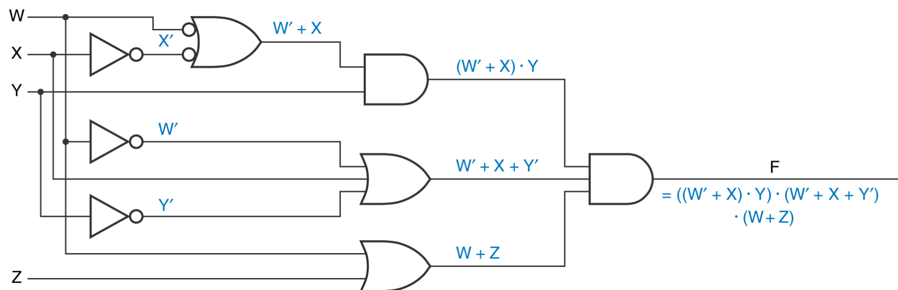


Figure 13: A different circuit for same logic function.

- We could also multiply out and add out expression (1) to obtain sum-of-products and product-of-sums expressions corresponding to two more physically different circuits for same logic function

- Logic expressions are not always used to convey information about physical structure of a circuit
  - An expression might describe more than one circuit structure
  - The only sure way to determine a circuit's structure is via its drawing
  - But, for certain classes of circuits, structural information could be described without reference to drawing
    - E.g., "a two-level NAND-NAND circuit for $W \cdot X \cdot Y + Y \cdot Z$"

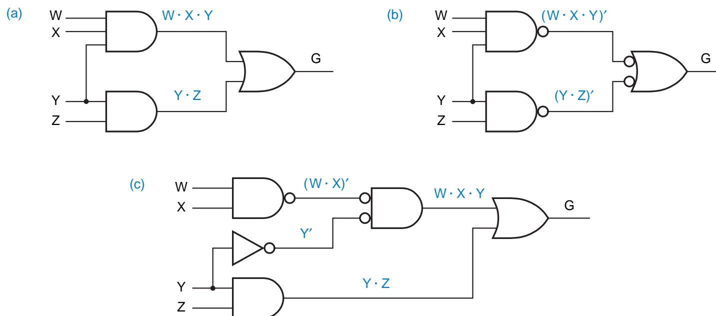


Figure 14: Three circuits for $G(W, X, Y, Z) = W \cdot X \cdot Y + Y \cdot Z$: (a) two-level AND-OR; (b) two-level NAND-NAND; (c) with 2-input gates only.

- Sometimes, a logic circuit description is a list of input combinations, verbal equivalent of a truth table or $\sum$ or $\prod$ notation
  - Example (prime-number detector): "Given a 4-bit input combination $N = N_3N_2N_1N_0$, produce a 1 output for $N = 1, 2, 3, 5, 7, 11, 13$ and 0 otherwise"
  - A logic function described in this way can be designed directly from canonical sum or product expression

$$
\begin{aligned}
F &= \sum_{N_3,N_2,N_1,N_0} (1, 2, 3, 5, 7, 11, 13) \\
&= N_3' \cdot N_2' \cdot N_1' \cdot N_0 + N_3' \cdot N_2' \cdot N_1 \cdot N_0' + N_3' \cdot N_2' \cdot N_1 \cdot N_0 \\
&\quad + N_3' \cdot N_2 \cdot N_1' \cdot N_0 + N_3' \cdot N_2 \cdot N_1 \cdot N_0 + N_3 \cdot N_2' \cdot N_1 \cdot N_0 \\
&\quad + N_3 \cdot N_2 \cdot N_1' \cdot N_0
\end{aligned}
$$

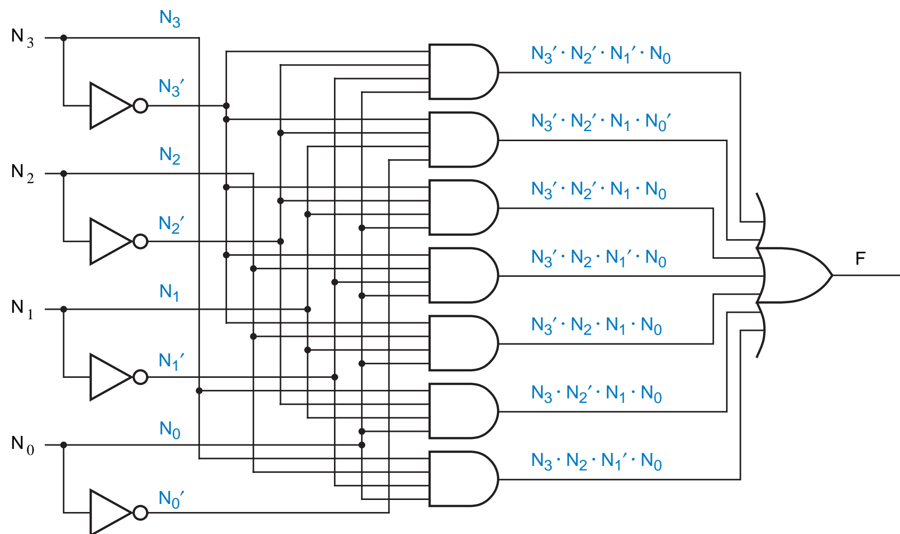


Figure 15: Canonical-sum design for 4-bit prime-number detector.

