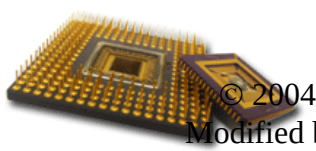
Lecture 3 – Logic Gate Level Minimization

Lan-Da Van (范倫達), Ph. D.
Department of Computer Science
National Chiao Tung University
Taiwan, R.O.C.
Fall, 2007

Source: Charles Kime & Thomas Kaminski

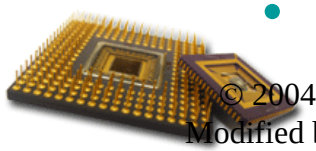
Outline

- **Circuit Optimization**
  - **Two-Level Optimization**
  - **Map Manipulation**
  - **Multi-Level Circuit Optimization**
- **Additional Gates and Circuits**
  - **Other Gate Types**
  - **Exclusive-OR Operator and Gates**
  - **High-Impedance Outputs**



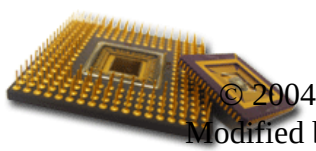
Circuit Optimization

- Goal: To obtain the simplest implementation for a given function
- Optimization is a more formal approach to simplification that is performed using a specific procedure or algorithm
- Optimization requires a cost criterion to measure the simplicity of a circuit
- Three distinct cost criteria will be used:
  - Literal cost (L)
  - Gate input cost (G)
  - Gate input cost with NOTs (GN)



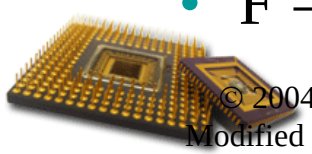
Literal Cost

- Literal – a variable or its complement
- Literal cost – the number of literal appearances in a Boolean expression corresponding to the logic circuit diagram
- Examples:
  - $F = BD + A\bar{B}C + A\bar{C}\bar{D}$ L = 8
  - $F = BD + A\bar{B}C + A\bar{B}\bar{D} + AB\bar{C}$ L = 11
  - $F = (A + B)(A + D)(B + C + \bar{D})(\bar{B} + \bar{C} + D)$ L = 10
  - Which solution is best? (first verify the equality)



Gate Input Cost

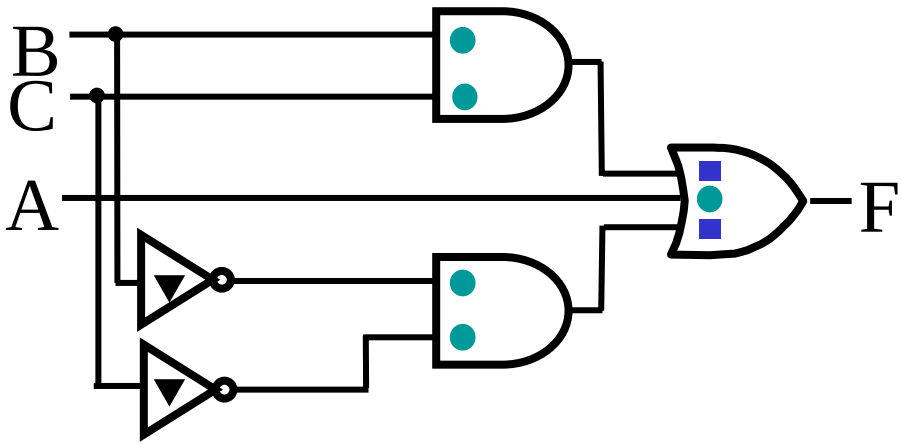
- Gate input costs - the number of inputs to the gates in the implementation corresponding exactly to the given equation or equations. (G - inverters not counted, GN - inverters counted)
- For SOP and POS equations, it can be found from the equation(s) by finding the sum of:
  - all literal appearances
  - the number of terms excluding terms consisting only of a single literal,(G) and
  - optionally, the number of distinct complemented single literals (GN).
- Example: which solution is best ?
  - $F = BD + A\overline{B}C + A\overline{C}\,\overline{D}$ G = 11, GN = 14
  - $F = BD + A\overline{B}C + A\overline{B}\,\overline{D} + AB\overline{C}$ G = 15, GN = 19
  - $F = (A + B)(A + D)(B + C + \overline{D})(\overline{B} + \overline{C} + D)$ G = 14, GN = 17



Cost Criteria (1/2)

- **Example 1:**
- **$F = A + BC + \overline{B}\,\overline{C}$**

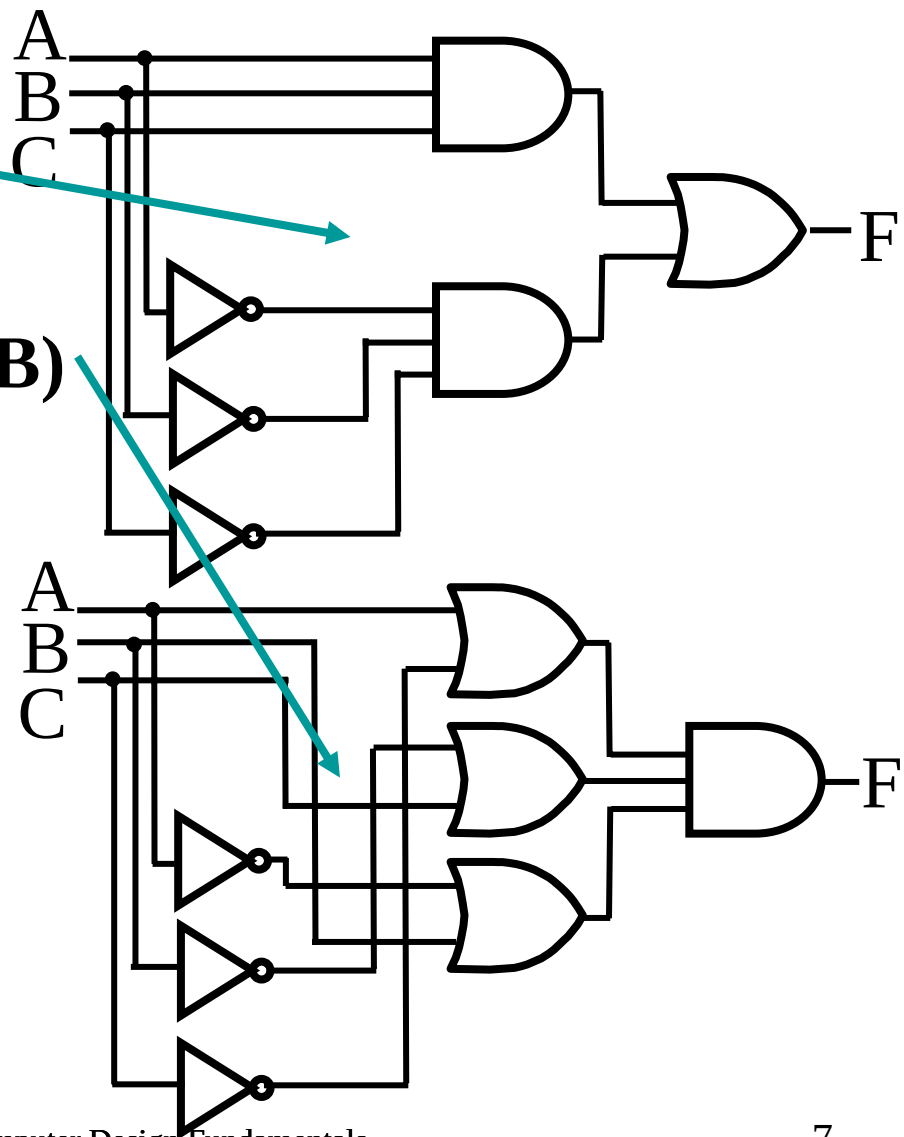
GN = G + 2 = 9
L = 5
G = L + 2 = 7



- **L (literal count) counts the AND inputs and the single literal OR input.**
- **G (gate input count) adds the remaining OR gate inputs**
- **GN(gate input count with NOTs) adds the inverter inputs**

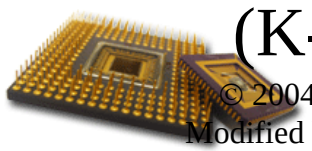
Cost Criteria (2/2)

- **Example 2:**
- **$F = A\,B\,C + \bar{A}\,\bar{B}\,\bar{C}$**
- **L = 6 G = 8 GN = 11**
- **$F = (A + \bar{C})(\bar{B} + C)(\bar{A} + B)$**
- **L = 6 G = 9 GN = 12**
- **Same function and same literal cost**
- **But first circuit has better gate input count and better gate input count with NOTs**
- **Select the former design!**



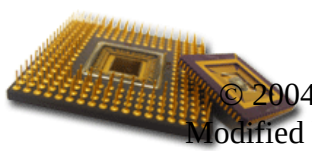
Boolean Function Optimization

- Minimizing the gate input (or literal) cost of a (a set of) Boolean equation(s) is to reduce the circuit cost.
- We choose gate input cost.
- Boolean algebra and graphical techniques are tools to minimize cost criteria values.
- Some important questions:
  - When do we stop trying to reduce the cost?
  - Do we know when we have a minimum cost?
- Treat optimum or near-optimum cost functions for two-level (SOP and POS) circuits first.
- Introduce a graphical technique using Karnaugh maps (K-maps, for short)



Karnaugh Maps (K-map)

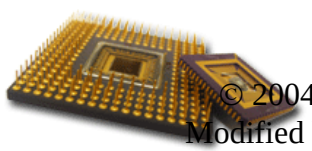
- A K-map is a collection of squares
  - Each square represents a minterm
  - The collection of squares is a graphical representation of a Boolean function
  - Adjacent squares differ in the value of one variable
  - Alternative algebraic expressions for the same function are derived by recognizing patterns of squares
- The K-map can be viewed as
  - A reorganized version of the truth table
  - A topologically-warped Venn diagram as used to visualize sets in algebra of sets



Some Uses of K-Maps

- Provide a means for:
  - Finding optimum or near optimum
    - SOP and POS standard forms, and
    - two-level AND/OR and OR/AND circuit implementations

    for functions with small numbers of variables
  - Visualizing concepts related to manipulating Boolean expressions, and
  - Demonstrating concepts used by computer-aided design programs to simplify large circuits



Two Variable Maps

- A 2-variable Karnaugh Map:
  - Note that minterm m_0 and minterm m_1 are "adjacent" and differ in the value of the variable y
  - Similarly, minterm m_0 and minterm m_2 differ in the x variable.
  - Also, m_1 and m_3 differ in the x variable as well.
  - Finally, m_2 and m_3 differ in the value of the variable y.

	y = 0	y = 1
x = 0	$m_0 = \bar{x}\,\bar{y}$	$m_1 = \bar{x}\,y$
x = 1	$m_2 = x\,\bar{y}$	$m_3 = x\,y$

K-Map and Truth Tables

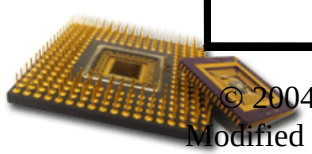
- The K-Map is just a different form of the truth table.
- Example – Two variable function:
  - We choose a,b,c and d from the set {0,1} to implement a particular function, F(x,y).

Function Table

Input Values (x,y)	Function Value F(x,y)
0 0	a
0 1	b
1 0	c
1 1	d

K-Map

	y = 0	y = 1
x = 0	a	b
x = 1	c	d



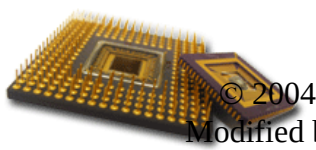
K-Map Function Representation

- Example: F(x,y) = x

F = x	y = 0	y = 1
x = 0	0	0
x = 1	1	1

- For function F(x,y), the two adjacent cells containing 1's can be combined using the Minimization Theorem:

$$F(x, y) = x\bar{y} + xy = x$$



K-Map Function Representation

- Example: G(x,y) = x + y

G = x+y	y = 0	y = 1
x = 0	0	1
x = 1	1	1

reduce *y* reduce *x*

- For G(x,y), two pairs of adjacent cells containing 1's can be combined using the Minimization Theorem:

$$G(x,y) = (x\bar{y} + xy) + (xy + \bar{x}y) = x + y$$

Duplicate xy

Three Variable Maps

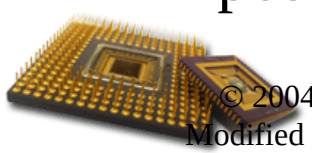
- A three-variable K-map:

	yz=00	yz=01	yz=11	yz=10
x=0	m_0	m_1	m_3	m_2
x=1	m_4	m_5	m_7	m_6

- Where each minterm corresponds to the product terms:

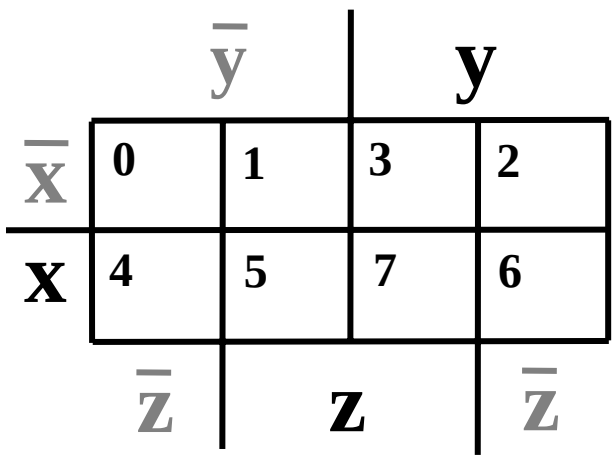
	yz=00	yz=01	yz=11	yz=10
x=0	$\bar{x}\bar{y}\bar{z}$	$\bar{x}\bar{y}z$	$\bar{x}yz$	$\bar{x}y\bar{z}$
x=1	$x\bar{y}\bar{z}$	$x\bar{y}z$	xyz	$xy\bar{z}$

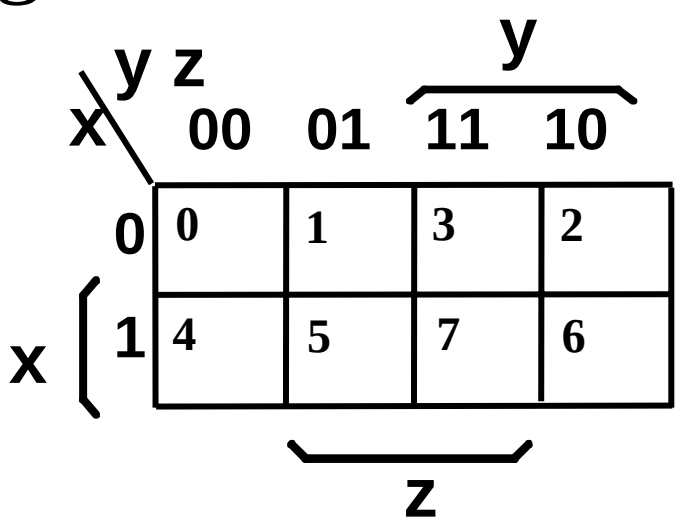
- Note that if the binary value for an index differs in one bit position, the minterms are adjacent on the K-Map