- Often, we describe a logic function using English-language connectives "and," "or," and "not"
  - Example (alarm circuit): "ALARM output is 1 if PANIC input is 1, or if ENABLE input is 1, EXITING input is 0, and house is not secure; house is secure if WINDOW, DOOR, and GARAGE inputs are all 1"
  - Such a description can be translated directly into algebraic expressions

$$\text{ALARM} = \text{PANIC} + \text{ENABLE} \cdot \text{EXITING}' \cdot \text{SECURE}'$$

$$\text{SECURE} = \text{WINDOW} \cdot \text{DOOR} \cdot \text{GARAGE}$$

$$\text{ALARM} = \text{PANIC} + \text{ENABLE} \cdot \text{EXITING}' \cdot (\text{WINDOW} \cdot \text{DOOR} \cdot \text{GARAGE})'$$

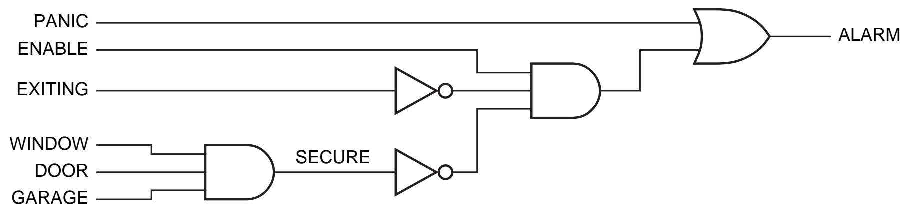


Figure 16: Alarm circuit derived from logic expression.

- Having an expression for a logic function, we can do some other operations
  - We can manipulate it to get different circuits
    - E.g., ALARM expression can be multiplied out to get sum-of-products circuit
  - We can construct the truth table for the expression and use any of synthesis methods that apply to truth tables
    - E.g., canonical sum or product method and minimization methods

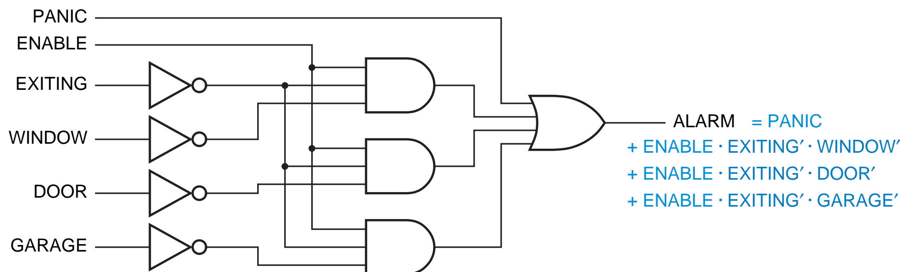


Figure 17: Sum-of-products version of alarm circuit.

Combinational-Circuit Synthesis: Circuit Manipulations

- We can translate any logic expression into an equivalent sum-of-products expression by multiplying it out
  - Such an expression may be realized directly with AND and OR gates
  - By substituting gates: two-level AND-OR $\longrightarrow$ two-level NAND-NAND

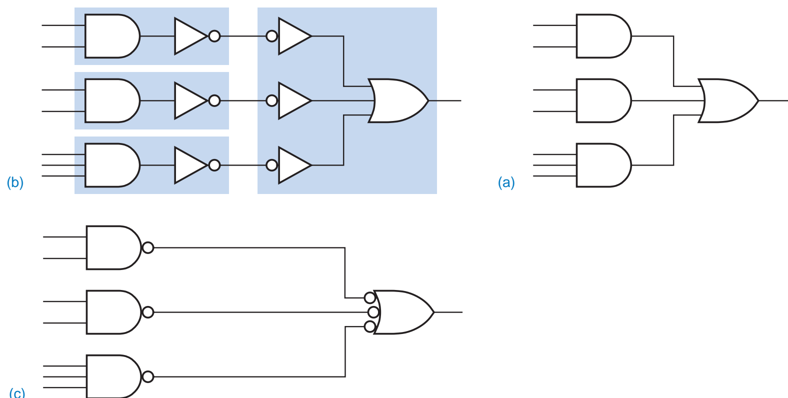


Figure 18: Alternative sum-of-products realizations: (a) AND-OR; (b) AND-OR with extra inverter pairs; (c) NAND-NAND.

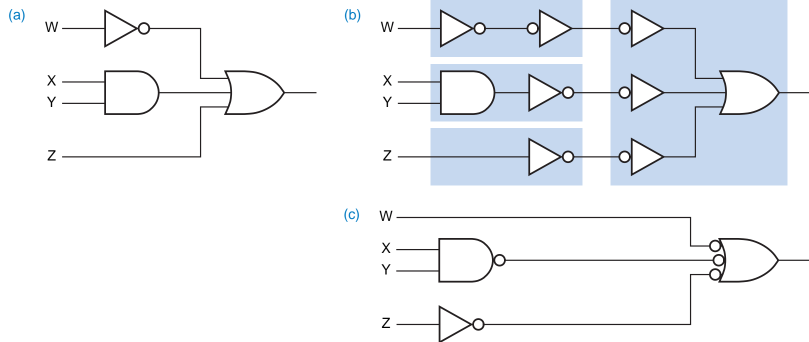


Figure 19: Another two-level sum-of-products circuit: (a) AND-OR; (b) AND-OR with extra inverter pairs; (c) NAND-NAND.

- We can translate any logic expression into an equivalent product-of-sums expression by adding it out
  - Such an expression has both OR-AND and NOR-NOR circuit realizations

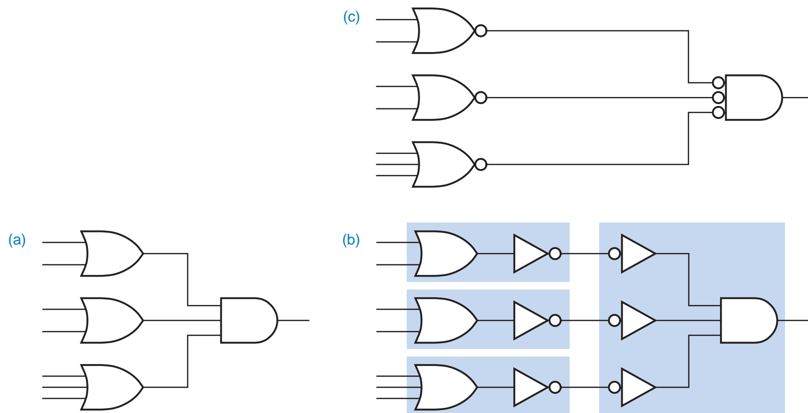


Figure 20: Realizations of a product-of-sums expression: (a) OR-AND; (b) OR-AND with extra inverter pairs; (c) NOR-NOR.

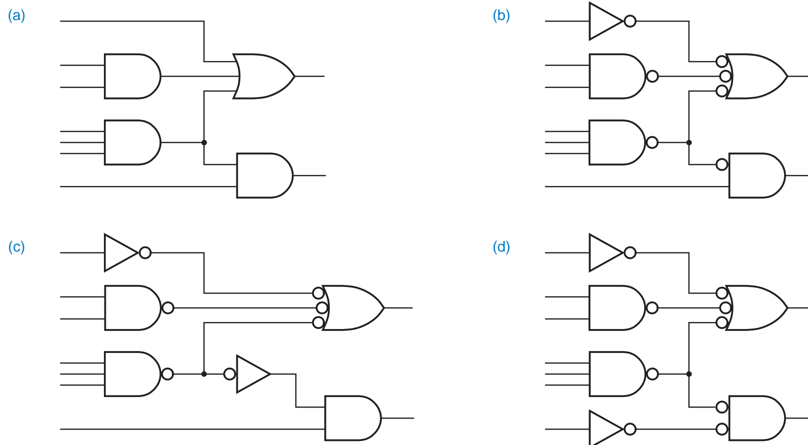


Figure 21: Logic-symbol manipulations: (a) original circuit; (b) transformation with a nonstandard gate; (c) inverter used to eliminate nonstandard gate; (d) preferred inverter placement; one level of gate delay is eliminated, and bottom gate becomes a NOR instead of AND.

Combinational-Circuit Synthesis: Minimization

- Conmbinational-circuit-minimization methods have as their starting point a truth table or, equivalently, a minterm list or maxterm list
  - Given a logic function that is not expressed in this form, we must convert it to an appropriate form before using these methods
- Minimization methods reduce cost of a two-level AND-OR, OR-AND, NAND-NAND, or NOR-NOR circuit in three ways
  1. By minimizing number of first-level gates
  2. By minimizing number of inputs on each first-level gate
  3. By minimizing number of inputs on second-level gate
     - This is a side effect of the first reduction
- Minimization methods do not consider cost of input inverters
  - They assume both true and complemented versions of all input variables are available
    - Not always the case in gate-level or ASIC design
    - But, appropriate for PLD-based design where both true and complemented versions of all input variables are available for free

Combinational-Circuit Synthesis: Minimization

- Most minimization methods are based on combining theorems, T10 and T10′

$$\text{given product term} \cdot Y + \text{given product term} \cdot Y' = \text{given product term}$$
$$(\text{given sum term} + Y) \cdot (\text{given sum term} + Y') = \text{given sum term}$$

- Applying this method repeatedly to combine minterms 1, 3, 5, and 7 of prime-number detector shown in Fig. 15

$$\begin{aligned}
F &= \sum_{N_3,N_2,N_1,N_0} (1,3,5,7,2,11,13) \\
&= N_3' \cdot N_2' \cdot N_1' \cdot N_0 + N_3' \cdot N_2' \cdot N_1 \cdot N_0 + N_3' \cdot N_2 \cdot N_1' \cdot N_0 + N_3' \cdot N_2 \cdot N_1 \cdot N_0 + \cdots \\
&= (N_3' \cdot N_2' \cdot N_1' \cdot N_0 + N_3' \cdot N_2' \cdot N_1 \cdot N_0) + (N_3' \cdot N_2 \cdot N_1' \cdot N_0 + N_3' \cdot N_2 \cdot N_1 \cdot N_0) + \cdots \\
&= N_3' \cdot N_2' \cdot N_0 + N_3' \cdot N_2 \cdot N_0 + \cdots \\
&= N_3' \cdot N_0 + \cdots
\end{aligned}$$

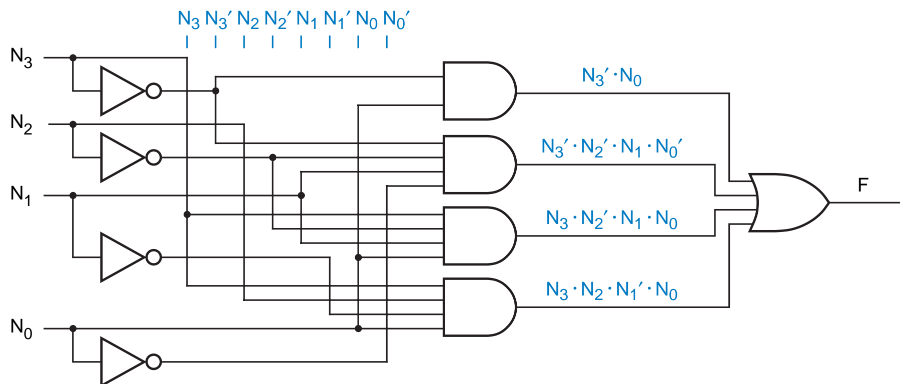


Figure 22: Simplified sum-of-products realization for 4-bit prime-number detector.

- Working more on preceding expression, we could save a couple more first-level gate inputs

- **Karnaugh map**
  - A graphical representation of a logic function's truth table
  - Map for an *n*-input logic function is an array with 2^n cells, one for each possible input combination or minterm
  - Number inside each cell is corresponding minterm number in truth table
    - Truth-table inputs are labeled alphabetically from left to right (e.g., X, Y, Z)
    - E.g., cell 13 in 4-variable map corresponds to truth table row in which $WXYZ = 1101$

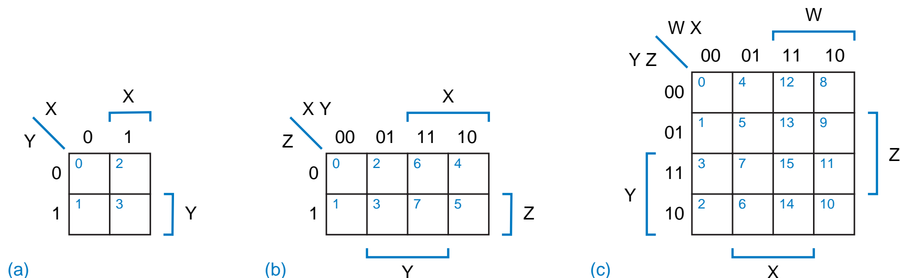


Figure 23: Karnaugh maps: (a) 2-variable; (b) 3-variable; (c) 4-variable.