Alternative Map Labeling

- Map use largely involves:
  - Entering values into the map, and
  - Reading off product terms from the map.
- Alternate labelings are useful:



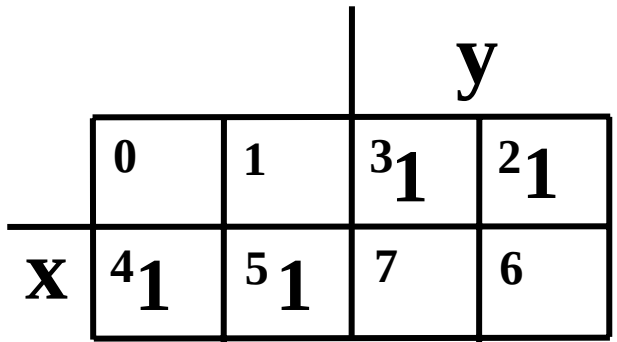


Example Functions

- By convention, we represent the minterms of F by a "1" in the map and leave the minterms of $\overline{F}$ blank
- Example:

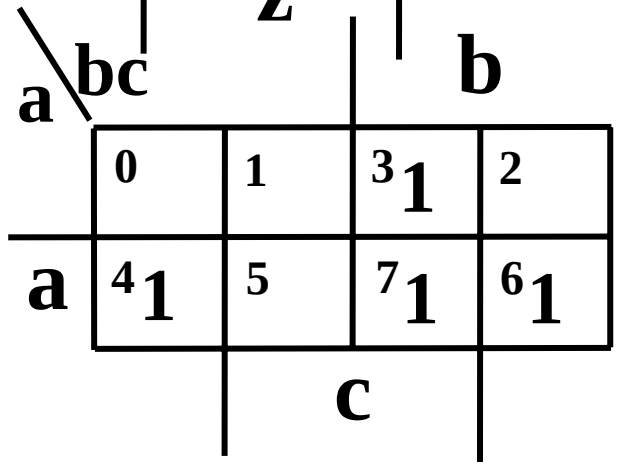
$$F(x, y, z) = \Sigma_m(2,3,4,5)$$

$$F(x,y,z)=?$$



- Example:

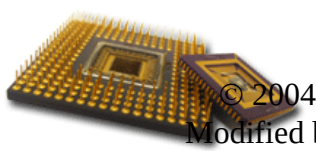
$$G(a, b, c) = \Sigma_m(3,4,6,7)$$



- Learn the locations of the 8 indices based on the variable order shown (x, most significant and z, least significant) on the map boundaries

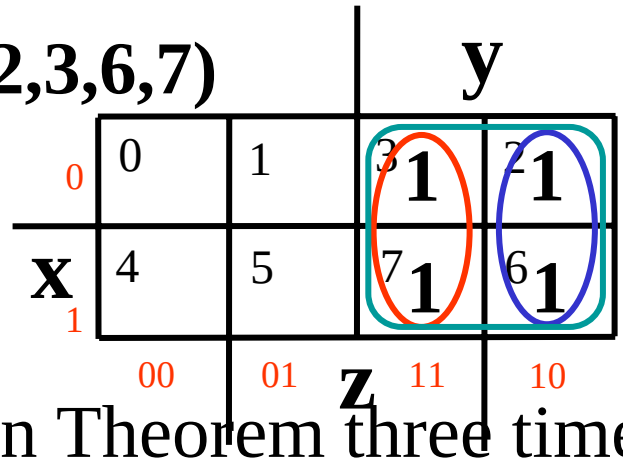
Combining Squares

- By combining squares, we reduce number of literals in a product term, reducing the literal cost, thereby reducing the other two cost criteria
- On a 3-variable K-Map:
  - One square represents a minterm with three variables
  - Two adjacent squares represent a product term with two variables (apply identities 7,14 one time)
  - Four “adjacent” terms represent a product term with one variable (apply identities 7,14 three times)
  - Eight “adjacent” terms is the function of all ones (no variables) = 1. (apply identities 7,14 seven times)



Combining Squares: Example

- Example: Let $F = \Sigma m(2,3,6,7)$



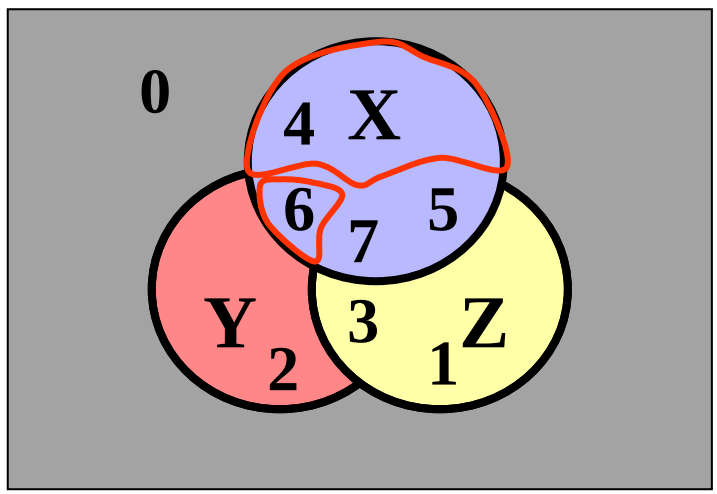
- Applying the Minimization Theorem three times:

$$F(x,y,z) = \bar{x}yz + xyz + \bar{x}y\bar{z} + xy\bar{z}$$
$$= yz + y\bar{z}$$
$$= y$$

- Thus the four terms that form a 2×2 square correspond to the term "y".

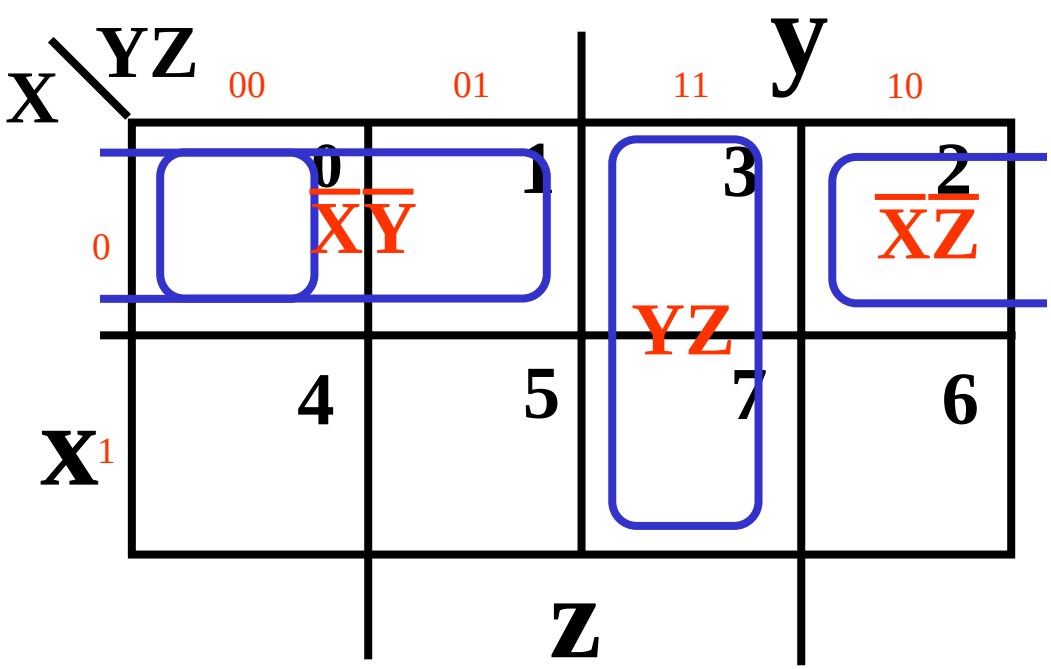
Three-Variable Maps (1/4)

- Topological warps of 3-variable K-maps that show *all* adjacencies:
  - Venn Diagram



Three-Variable Maps (2/4)

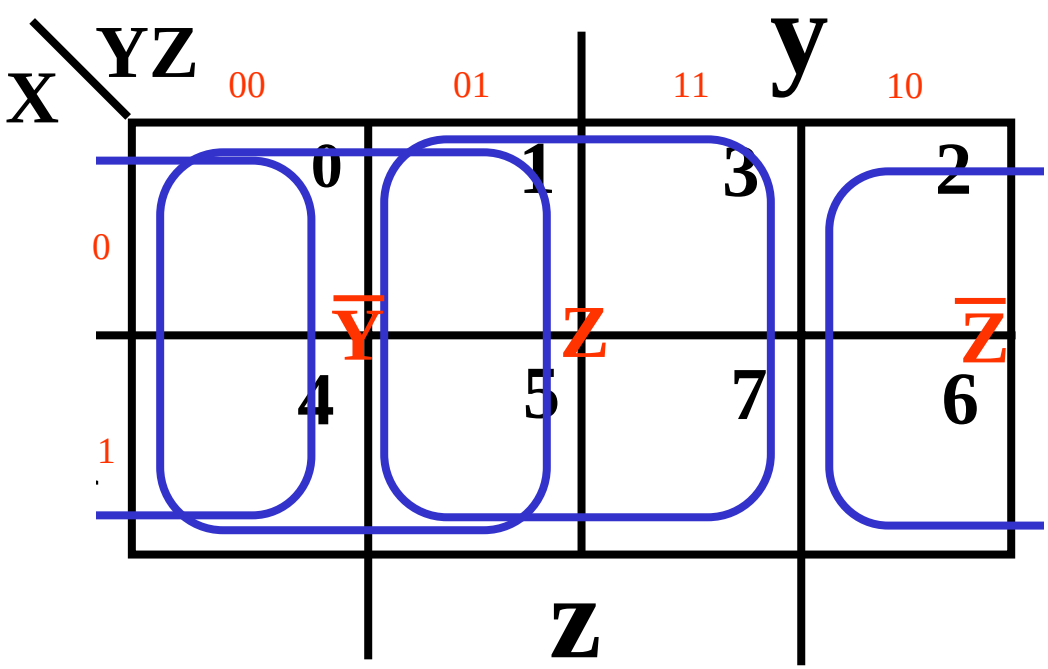
- Example Shapes of 2-cell Rectangles:



- Read off the product terms for the rectangles shown

Three-Variable Maps (3/4)

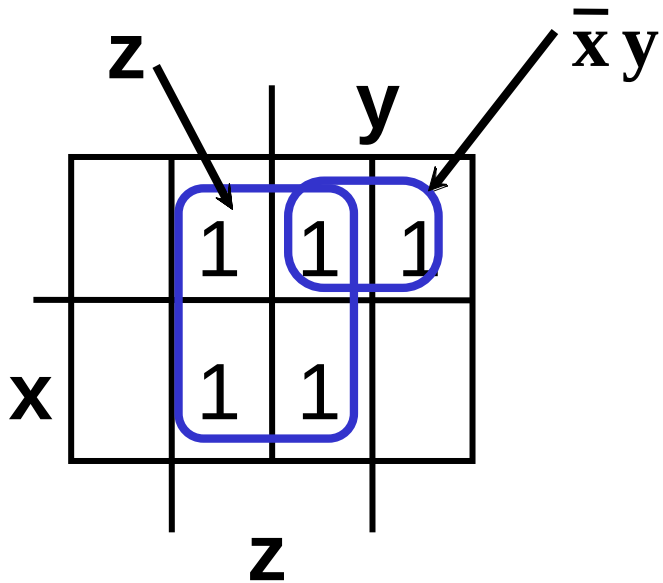
- Example Shapes of 4-cell Rectangles:



- Read off the product terms for the rectangles shown

Three Variable Maps (4/4)

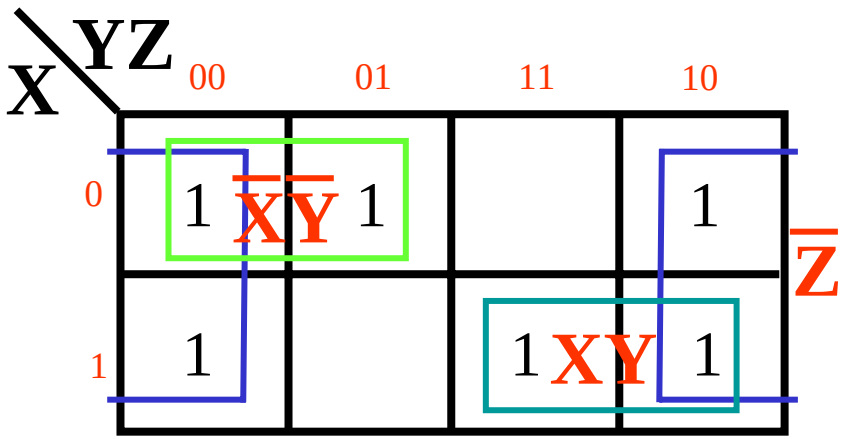
- K-Maps can be used to simplify Boolean functions by systematic methods. Terms are selected to cover the "1s"in the map.
- Example: Simplify $\mathbf{F(x, y, z) = \Sigma_m(1,2,3,5,7)}$



$$\mathbf{F(x, y, z) = z + \bar{x}\, y}$$

Three-Variable Map Simplification

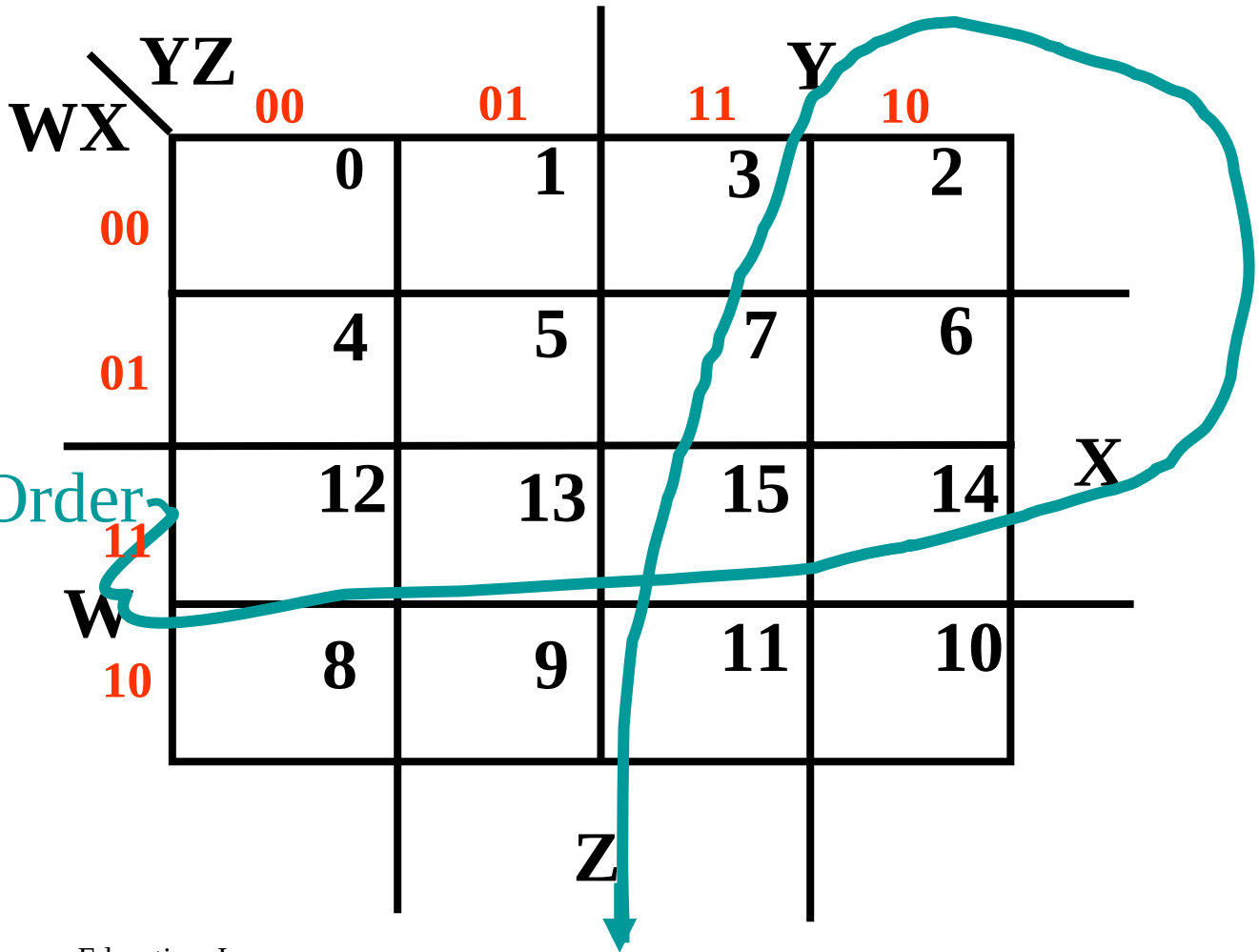
- **Use a K-map to find an optimum SOP equation for $F(X, Y, Z) = \Sigma_m(0,1,2,4,6,7)$**



$$F = \overline{X}\,\overline{Y} + XY + \overline{Z}$$

Four Variable Maps

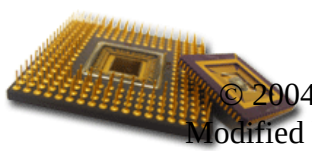
- Map and location of minterms:



Variable Order

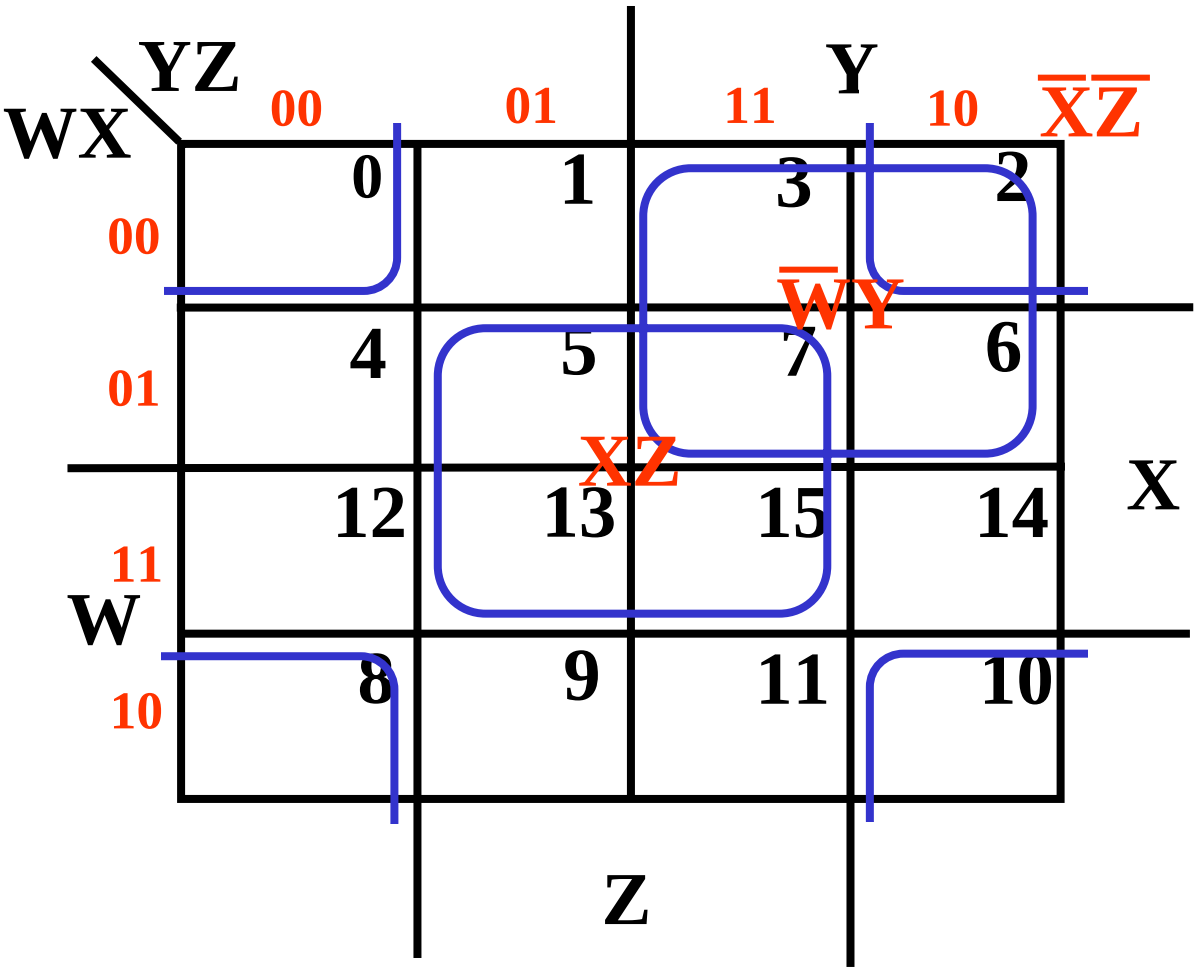
Four Variable Terms

- Four variable maps can have rectangles corresponding to:
  - A single 1 = 4 variables, (i.e. Minterm)
  - Two 1s = 3 variables,
  - Four 1s = 2 variables
  - Eight 1s = 1 variable,
  - Sixteen 1s = zero variables (i.e. Constant "1")



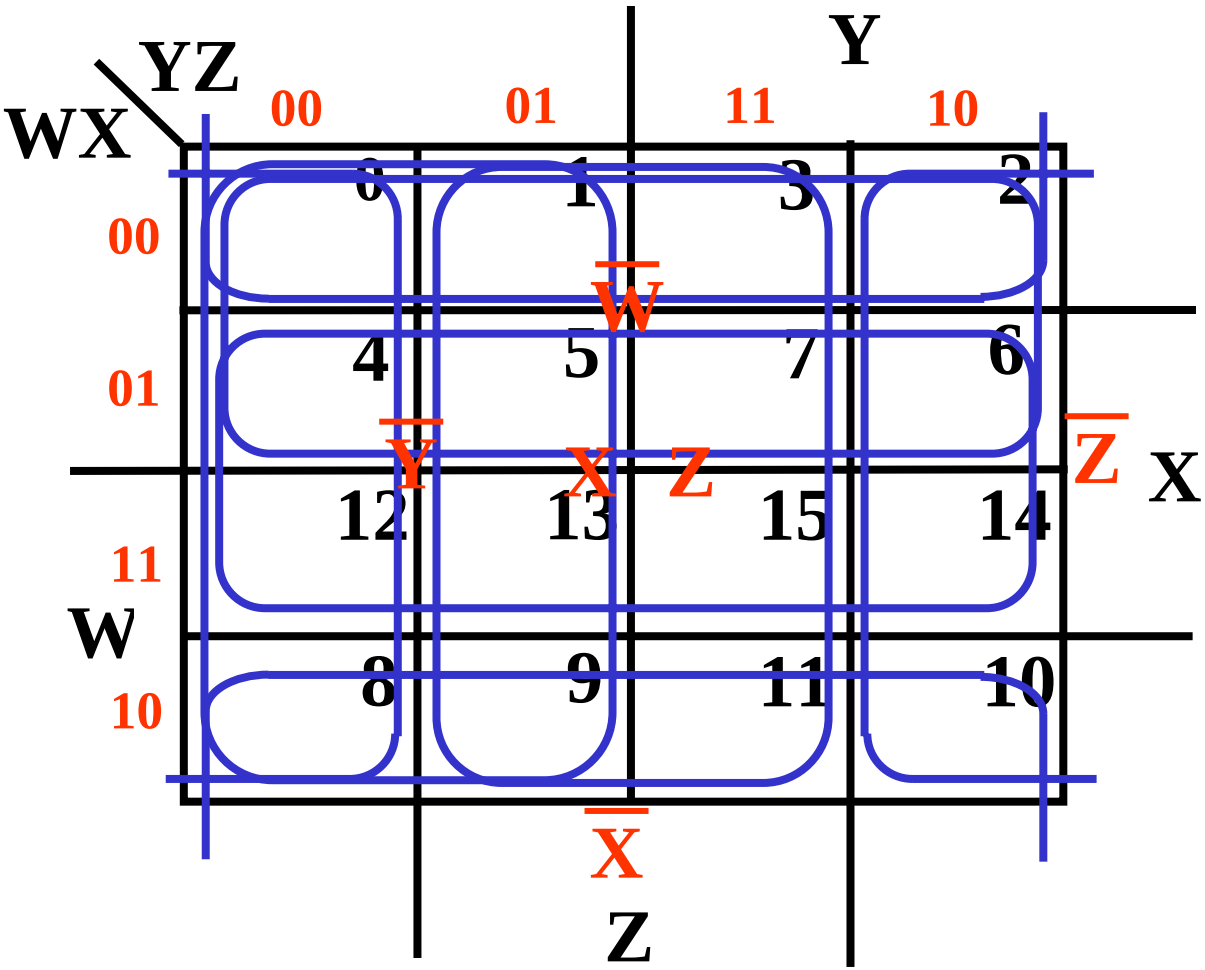
Four-Variable Maps

- Example Shapes of Rectangles:



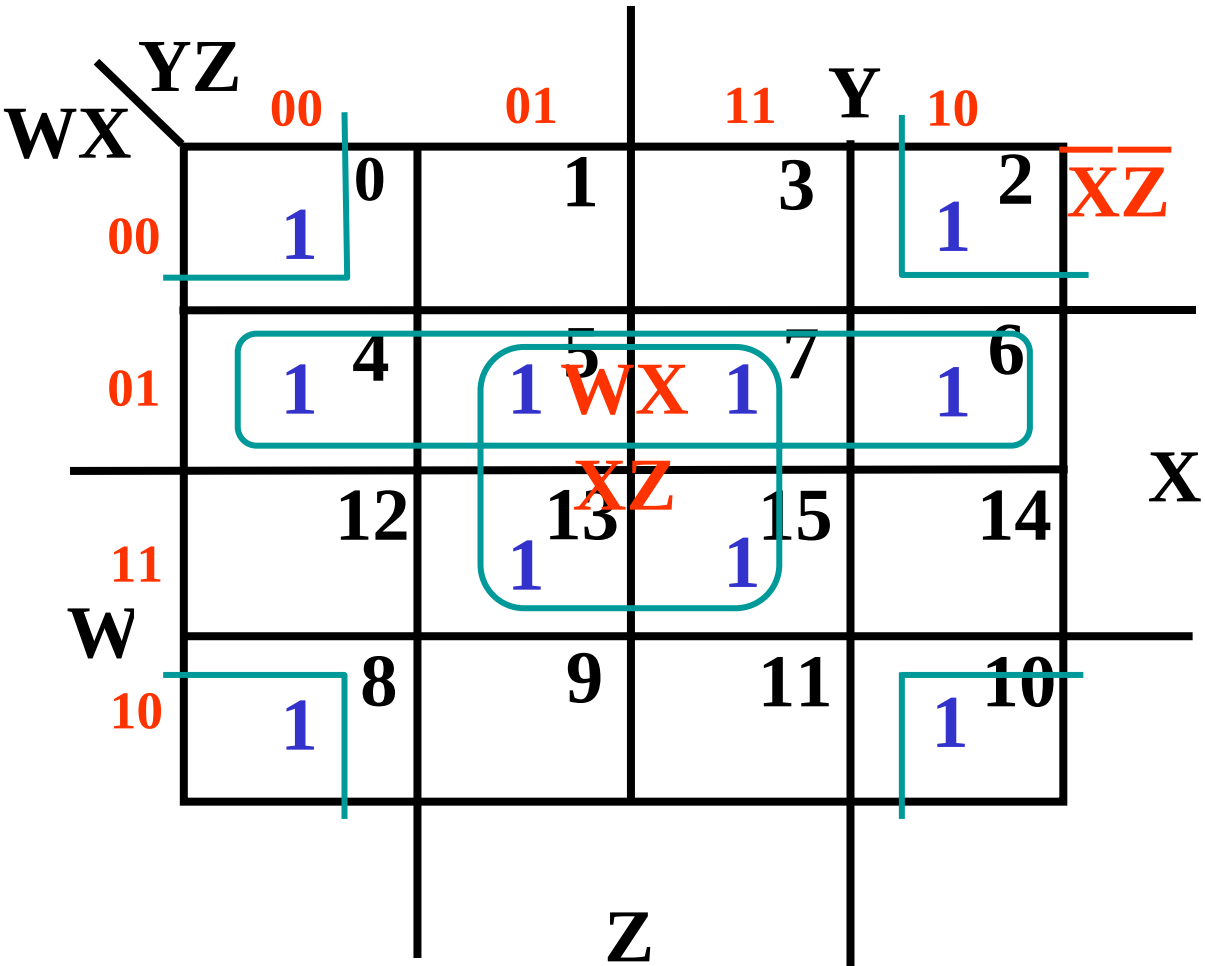
Four-Variable Maps

- Example Shapes of Rectangles:



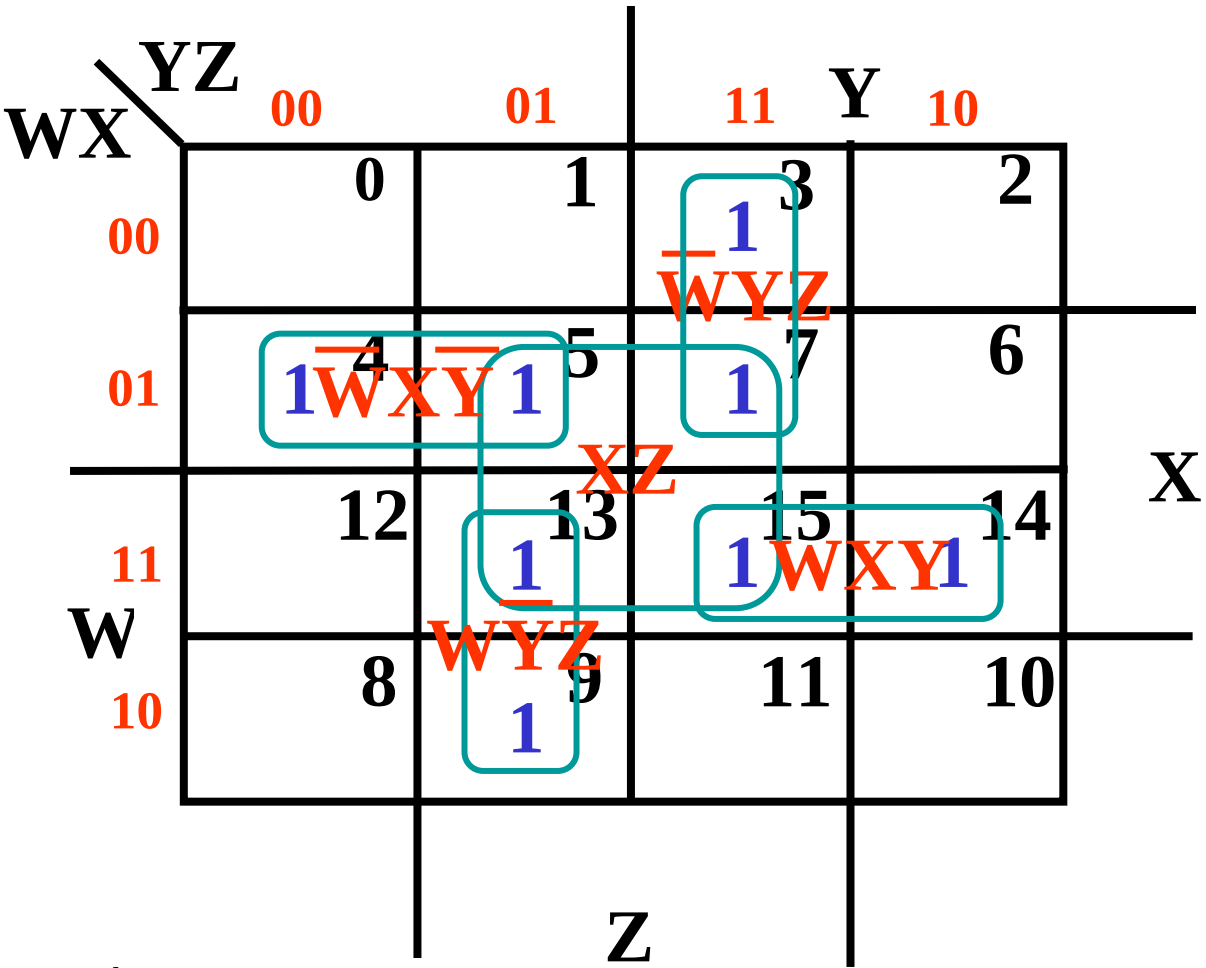
Four-Variable Map Simplification (1/2)

- $F(W, X, Y, Z) = \Sigma_m(0, 2, 4, 5, 6, 7, 8, 10, 13, 15)$



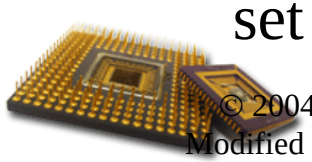
Four-Variable Map Simplification (2/2)

- $F(W, X, Y, Z) = \sum_m(3,4,5,7,9,13,14,15)$



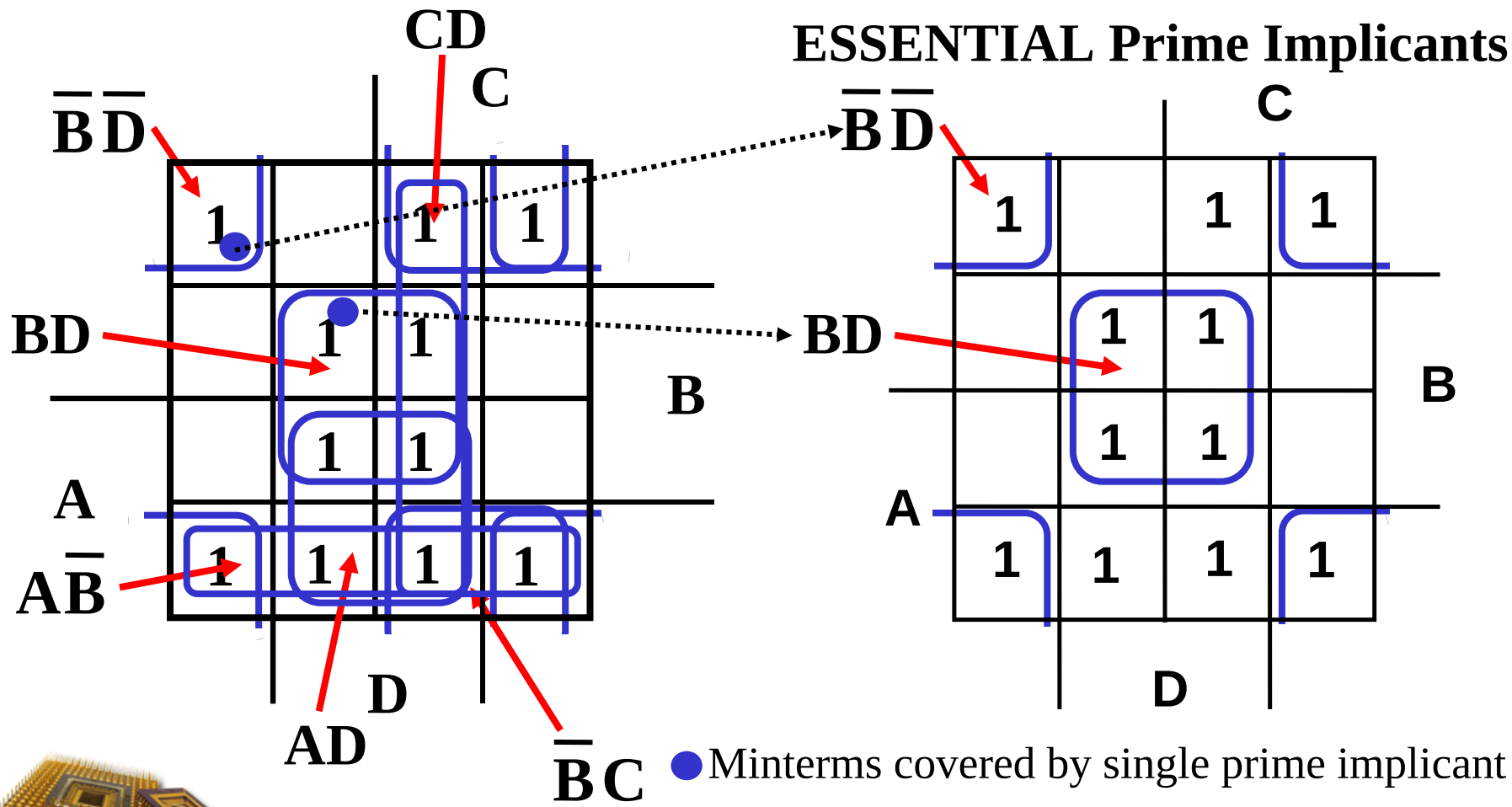
Systematic Simplification

- *Implicant* – a product term of which if the function has the value 1 for all minterms.
- A *Prime Implicant* is a product term obtained by combining the maximum possible number of adjacent squares in the map into a rectangle with the number of squares a power of 2 – remove any literal → not a implicant.
- A prime implicant is called an *Essential Prime Implicant* if it is the only prime implicant that covers (includes) one or more minterms.
- Prime Implicants and Essential Prime Implicants can be determined by inspection of a K-Map.
- A set of prime implicants *"covers all minterms"* if, for each minterm of the function, at least one prime implicant in the set of prime implicants includes the minterm.



Example of Prime Implicants

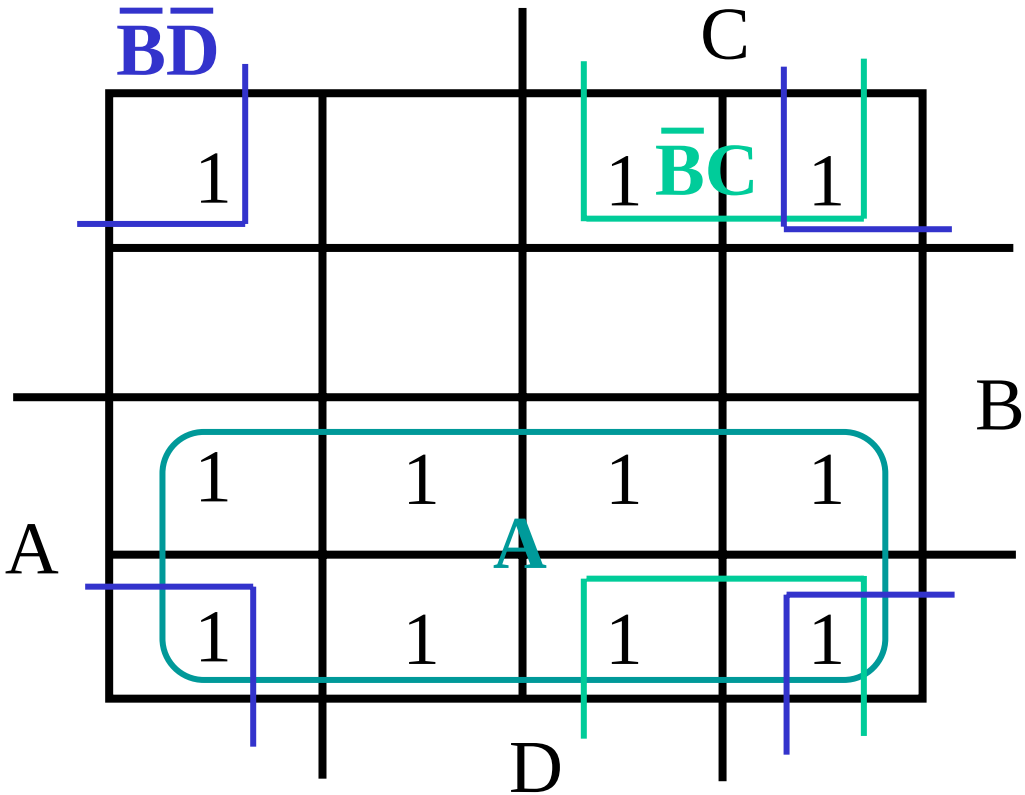
- **Find ALL Prime Implicants**



Prime Implicant Practice

- Find all prime implicants for:

$$F(A, B, C, D) = \Sigma_m(0,2,3,8,9,10,11,12,13,14,15)$$

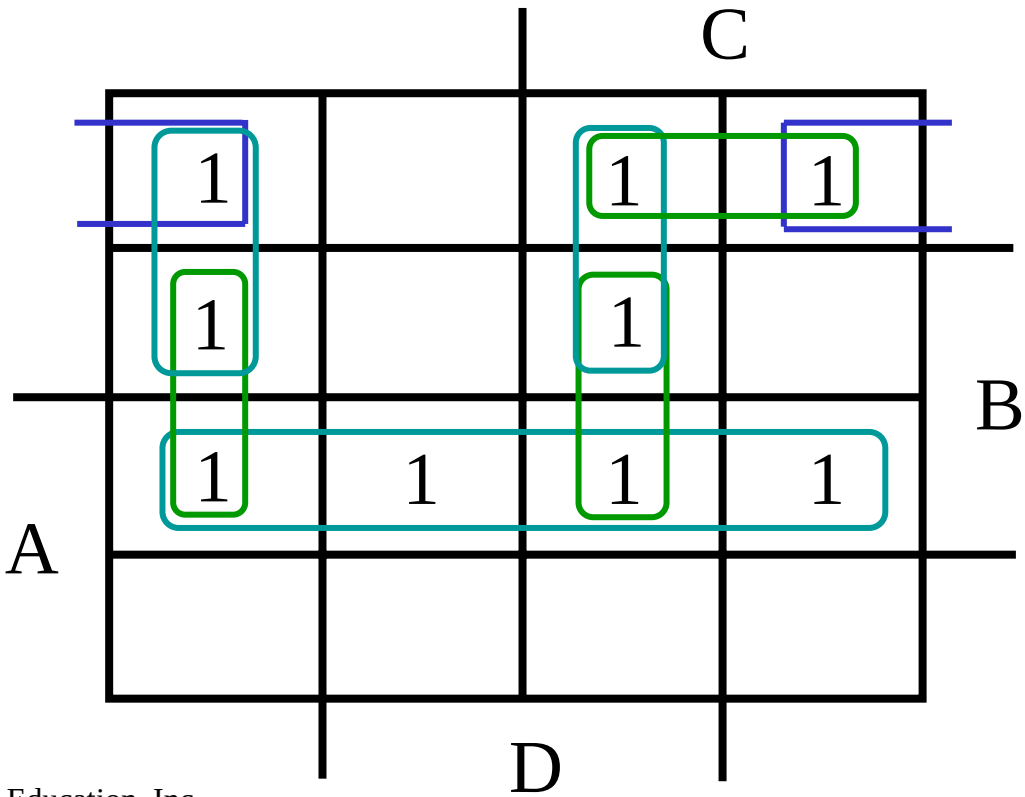


Another Example

- Find all prime implicants for:

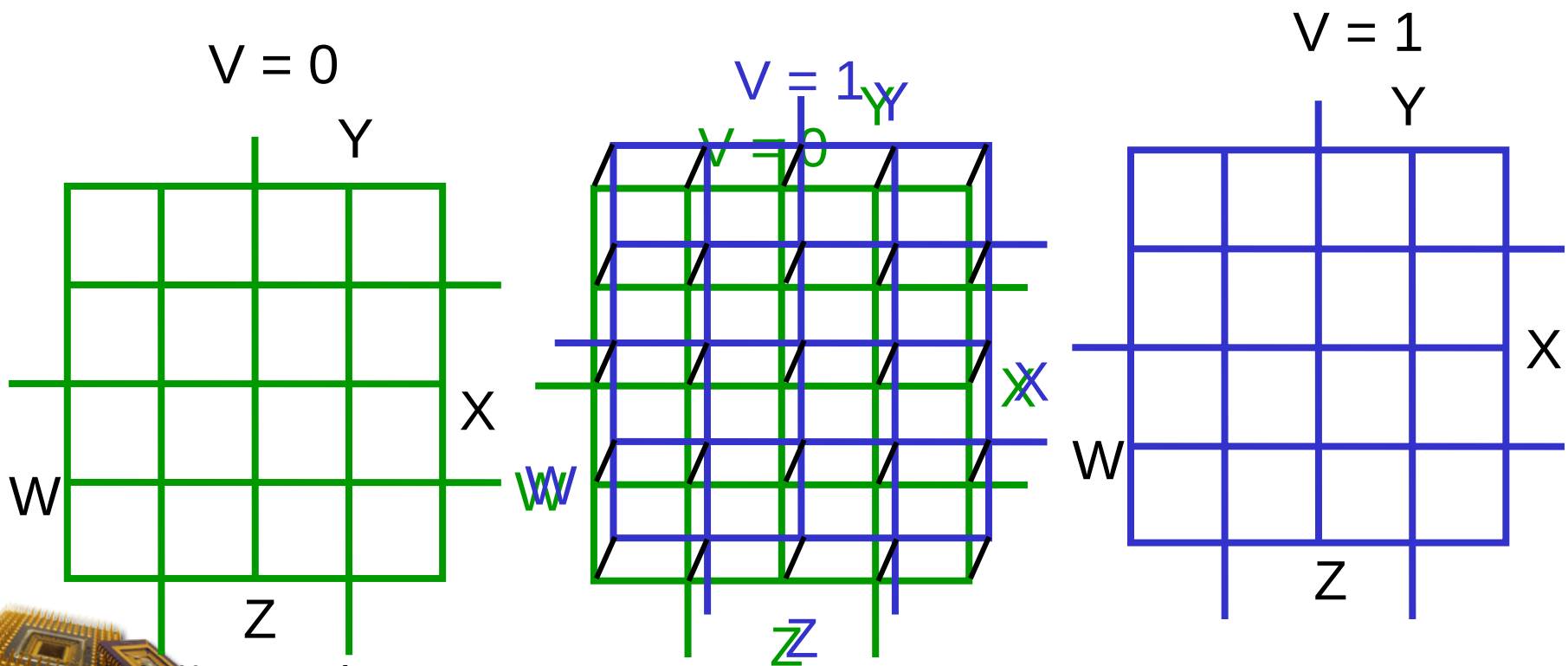
$$G(A, B, C, D) = \Sigma_m(0,2,3,4,7,12,13,14,15)$$

  - Hint: There are seven prime implicants!