- To represent a logic function on a Karnaugh map, we copy 1s and 0s from truth table or equivalent to the corresponding cells of map

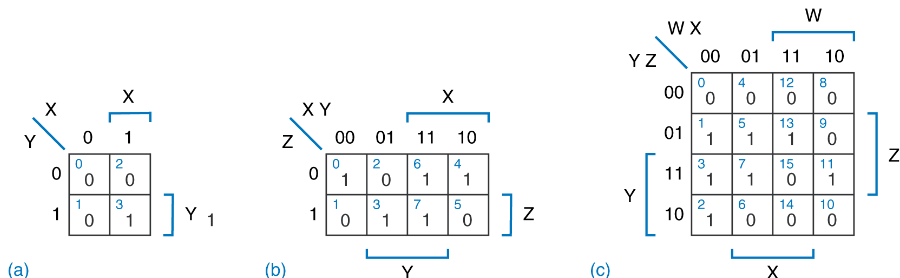


Figure 24: Karnaugh maps for logic functions: (a) $F = \sum_{X,Y}(3)$; (b) $F = \sum_{X,Y,Z}(0, 3, 4, 6, 7)$; (c) $F = \sum_{W,X,Y,Z}(1, 2, 3, 5, 7, 11, 13)$.

Com.-Circuit Synthesis: Minimizing Sums of Products

- Particular order of row and column numbers in a Karnaugh map makes each cell correspond to an input combination that differs from each of its immediately adjacent neighbors in only one variable
  - Corresponding cells on left/right or top/bottom borders also differ in one variable and hence neighbors; e.g., cells 12 and 14 in 4-variable map
- Each input combination with a "1" in truth table corresponds to a minterm in logic function's canonical sum
  - Pairs of adjacent "1" cells in map have minterms that differ in only one variable
  - Thus, minterm pairs can be combined into a single product term

$$\text{term} \cdot Y + \text{term} \cdot Y' = \text{term}$$

  - Thus, we can use a Karnaugh map to simplify canonical sum of a logic function

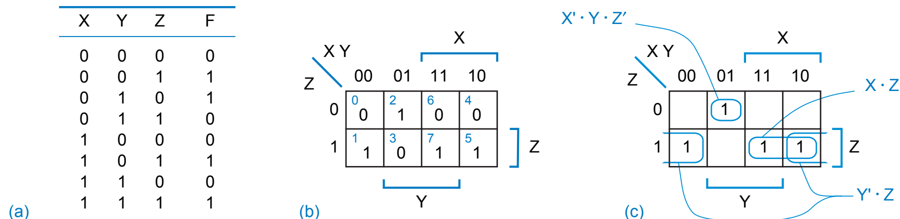


Figure 25: $F = \sum_{X,Y,Z}(1,2,5,7)$: (a) truth table; (b) Karnaugh map; (c) combining adjacent 1-cells.

- In Fig. 25(b)

For cells 5 and 7:

$$
\begin{aligned}
F &= \cdots + X \cdot Y' \cdot Z + X \cdot Y \cdot Z \\
&= \cdots + (X \cdot Z) \cdot Y' + (X \cdot Z) \cdot Y \\
&= \cdots + X \cdot Z
\end{aligned}
$$

For cells 1 and 5:

$$
\begin{aligned}
F &= X' \cdot Y' \cdot Z + X \cdot Y' \cdot Z + \cdots \\
&= X' \cdot (Y' \cdot Z) + X \cdot (Y' \cdot Z) + \cdots \\
&= Y' \cdot Z + \cdots
\end{aligned}
$$

- We can simplify a logic function by first combining pairs of adjacent 1-cells (minterms) wherever possible and then selecting a set of product terms that covers all of 1-cells and summing them
  - Fig. 25(c) shows the result for our example logic function
  - Corresponding AND-OR circuit is shown in Fig. 26

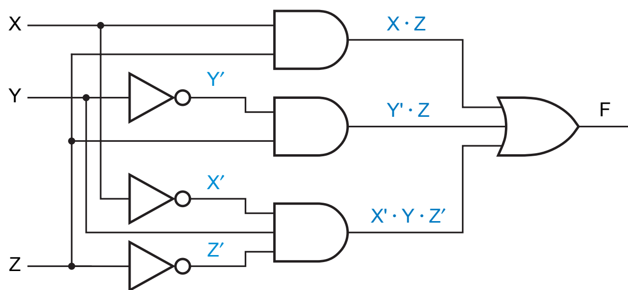


Figure 26: Minimized AND-OR circuit.

Com.-Circuit Synthesis: Minimizing Sums of Products

- In many logic functions, cell-combining procedure can be extended to combine more than two 1-cells into a single product term
  - Number of cells combined is always a power of 2
  - Example

$$
\begin{aligned}
F &= \sum_{X,Y,Z}(0,1,4,5,6) \\
&= X' \cdot Y' \cdot Z' + X' \cdot Y' \cdot Z + X \cdot Y' \cdot Z' + X \cdot Y' \cdot Z + X \cdot Y \cdot Z' \\
&= [(X' \cdot Y') \cdot Z' + (X' \cdot Y') \cdot Z] + [(X \cdot Y') \cdot Z' + (X \cdot Y') \cdot Z] + X \cdot Y \cdot Z' \\
&= X' \cdot Y' + X \cdot Y' + X \cdot Y \cdot Z' \\
&= [X' \cdot (Y') + X \cdot (Y')] + X \cdot Y \cdot Z' \\
&= Y' + X \cdot Y \cdot Z'
\end{aligned}
$$

  - 2^i 1-cells may be combined to form a product term containing $n - i$ literals ($n =$ number of variables in function)
  - A set of 2^i 1-cells are combined if there are i variables that take on all 2^i possible combinations within that set, while remaining $n - i$ variables have the same value throughout that set
    - Corresponding product term has $n - i$ literals, where a variable is complemented if it is 0 in all of 1-cells, and uncomplemented if it is 1

Com.-Circuit Synthesis: Minimizing Sums of Products

- Graphically, we circle *rectangular* sets of 2^i 1s, stretching definition of rectangular to account for wraparound at edges of map
  - For each variable, if a circle covers only areas of map where it is 0, the variable is complemented in product term
  - If a circle covers only areas of map where the variable is 1, the variable is uncomplemented in product term
  - If a circle covers areas of map where the variable is 0 as well as areas where it is 1, the variable does not appear in product term
  - Finally, a sum-of-products expression for a function must contain product terms that cover all of 1s and none of 0s on map
  - By circling largest possible set of 1s, a less expensive realization of logic function is found

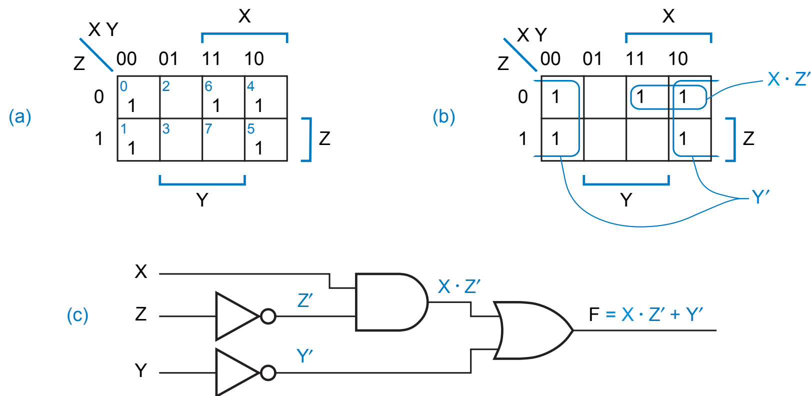
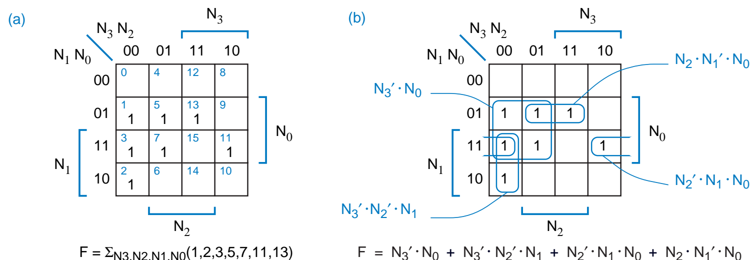


Figure 27: $F = \sum_{X,Y,Z}(0,1,4,5,6)$: (a) initial Karnaugh map; (b) Karnaugh map with circled product terms; (c) AND/OR circuit.

Com.-Circuit Synthesis: Minimizing Sums of Products



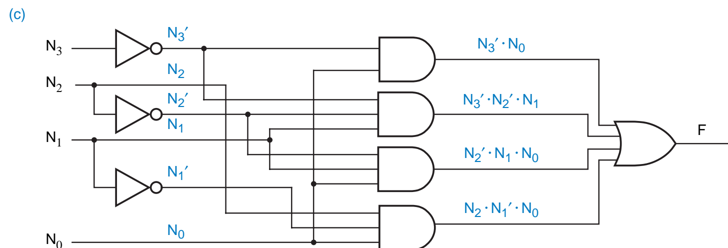


Figure 28: Prime-number detector: (a) initial Karnaugh map; (b) circled product terms; (c) minimized circuit.

Com.-Circuit Synthesis: Minimizing Sums of Products

- **Minimal sum** of a logic function $F(X_1, \ldots, X_n)$
  1. Has the fewest possible product terms
  2. Within constraint 1, has the fewest possible literals
- A logic function $P(X_1, \ldots, X_n)$ **implies** a logic function $F(X_1, \ldots, X_n)$ if for every input combination such that $P = 1$, then $F = 1$ too
  - "P implies F" $\equiv$ "F includes P" $\equiv$ "F covers P" $\equiv P \Rightarrow F$
- **Prime implicant** of a logic function $F(X_1, \ldots, X_n)$
  - A normal product term $P(X_1, \ldots, X_n)$ that implies F, such that if any variable is removed from P, resulting product term does not imply F
  - In Karnaugh map, a prime implicant of F is a circled set of 1-cells, such that if we make it larger (twice as many cells), it covers one or more 0s
- **Prime-implicant theorem**
  - A minimal sum is a sum of prime implicants

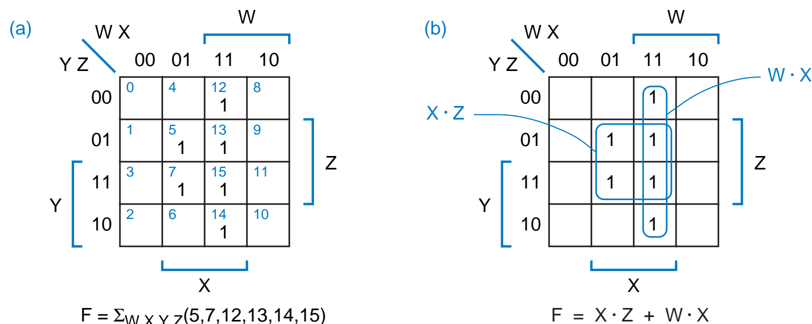


Figure 29: $F = \sum_{W,X,Y,Z}(5, 7, 12, 13, 14, 15)$: (a) Karnaugh map; (b) prime implicants.