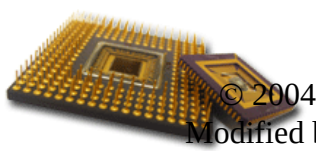
Five Variable or More K-Maps

- For five variable problems, we use *two adjacent K-maps*. It becomes harder to visualize adjacent minterms for selecting PIs. You can extend the problem to six variables by using four K-Maps.



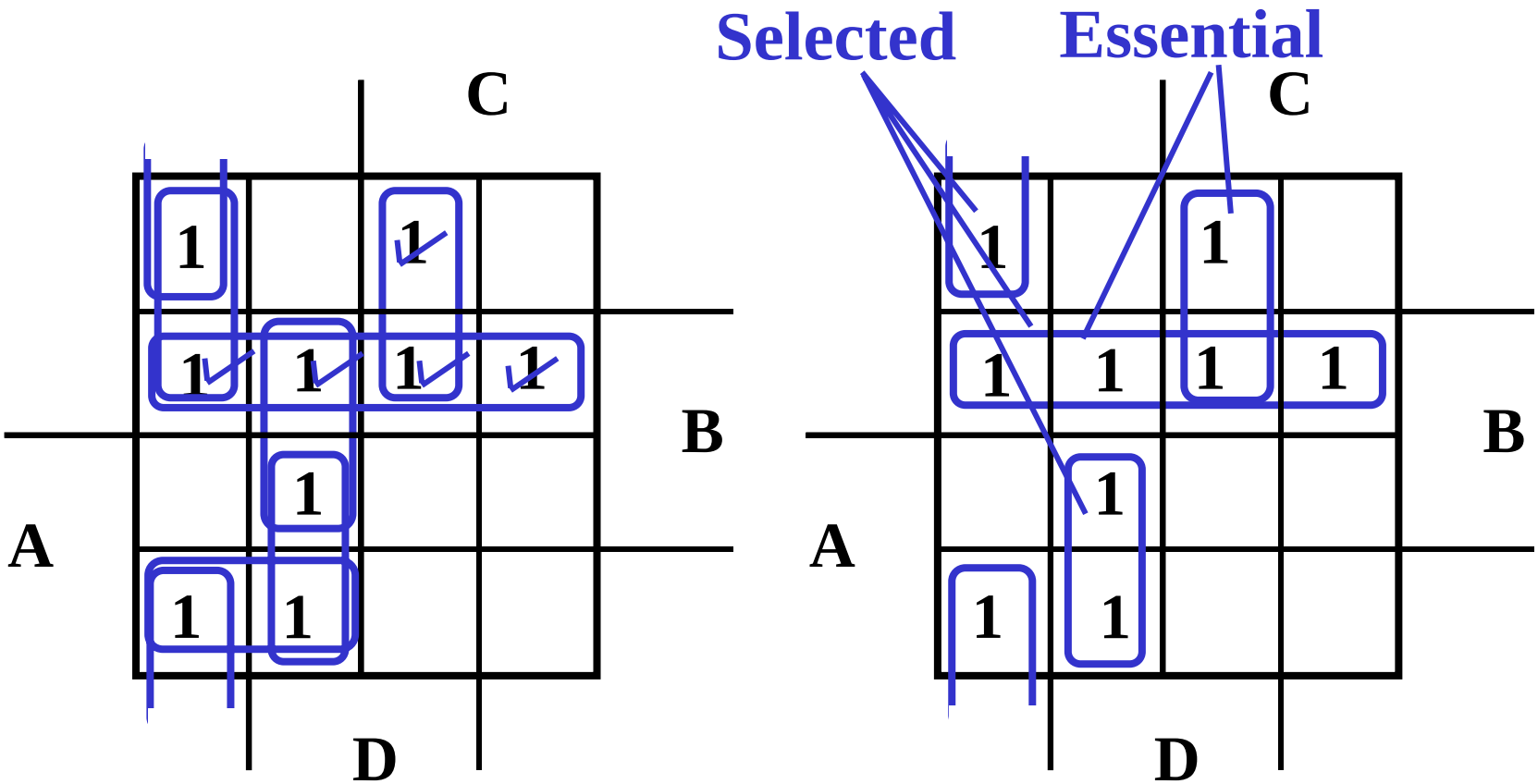
Prime Implicants Selection Rule

- Find all prime implicants.
- Include all essential prime implicants in the solution
- Select a minimum cost set of non-essential prime implicants to cover all minterms not yet covered:
  - Minimize the overlap among prime implicants as much as possible.
  - Make sure that each prime implicant selected includes at least one minterm not included in any other prime implicant selected – avoid redundancy.



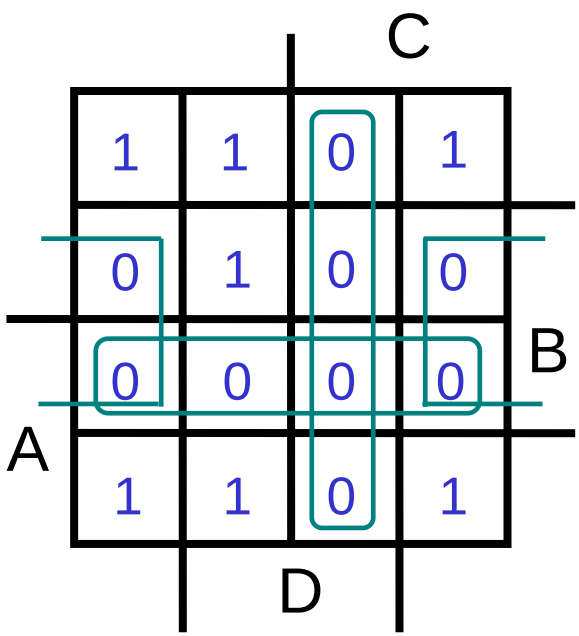
Selection Rule Example

- Simplify F(A, B, C, D) given on the K-map.



Product-of-Sums Simplification

- Concept: $\overline{m_i} = M_i$
- How to derive a POS using SOP derivation:
  - Derive $\overline{F} = m_i + m_j \ldots + m_n \leftrightarrow F = M_i \cdot M_j \ldots \cdot M_n$



$\overline{F} = AB + CD + B\overline{D}$

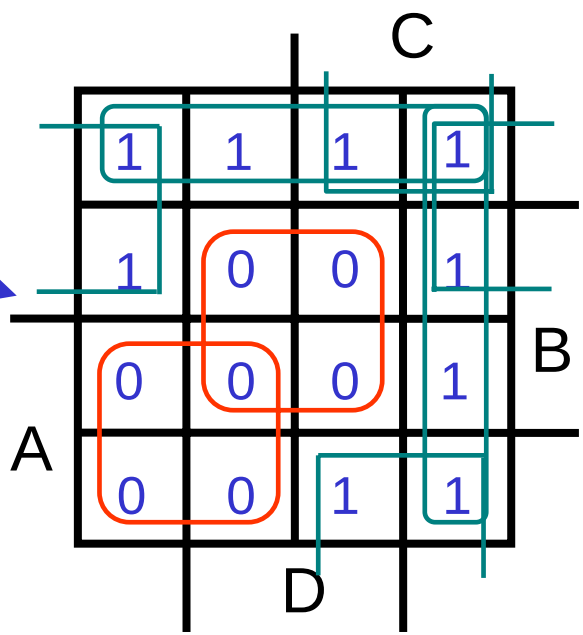
$F = (\overline{A}+\overline{B})(\overline{C}+\overline{D})(\overline{B}+D)$

Product-of-Sums Simplification

- How to derive a simplified SOP and POS from a POS?

$$F = (\overline{A}+C+D)(\overline{A}+C+\overline{D})(A+\overline{B}+\overline{D})(\overline{A}+\overline{B}+\overline{D})$$

$$\overline{F} = A\overline{C}\,\overline{D} + A\overline{C}D + \overline{A}BD + ABD$$



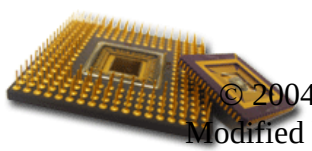
$$F = \overline{A}\,\overline{B} + C\overline{D} + \overline{A}\,\overline{D} + \overline{B}C$$

$$\overline{F} = A\overline{C} + BD$$

$$F = (\overline{A}+C)(\overline{B}+\overline{D})$$

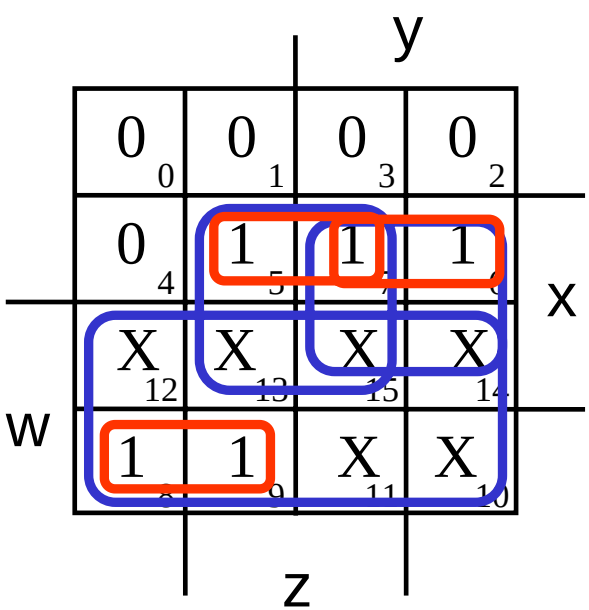
Don't Cares in K-Maps

- Sometimes a function table or map contains entries for which it is known:
  - the input values for the minterm will never occur, or
  - The output value for the minterm is not used
- In these cases, the output value need not be defined
  - Incompletely specified functions – functions that have unspecified outputs (it can be 1 or 0)for some input combinations.
- Instead, the output value is defined as a “don't care”
- By placing “don't cares” (an “x” entry) in the function table or map, the cost of the logic circuit may be lowered.
- Example 1: A logic function having the binary codes for the BCD digits as its inputs. Only the codes for 0 through 9 are used. The six codes, 1010 through 1111 never occur, so the output values for these codes are “x” to represent “don’t cares.”



Example: BCD "5 or More"

- The map below gives a function F1(w,x,y,z) which is defined as "5 or more" over BCD inputs. With the don't cares used for the 6 non-BCD combinations:



$F1(w,x,y,z) = \overline{w}xz + \overline{w}xy + w\overline{x}\overline{y}$ G =12

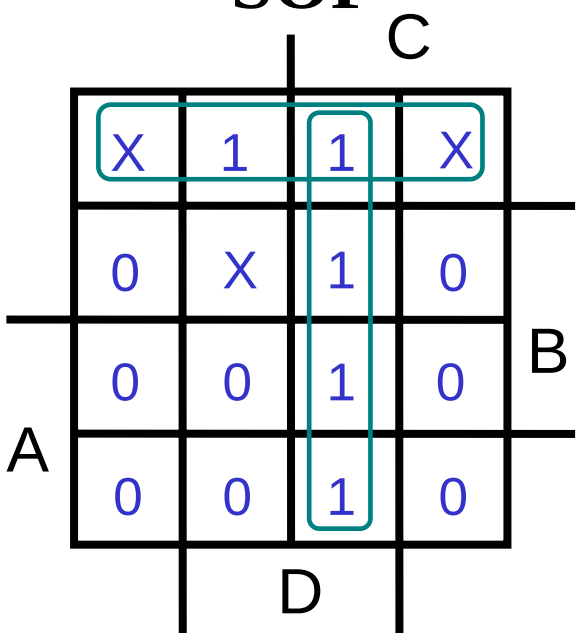
- Consider the input characteristics

$F2(w,x,y,z) = w + xz + xy$ G = 7

- This is much lower in cost than F2 where the "don't cares" were treated as "0s."