- **Complete sum**
  - Sum of all prime implicants of a logic function
  - Is not always minimal
  - E.g., logic function shown in Fig. 30 has five prime implicants, but minimal sum includes only three of them

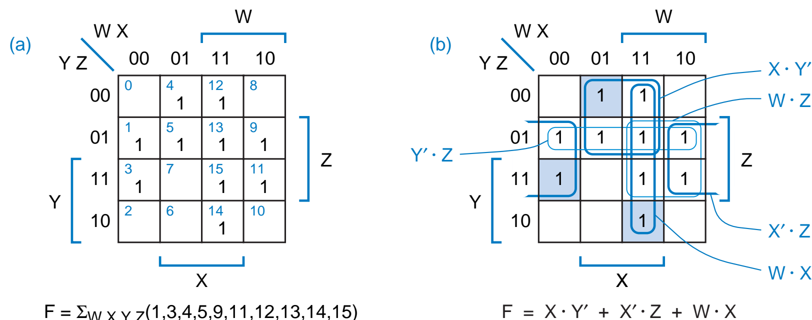


Figure 30: $F = \sum_{W,X,Y,Z}(1,3,4,5,9,11,12,13,14,15)$: (a) Karnaugh map; (b) prime implicants and distinguished 1-cells.

- **Distinguished 1-cell** of a logic function
  - An input combination that is covered by only one prime implicant
- **Essential prime implicant** of a logic function
  - A prime implicant that covers one or more distinguished 1-cells

Com.-Circuit Synthesis: Minimizing Sums of Products

- First step in prime-implicant selection process
  - Identify distinguished 1-cells and corresponding essential prime implicants, and include them in minimal sum
  - In Fig. 30, three distinguished 1-cells are shaded, and corresponding essential prime implicants are circled with heavier lines
    - All of 1-cells are covered by essential prime implicants, so we need go no further
  - In Fig. 31, all of prime implicants are essential, and so all are included in minimal sum

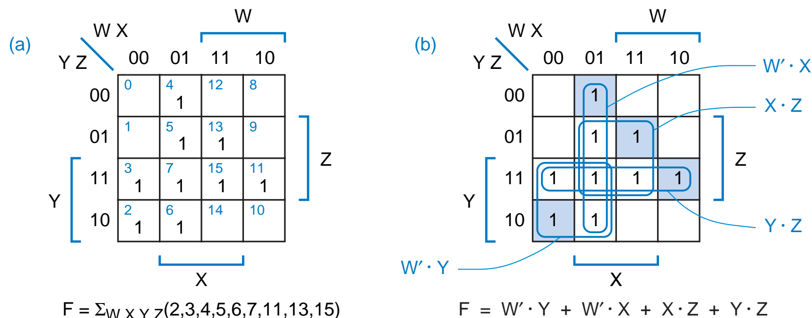


Figure 31: $F = \sum_{W,X,Y,Z}(2,3,4,5,6,7,11,13,15)$: (a) Karnaugh map; (b) prime implicants and distinguished 1-cells.

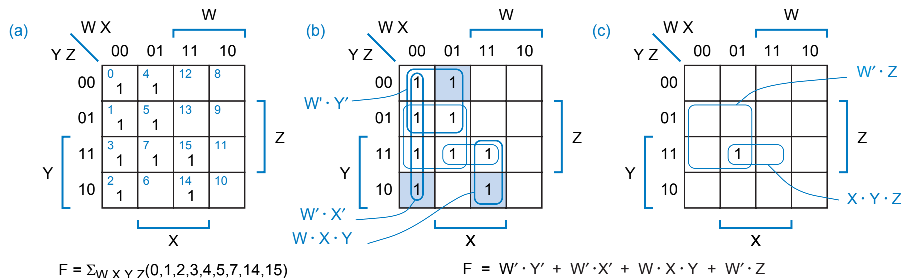


Figure 32: $F = \sum_{W,X,Y,Z}(0,1,2,3,4,5,7,14,15)$: (a) Karnaugh map; (b) prime implicants and distinguished 1-cells; (c) reduced map after removal of essential prime implicants and covered 1-cells.

- In Fig. 32, by removing essential prime implicants and the 1-cells they cover, we obtain a reduced map with only a single 1-cell and two prime implicants that cover it
  - We use $W' \cdot Z$ product term because it has fewer inputs and therefore lower cost

- **Eclipse**
  - Given two prime implicants P and Q in a reduced map, P is said to eclipse Q ($P \supseteq Q$) if P covers at least all 1-cells covered by Q
  - If P eclipses Q, then Q can be ignored when finding a minimal sum
  - In Fig. 33(c), $X \cdot Y \cdot Z$ eclipses the other two prime implicants
    - $X \cdot Y \cdot Z$ is a **secondary essential prime implicant** that must be included in minimal sum

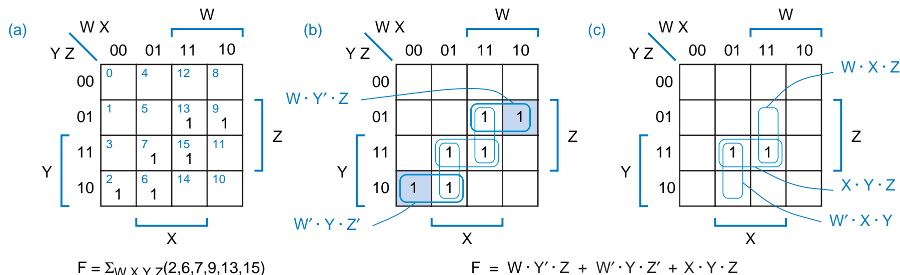


Figure 33: $F = \sum_{W,X,Y,Z}(2,6,7,9,13,15)$: (a) Karnaugh map; (b) prime implicants and distinguished 1-cells; (c) reduced map after removal of essential prime implicants and covered 1-cells.