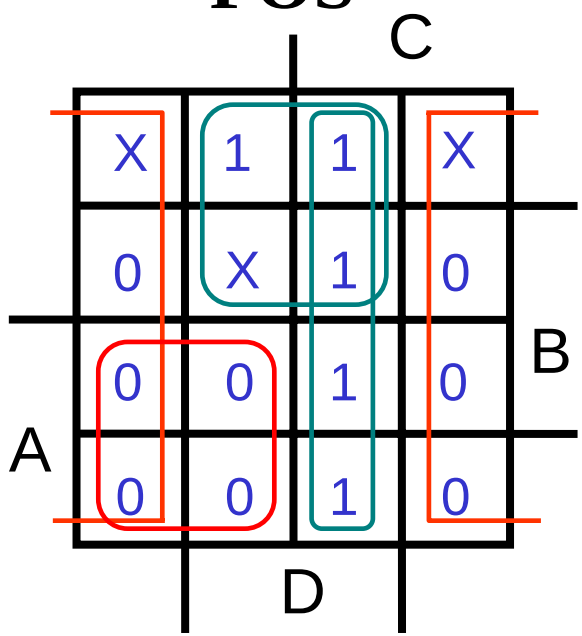
Don't Care – Another Example

SOP



$$F = CD + \overline{A}\overline{B}$$

POS



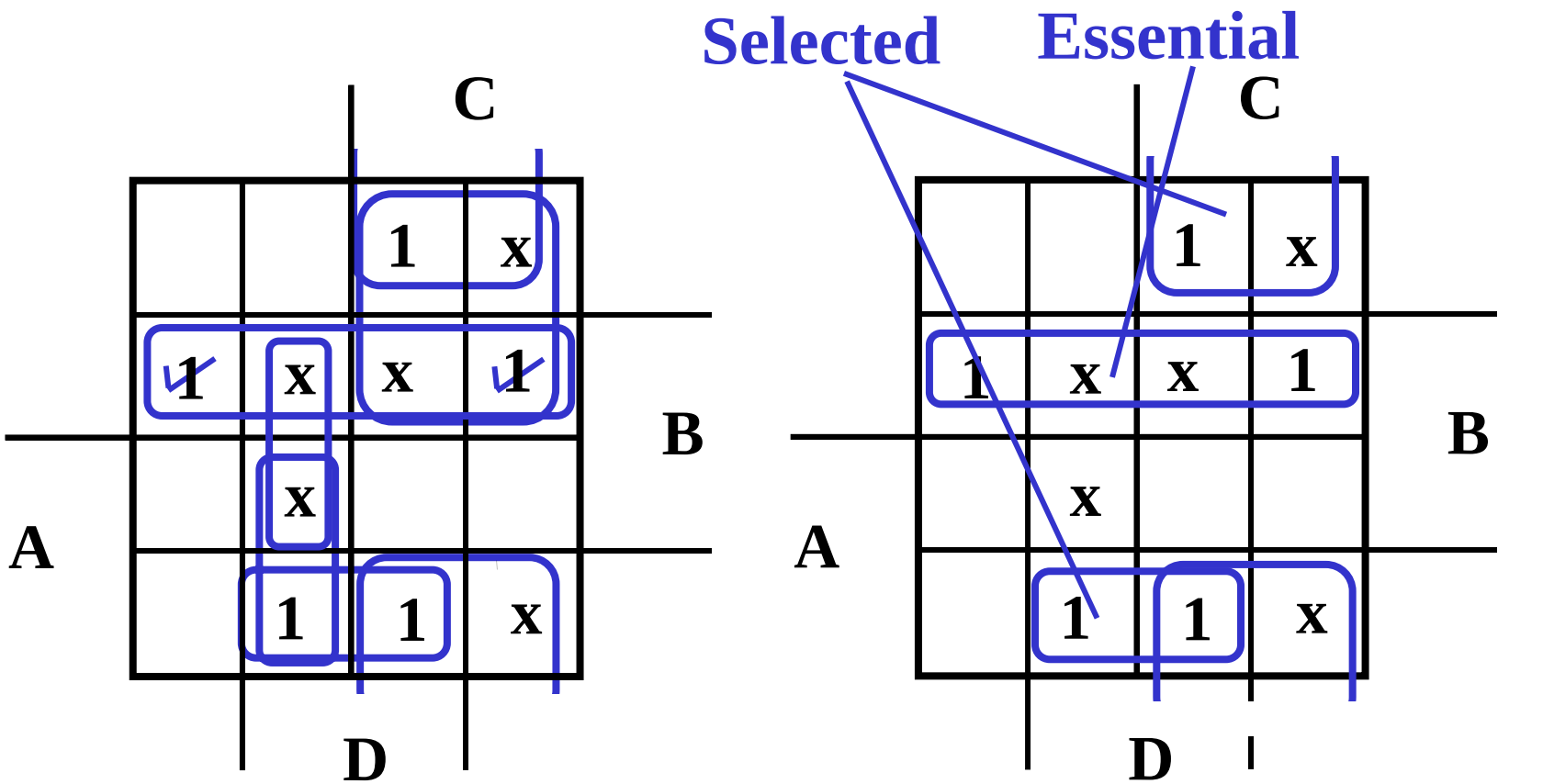
$$\overline{F} = \overline{D} + A\overline{C}$$

$$F = D\,(\overline{A} + C)$$

$$F = CD + \overline{A}D$$

Selection Rule Example with Don't Cares

- Simplify F(A, B, C, D) given on the K-map.

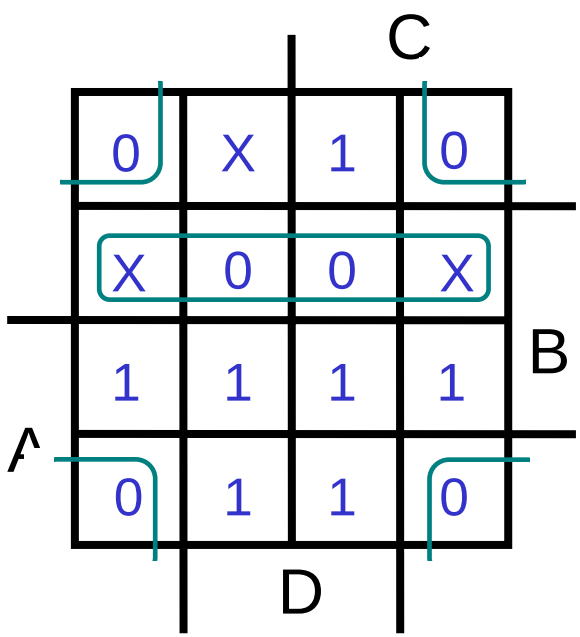


Example: Product of Sums

- Find the optimum POS solution:

$$\mathbf{F(A,B,C,D) = \Sigma_m(3,9,11,12,13,14,15) + \Sigma d\,(1,4,6)}$$

  - Hint: Use $\overline{F}$ and complement it to get the result.



$$\overline{F} = \overline{A}B + \overline{B}\overline{D}$$

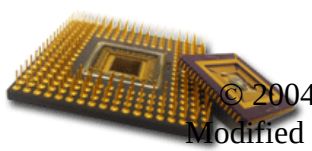
$$F = (A + \overline{B})\,(B + D)$$

Multiple-Level Optimization

- Multiple-level circuits - circuits that are not two-level (with or without input and/or output inverters)
- Multiple-level circuits can have reduced gate input cost compared to two-level (SOP and POS) circuits
- Multiple-level optimization is performed by applying transformations to circuits represented by equations while evaluating cost

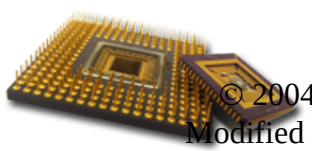
$F = ABC + ABD + AB\overline{C}\,\overline{D}$ $\qquad$ $F = AB(C + D + \overline{C}\,\overline{D})$

2 levels, 4 gates, 13 gate inputs $\qquad$ 3 levels, 3 gates, 8 gate inputs



Transformations

- Factoring - finding a factored form from SOP or POS expression
- Decomposition - expression of a function as a set of new functions
- Substitution of G into F - expression function F as a function of G and some or all of its original variables
- Elimination - Inverse of substitution
- Extraction - decomposition applied to multiple functions simultaneously



Transformation Examples (1/6)

- Algebraic Factoring

$F = \bar{A}\,\bar{C}\,\bar{D} + \bar{A}B\,\bar{C} + ABC + AC\,\bar{D}$ G = 16

  - Factoring:

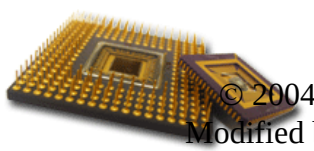
$F = \bar{A}\,(\bar{C}\,\bar{D} + B\,\bar{C}) + A\,(BC + C\,\bar{D})$ G = 18

  - Factoring again:

$F = \bar{A}\,\bar{C}(B + \bar{D}) + AC\,(B + \bar{D})$ G = 12

  - Factoring again:

$F = (\bar{A}\,\bar{C} + AC)\,(B + \bar{D})$ G = 10

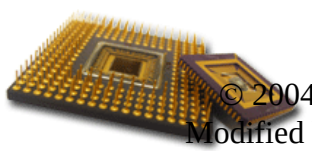


Transformation Examples (2/6)

- Decomposition
  - The terms $B + \overline{D}$ and $\overline{A}\,\overline{C} + AC$ can be defined as new functions E and H respectively, decomposing F:

  $F = E\,H$, $E = B + \overline{D}$, and $H = \overline{A}\,\overline{C} + AC$ G = 10

- This series of transformations has reduced G from 16 to 10, a substantial savings. The resulting circuit has three levels plus input inverters.



Transformation Examples (3/6)

- Substitution of E into F
  - Returning to F just before the final factoring step:

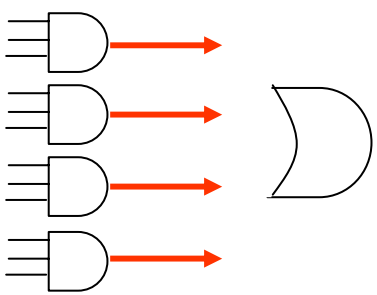
$$F = \bar{A}\,\bar{C}(B + \bar{D}) + AC(B + \bar{D}) \qquad G = 12$$

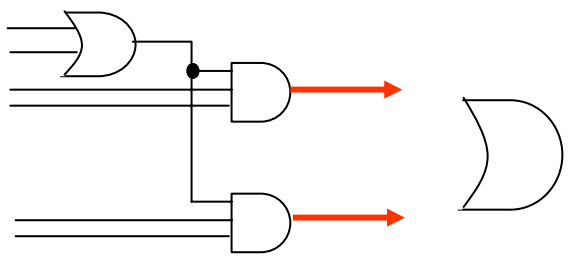
  - Defining $E = B + \bar{D}$, and substituting in F:

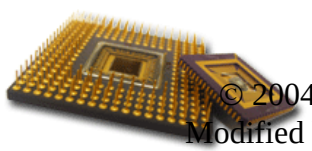
$$F = \bar{A}\,\bar{C}\,E + ACE \qquad G = 10$$

  - This substitution has resulted in the same cost as the decomposition

$F = \bar{A}\bar{C}\bar{D} + \bar{A}B\bar{C} + ABC + AC\bar{D}$ $\qquad$ $F = \bar{A}\bar{C}E + ACE,\ E = B + \bar{D}$







Transformation Examples (4/6)

- Elimination
  - Beginning with a new set of functions:

$X = B + C$

$Y = A + B$

$Z = \overline{A}X + C\,Y$ $\qquad$ $G = 10$

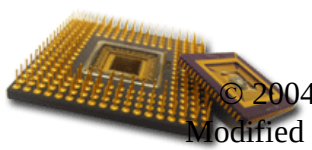
  - Eliminating X and Y from Z:

$Z = \overline{A}(B + C) + C\,(A + B)$ $\qquad$ $G = 10$

  - "Flattening" (Converting to SOP expression):

$Z = \overline{A}\,B + \overline{A}\,C + AC + BC$ $\qquad$ $G = 12$

  - This has increased the cost, but has provided an new SOP expression for two-level optimization.

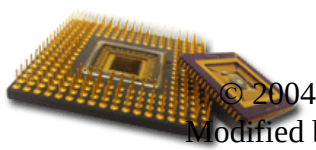


Transformation Examples (5/6)

- Two-level Optimization
  - The result of 2-level optimization is:

$Z = \overline{A}\,B + C$ G = 4

- This example illustrates that:
  - Optimization can begin with any set of equations, not just with minterms or a truth table
  - Increasing gate input count G temporarily during a series of transformations can result in a final solution with a smaller G



Transformation Examples (6/6)

- Extraction
  - Beginning with two functions:

$$E = \overline{A}\,\overline{B}\,\overline{D} + \overline{A}BD$$

$$H = \overline{B}\,C\,\overline{D} + BCD \qquad G = 16$$

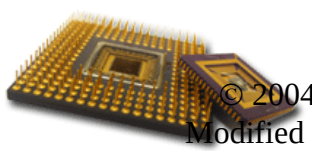
  - Finding a common factor and defining it as a function:

$$F = \overline{B}\overline{D} + BD$$

  - We perform extraction by expressing E and H as the three functions:

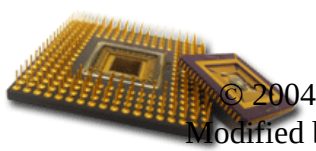
$$F = \overline{B}\overline{D} + BD,\ E = \overline{A}F,\ H = CF \qquad G = 10$$

  - The reduced cost G results from the sharing of logic between the two output functions



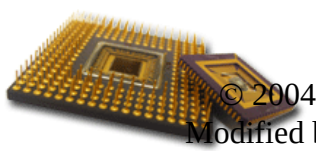
Universal

- A set of gates is said to be *universal* if any Boolean function can be composed of the elements of this set.
  - {AND, OR, NOT} – a universal set.
- *Universal gate* - a gate type that can implement any Boolean function.
  - NAND, NOR,…



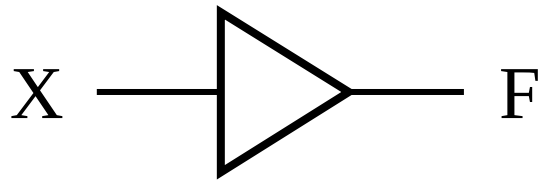
Other Gate Types

- Why?
  - Implementation feasibility and low cost
  - Power in implementing Boolean functions
  - Convenient conceptual representation
- Gate classifications
  - Primitive gate - a gate that can be described using a single primitive operation type (AND or OR) plus an optional inversion(s).
  - Complex gate - a gate that requires more than one primitive operation type for its description (XOR, XNOR).
- Primitive gates will be covered first.

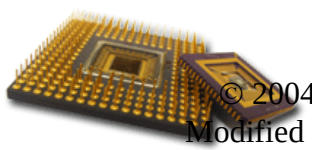


Buffer

- A buffer is a gate with the function F = X:

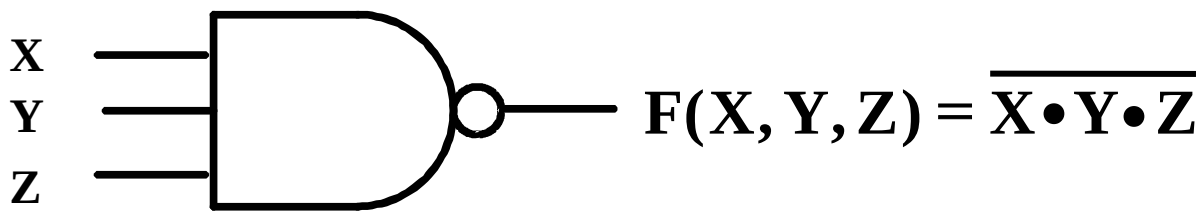


- In terms of Boolean function, a buffer is the same as a connection!
- So why use it?
  - A buffer is an electronic amplifier used to improve circuit voltage levels and increase the speed of circuit operation.



NAND Gate (1/3)

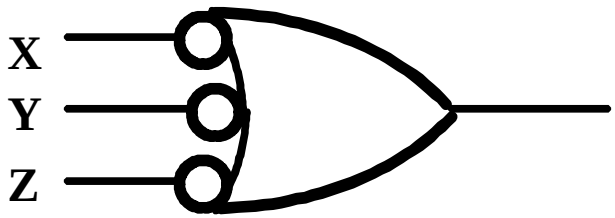
- The basic NAND gate has the following symbol, illustrated for three inputs:
  - AND-Invert (NAND)



- NAND represents NOT AND, i. e., the AND function with a NOT applied. The symbol shown is an AND-Invert. The small circle ("bubble") represents the invert function.