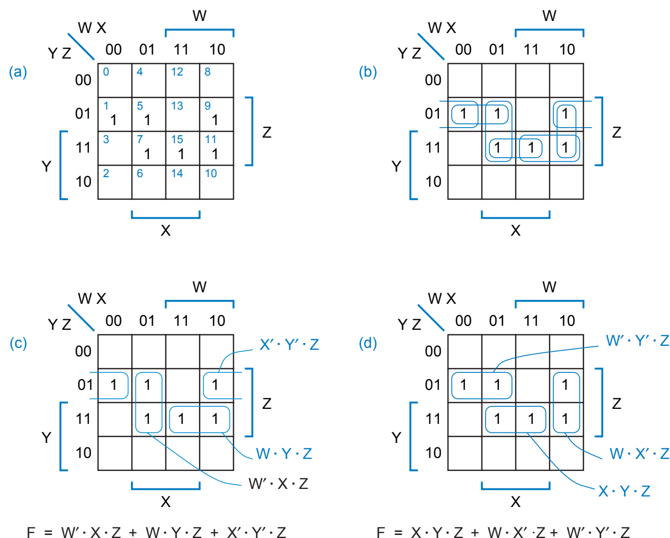


Figure 34: $F = \sum_{W,X,Y,Z}(1, 5, 7, 9, 11, 15)$: (a) Karnaugh map; (b) prime implicants (no essential); (c) a minimal sum; (d) another minimal sum.

- Using principle of duality, we can minimize product-of-sums expressions by looking at 0s on a Karnaugh map
  - Each 0 on map corresponds to a maxterm in canonical product of logic function
  - To find **minimal product**, we write sum terms corresponding to circled sets of 0s
- In Fig. 35

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

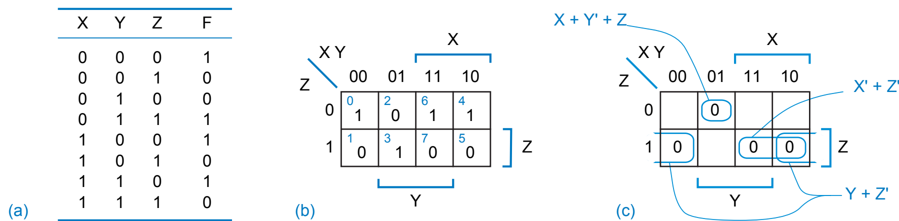


Figure 35: $F = \prod_{X,Y,Z}(1,2,5,7)$: (a) truth table; (b) Karnaugh map; (c) combining adjacent 0-cells.

- Indirect method to find minimal product
  - For circled sets of 0s in Karnaugh map, write *product terms*
  - Equate F' to *minimal sum*
  - Use DeMorgan's theorem to find F
  - E.g., for Fig. 35(c), product terms of circled 0s are: $X' \cdot Y \cdot Z'$, $X \cdot Z$, $Y' \cdot Z$

$$F' = X' \cdot Y \cdot Z' + X \cdot Z + Y' \cdot Z$$
$$[F']' = [X' \cdot Y \cdot Z' + X \cdot Z + Y' \cdot Z]'$$
$$F = (X + Y' + Z) \cdot (X' + Z') \cdot (Y + Z')$$

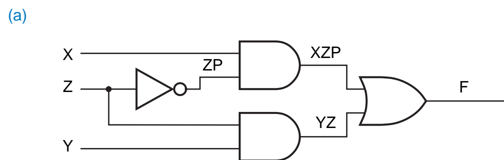
- **PLD minimization**
  - PLDs have an AND-OR array corresponding to sum-of-products form
  - Most PLDs, also have an inverter/buffer at output of AND-OR array, which can be programmed to invert or not
    - Thus, PLD can utilize the equivalent of minimal sum by using AND-OR array to realize complement of desired function and then programming inverter/buffer to invert
    - Most logic-minimization programs for PLDs find both minimal sum and minimal product and select the one that requires fewer terms

Timing Hazards

- Predicting **steady-state behavior** of combinational logic circuits
  - Predicting a circuit's output as a function of its inputs under assumption that inputs have been stable for a long time, relative to delays in circuit's electronics
    - Circuit delay is ignored
  - But actual delay from an input change to corresponding output change in a real circuit is nonzero
- **Transient behavior** of a combinational logic circuit
  - Considers circuit delays
  - May differ from what is predicted by a steady-state analysis
  - A circuit's output may produce a short pulse, called a **glitch**, at a time when steady-state analysis predicts that output should not change
  - A **hazard** exists when a circuit has possibility of producing such a glitch
  - A logic designer must eliminate hazards

- **Static-1 hazard**
  - A pair of input combinations that
    1. Differ in only one input variable
    2. Both give a 1 output

    such that it is possible for a momentary 0 output to occur during a transition in the differing input variable



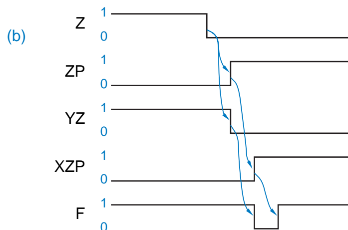


Figure 36: Circuit with a static-1 hazard: (a) logic diagram; (b) timing diagram ($X = 1, Y = 1, Z : 1 \rightarrow 0$, propagation delay through each gate or inverter is one unit time).

- **Static-0 hazard**
  - A pair of input combinations that
    1. Differ in only one input variable
    2. Both give a 0 output

    such that it is possible for a momentary 1 output to occur during a transition in the differing input variable

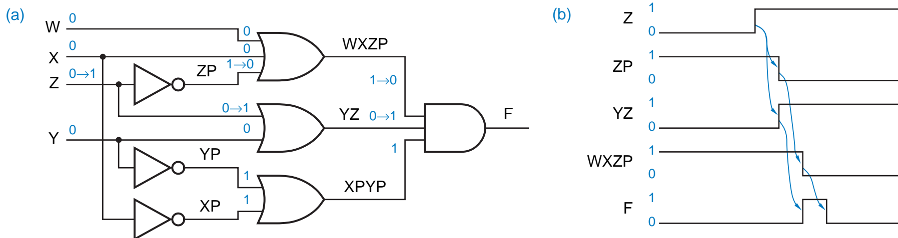
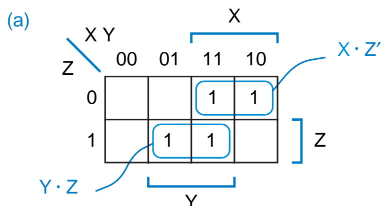


Figure 37: Circuit with static-0 hazards: (a) logic diagram; (b) timing diagram.

Timing Hazards: Static Hazards

- A Karnaugh map can be used to detect static hazards in a two-level sum-of-products or product-of-sums circuit
- A properly designed two-level sum-of-products (AND-OR) circuit has no static-0 hazards
  - A static-0 hazard would exist only if both a variable and its complement were connected to the same AND gate, which would be silly
  - But the circuit may have static-1 hazards



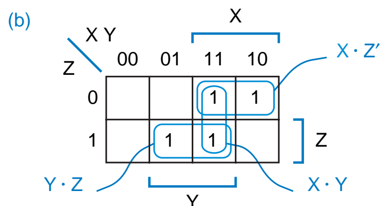


Figure 38: Karnaugh map for the circuit of Fig. 36: (a) as originally designed; (b) with static-1 hazard eliminated.

- In Fig. 38
  - There is no single product term that covers both input combinations $X, Y, Z = 111$ and $X, Y, Z = 110$
    - Possible for output to glitch momentarily to 0 if AND gate output that covers one of combinations goes to 0 before AND gate output covering the other input combination goes to 1
  - To eliminate hazard, include an extra product term (AND gate) to cover hazardous input pair
    - The extra product term to be added is consensus of the two original terms

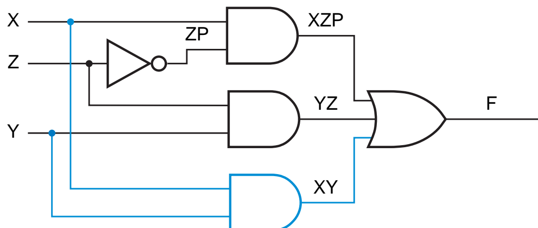


Figure 39: Circuit of Fig. 36 with static-1 hazard eliminated.

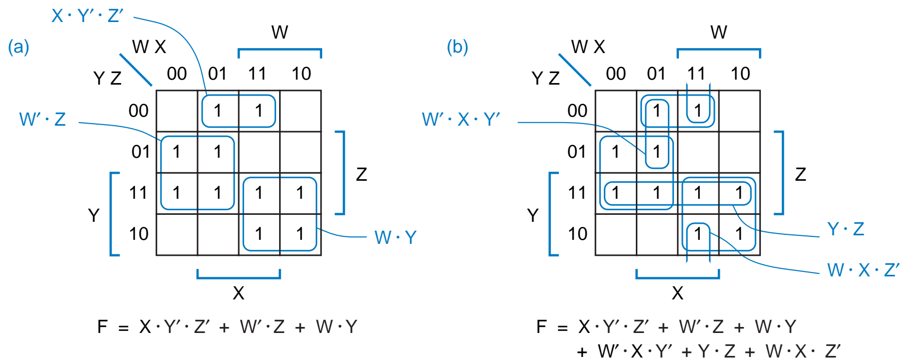
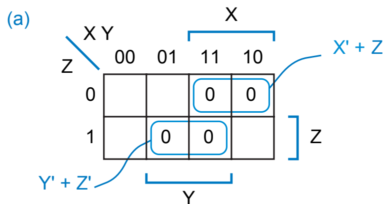


Figure 40: Karnaugh map for another sum-of-products circuit: (a) as originally designed; (b) with extra product terms to cover static-1 hazards.

- A properly designed two-level product-of-sums (OR-AND) circuit has no static-1 hazards
  - But, it may have static-0 hazards
    - These hazards can be detected and eliminated by studying adjacent 0s in Karnaugh map



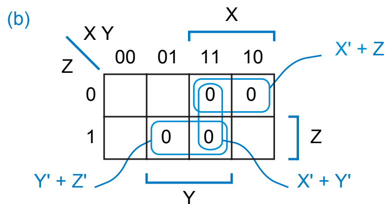


Figure 41: Karnaugh map for a product-of-sums circuit: (a) as originally designed; (b) with extra sum term to cover the static-0 hazard.

- **Dynamic hazard**
  - Possibility of an output changing more than once as the result of a single input transition
  - Multiple output transitions can occur if there are multiple paths with different delays from the changing input to the changing output

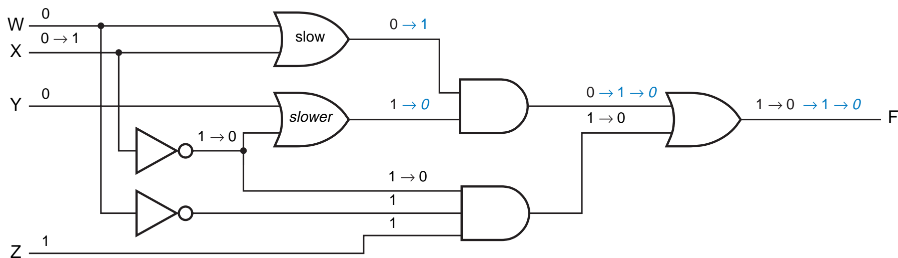


Figure 42: Circuit with a dynamic hazard.

Timing Hazards: Dynamic Hazards

- In Fig. 42
  - Three different paths with different delays from input X to output F
  - If all of gates except the two marked "slow" and "slower" are very fast, the transitions shown in black occur first, and output goes to 0
  - Then, output of "slow" OR gate changes, creating transitions shown in nonitalic color, and output goes to 1
  - Finally, output of "slower" OR gate changes, creating transitions shown in italic color, and output goes to 0
- Dynamic hazards do not occur in a properly designed two-level AND-OR or OR-AND circuit
  - In such a circuit, no variable and its complement are connected to the same first-level gate

References

John F. Wakerly, *Digital Design: Principles and Practices (4th Edition)*, Prentice Hall, 2005.