NAND Gates (2/3)

- Applying DeMorgan's Law gives Invert-OR (NAND)

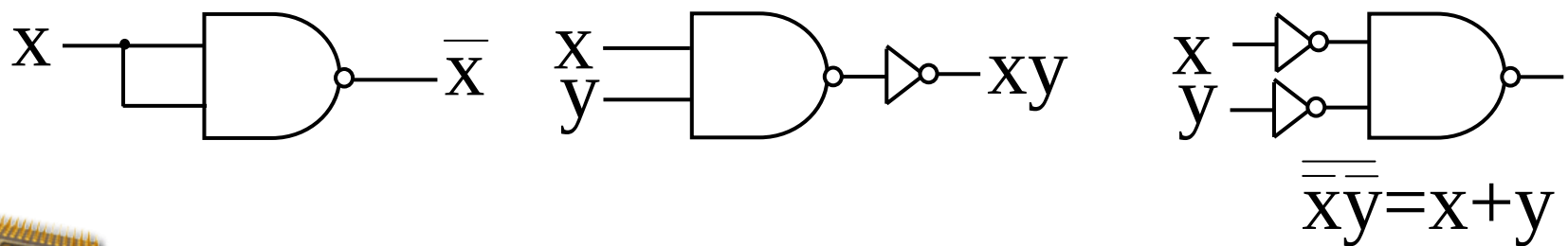


$$F(X, Y, Z) = \overline{X} + \overline{Y} + \overline{Z}$$

- This NAND symbol is called Invert-OR, since inputs are inverted and then ORed together.
- AND-Invert and Invert-OR both represent the NAND gate. Having both makes visualization of circuit function easier.
- A NAND gate with one input degenerates to an inverter.

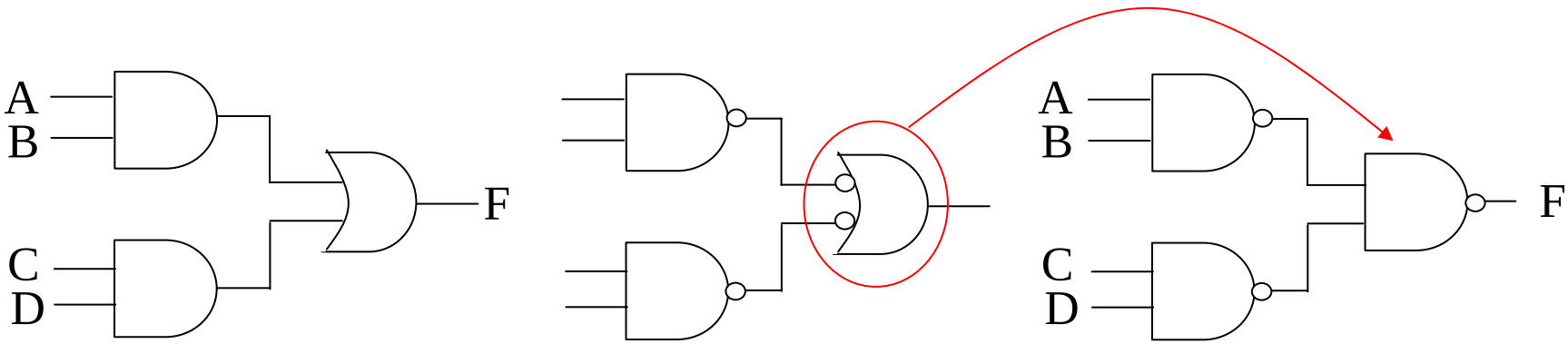
NAND Gates (3/3)

- The NAND gate is the natural implementation for the simplest and fastest electronic circuits
- The NAND gate is a universal gate as shown in the following.
- NAND usually does not have an operation symbol defined since
  - the NAND operation is not associative, and
  - we have difficulty dealing with non-associative mathematics!



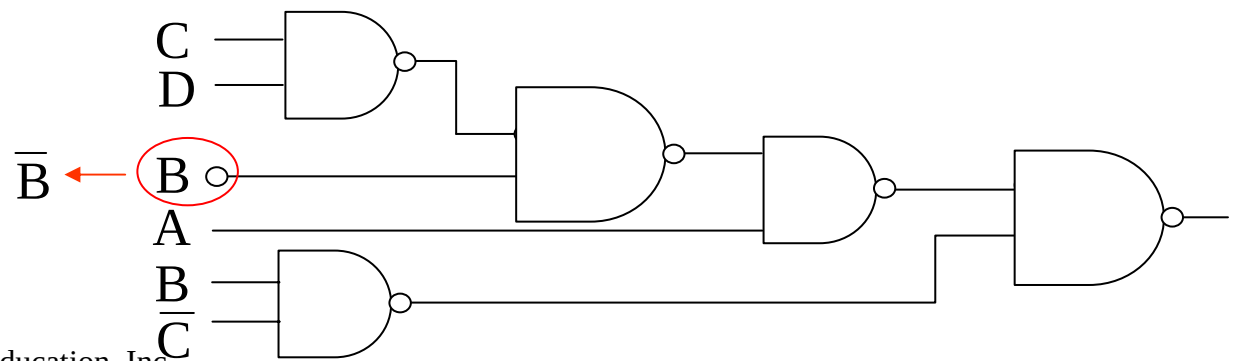
Two-level NAND Gates (1/2)

- A function of sum-of-products form is easy to convert to NAND gates.
  - $F = AB + CD \rightarrow F = \overline{\overline{AB + CD}} \rightarrow F = \overline{\overline{AB} \cdot \overline{CD}}$



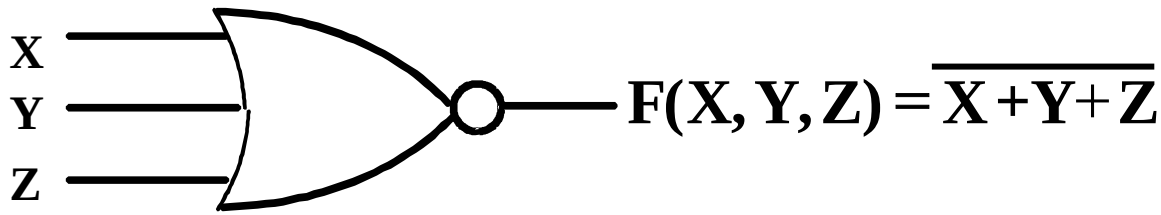
Multilevel NAND Gates (2/2)

- The procedures to convert to multilevel NAND gates circuits:
  - Convert all AND gates to NAND gates with AND-NOT graphic symbols.
  - Convert all OR gates to NAND gates with NOT-OR graphic symbols.
  - Complement the input literal if the signal has single bubble, and insert a NOT gate for the output signal with single bubble.



NOR Gate (1/4)

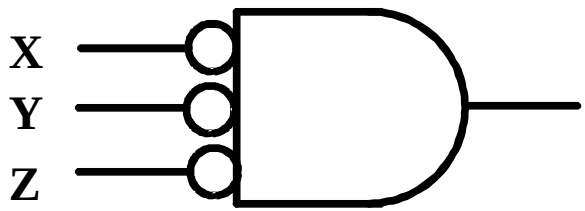
- The basic NOR gate has the following symbol, illustrated for three inputs:
  - OR-Invert (NOR)



- NOR represents NOT-OR, i. e., the OR function with a NOT applied. The symbol shown is an OR-Invert. The small circle ("bubble") represents the invert function.

NOR Gate (2/4)

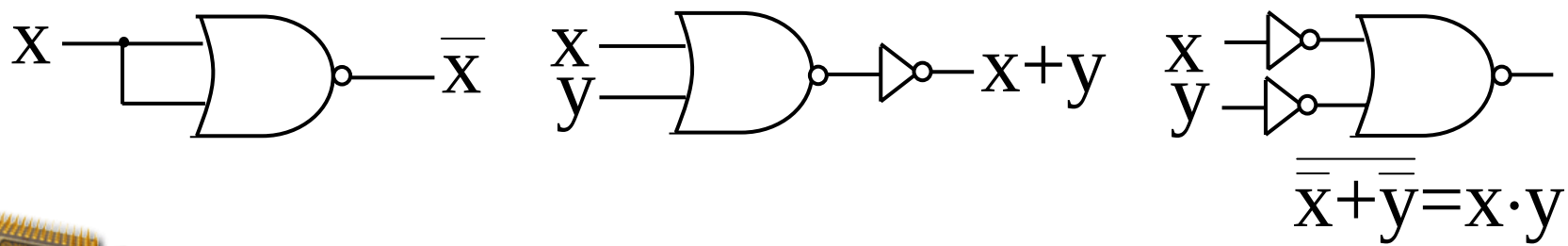
- Applying DeMorgan's Law gives Invert-AND (NOR)



- This NOR symbol is called Invert-AND, since inputs are inverted and then ANDed together.
- OR-Invert and Invert-AND both represent the NOR gate. Having both makes visualization of circuit function easier.
- A NOR gate with one input degenerates to an inverter.

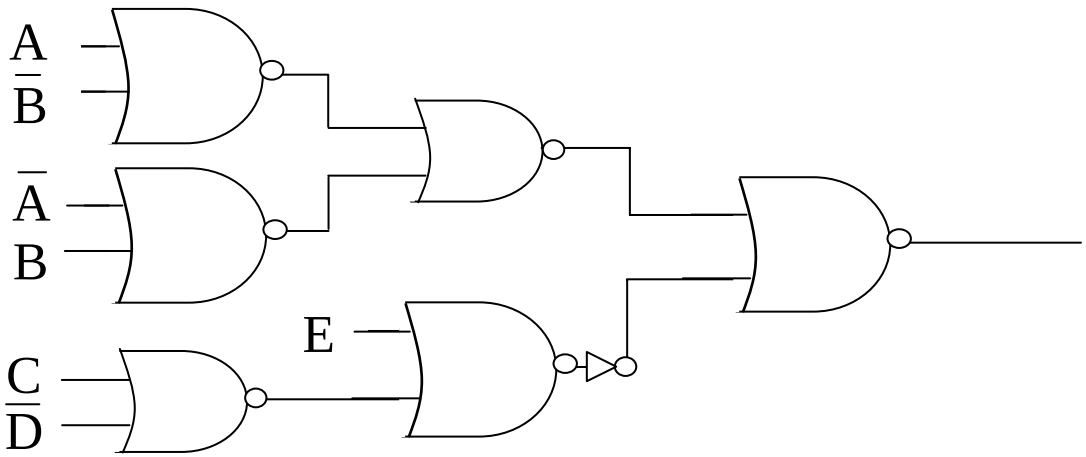
NOR Gate (3/4)

- The NOR gate is another natural implementation for the simplest and fastest electronic circuits
- The NOR gate is a universal gate
- NOR usually does not have a defined operation symbol since
  - the NOR operation is not associative, and
  - we have difficulty dealing with non-associative mathematics!



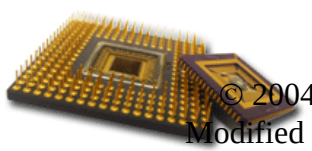
NOR Gate (4/4)

- **Example:**



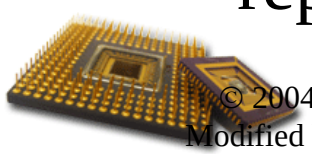
XOR/XNOR (1/4)

- The *eXclusive OR* (*XOR*) function is an important Boolean function used extensively in logic circuits.
- The XOR function may be;
  - implemented directly as an electronic circuit (truly a gate) or
  - implemented by interconnecting other gate types (used as a convenient representation)
- The *eXclusive NOR* function is the complement of the XOR function
- By our definition, XOR and XNOR gates are complex gates.



XOR/XNOR (2/4)

- Uses for the XOR and XNORs gate include:
  - Adders/subtractors/multipliers
  - Counters/incrementers/decrementers
  - Parity generators/checkers
- Definitions
  - The XOR function is: $X \oplus Y = X\overline{Y} + \overline{X}Y$
  - The eXclusive NOR (XNOR) function, otherwise known as *equivalence* is: $\overline{X \oplus Y} = XY + \overline{X}\,\overline{Y}$
- Strictly speaking, XOR and XNOR gates do no exist for more than two inputs. Instead, they are replaced by odd and even functions.



XOR/XNOR (3/4)

- Operator Rules: XOR

X	Y	X⊕Y
0	0	0
0	1	1
1	0	1
1	1	0

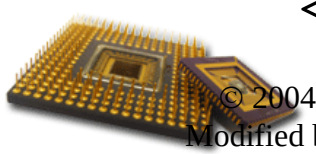
XNOR

X	Y	$\overline{(X \oplus Y)}$ or X≡Y
0	0	1
0	1	0
1	0	0
1	1	1

- The XOR function means:

  X OR Y, but NOT BOTH

- Why is the XNOR function also known as the *equivalence* function, denoted by the operator ≡?
  <XNOR (X, X)=1>



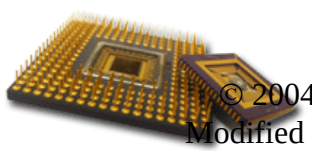
XOR/XNOR (4/4)

- The XOR function can be extended to 3 or more variables. For more than 2 variables, it is called an *odd function* or *modulo 2 sum* (*Mod 2 sum*), not an XOR:

$$X \oplus Y \oplus Z = \overline{X}\,\overline{Y}\,Z + \overline{X}\,Y\,\overline{Z} + X\,\overline{Y}\,\overline{Z} + X\,Y\,Z$$

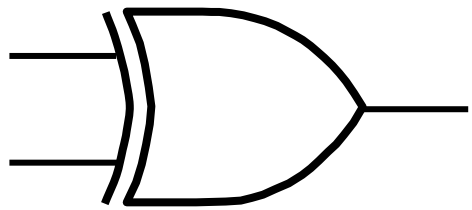
- The complement of the odd function is the even function.
- The XOR identities:

$$X \oplus 0 = X \qquad X \oplus 1 = \overline{X}$$
$$X \oplus X = 0 \qquad X \oplus \overline{X} = 1$$
$$X \oplus Y = Y \oplus X$$
$$(X \oplus Y) \oplus Z = X \oplus (Y \oplus Z) = X \oplus Y \oplus Z$$

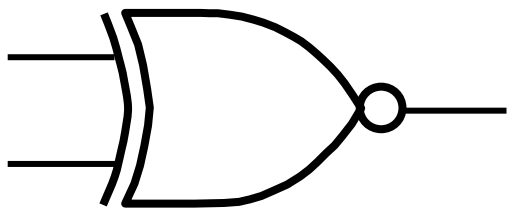


Symbols For XOR and XNOR

- XOR symbol:



- XNOR symbol:

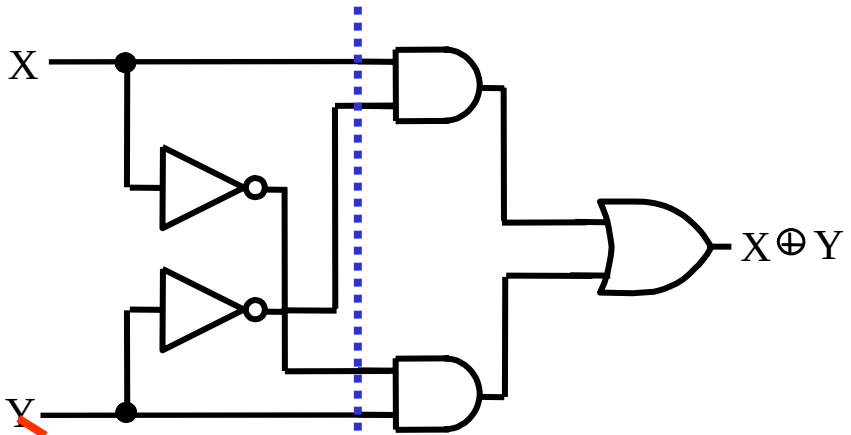


- Symbols exist only for two inputs

XOR Implementations

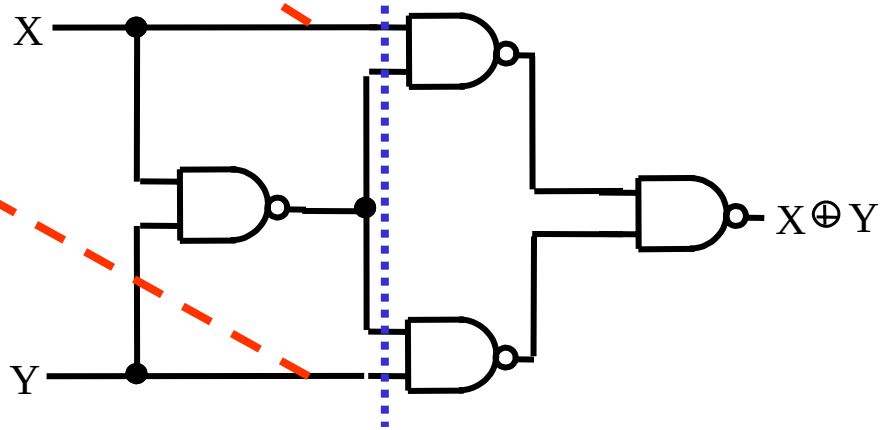
- The simple SOP implementation uses the following structure:

$$x\bar{y} = x\bar{y} + x\bar{x}$$
$$= x(\bar{x}+\bar{y})$$
$$= x\overline{x\bar{y}}$$



- A NAND only implementation is:

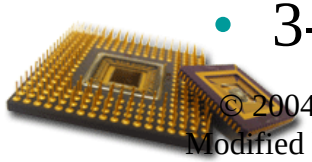
$$y\bar{x} = y\bar{x} + y\bar{y}$$
$$= y(\bar{x}+\bar{y})$$
$$= y\overline{x\bar{y}}$$



Odd and Even Functions

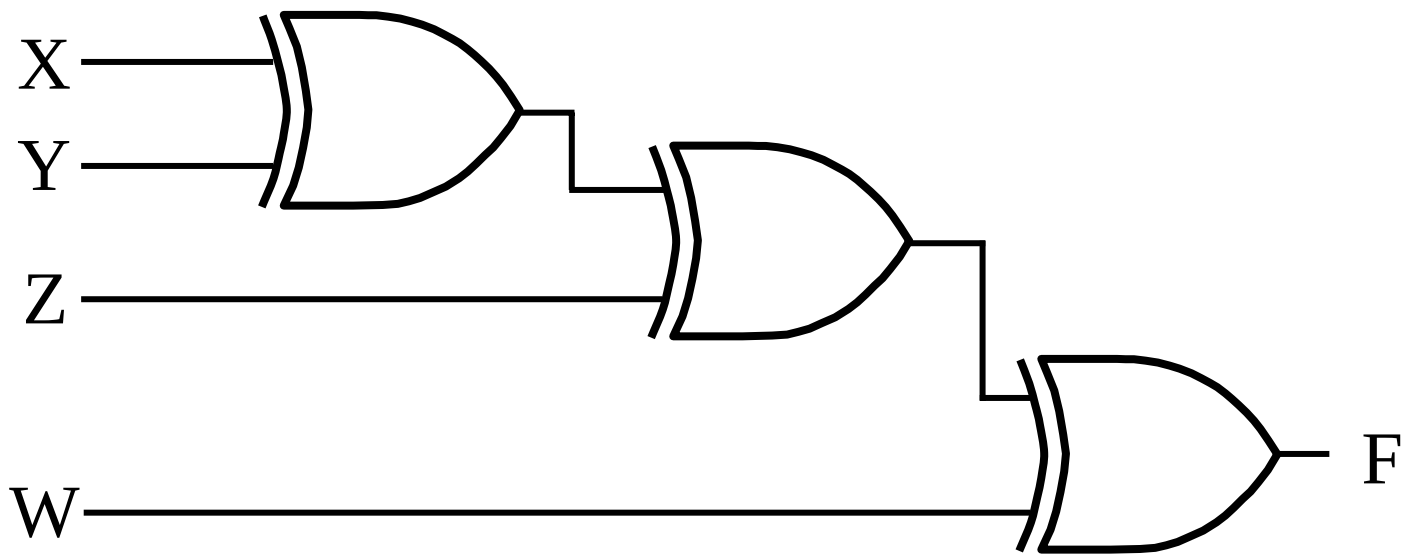
- The odd and even functions on a K-map form “checkerboard” patterns.
- The 1s of an odd function correspond to minterms having an index with an odd number of 1s.
- The 1s of an even function correspond to minterms having an index with an even number of 1s.
- Implementation of odd and even functions for greater than 4 variables as a two-level circuit is difficult, so we use “trees” made up of :
  - 2-input XOR or XNORs
  - 3- or 4-input odd or even functions

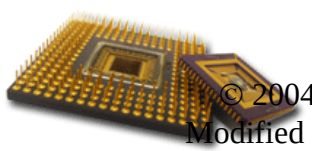
	00	01	11	1O
00	E	O	E	O
01	O	E	O	E
11	E	O	E	O
10	O	E	O	E



Example: Odd Function Implementation

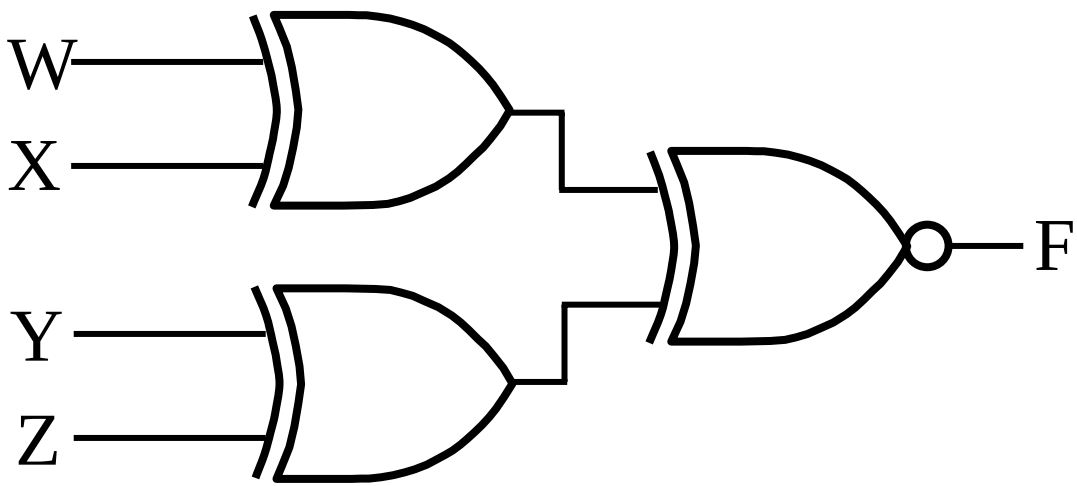
- Design a 4-input odd function $F = X \oplus Y \oplus Z \oplus \mathbf{W}$ with 2-input XOR gates
- Factoring, $F = (X \oplus Y) \oplus Z \oplus \mathbf{W}$
- The circuit:





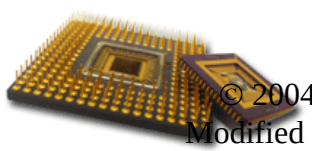
Example: Even Function Implementation

- Design a 4-input even function $F = \overline{W \oplus X \oplus Y \oplus Z}$ with 2-input XOR and XNOR gates
- Factoring, $F = \overline{(W \oplus X) \oplus (Y \oplus Z)}$
- The circuit:



Hi-Impedance Outputs (1/2)

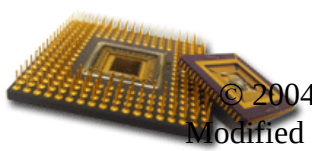
- Logic gates introduced thus far
  - have 1 and 0 output values,
  - cannot have their outputs connected together, and
  - transmit signals on connections in only one direction.
- Three-state logic adds a third logic value, Hi-Impedance (Hi-Z), giving three states: 0, 1, and Hi-Z on the outputs.
- The presence of a Hi-Z state makes a gate output as described above behave quite differently:
  - "1 and 0" become "1, 0, and Hi-Z"
  - "cannot" becomes "can," and
  - "only one" becomes "two"



Hi-Impedance Outputs (2/2)

- What is a Hi-Z value?
  - The Hi-Z value behaves as an open circuit
  - This means that, looking back into the circuit, the output appears to be disconnected.
  - It is as if a switch between the internal circuitry and the output has been opened.
- Hi-Z may appear on the output of any gate, but we restrict gates to:
  - a 3-state buffer, or
  - a transmission gate,

  each of which has one data input and one control input.

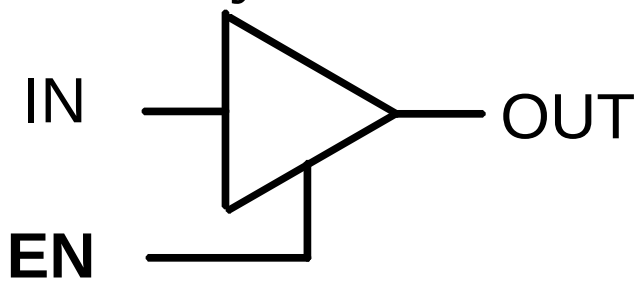


3-State Buffer

- For the symbol and truth table, IN is the data input, and EN, the control input.
- For EN = 0, regardless of the value on IN (denoted by X), the output value is Hi-Z.
- For EN = 1, the output value follows the input value.
- Variations:
  - Data input, IN, can be inverted
  - Control input, EN, can be inverted

by addition of "bubbles" to signals.

Symbol



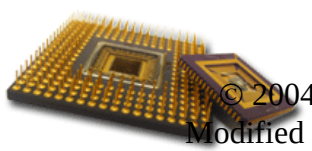
Truth Table

EN	IN	OUT
0	X	Hi-Z
1	0	0
1	1	1

Resolving 3-State Values on a Connection

- Connection of two 3-state buffer outputs, B1 and B0, to a wire, OUT
- Assumption: Buffer data inputs can take on any combination of values 0 and 1
- Resulting Rule: At least one buffer output value must be Hi-Z. Why?
- How many valid buffer output combinations exist?
- What is the rule for *n* 3-state buffers connected to wire, OUT?
- How many valid buffer output combinations exist?

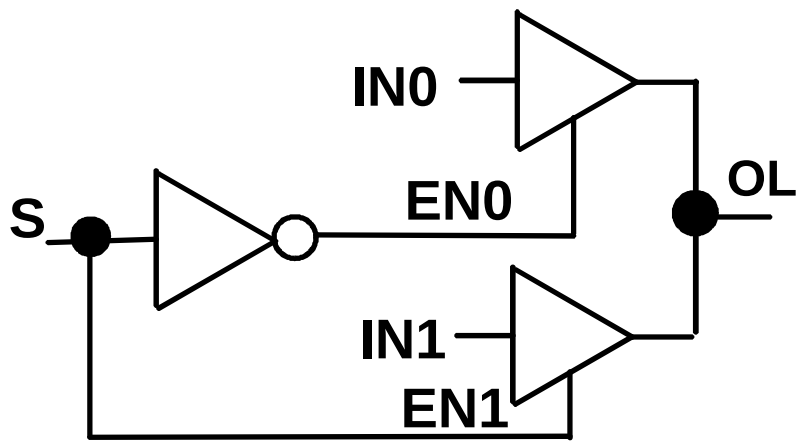
Resolution Table		
B1	B0	OUT
0	Hi-Z	0
1	Hi-Z	1
Hi-Z	0	0
Hi-Z	1	1
Hi-Z	Hi-Z	Hi-Z



3-State Logic Circuit

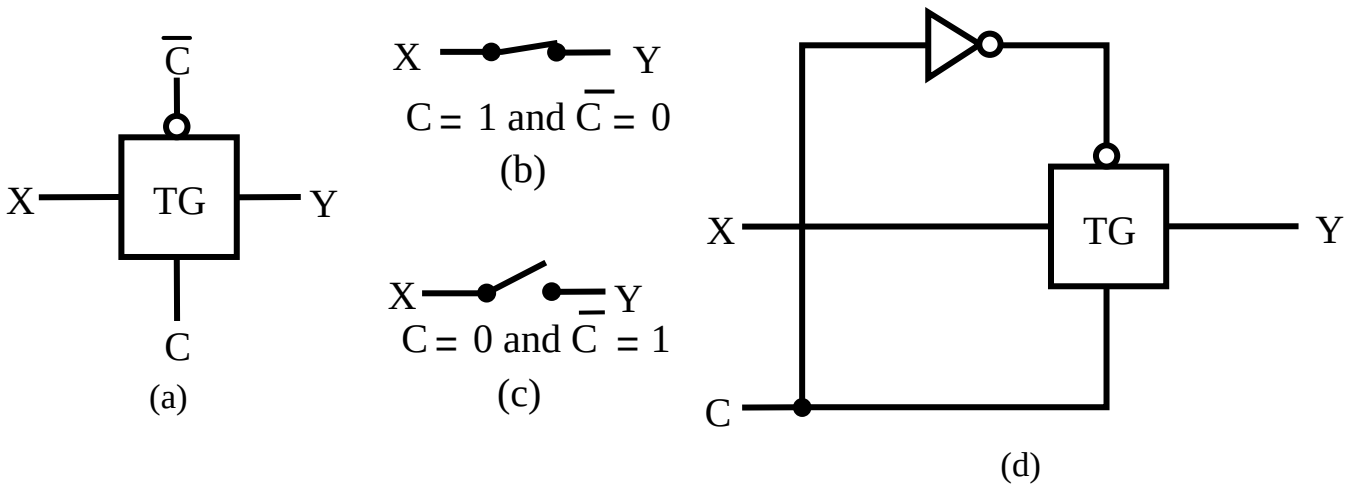
- Data Selection Function: If s = 0, OL = IN0, else OL = IN1
- Performing data selection with 3-state buffers:
- Since EN0 = S and EN1 = S, one of the two buffer outputs is always Hi-Z plus the last row of the table never occurs.

EN1	EN0	IN1	IN0	OL
0	0	X	X	Hi-Z
(S) 0	($\overline{S}$) 1	X	0	0
0	1	X	1	1
1	0	0	X	0
1	0	1	X	1
1	1	0	0	0
1	1	1	1	1
1	1	0	1	
1	1	1	0	



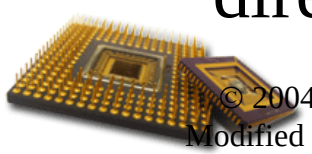
Transmission Gates (1/2)

- The transmission gate is one of the designs for an electronic switch for connecting and disconnecting two points in a circuit:



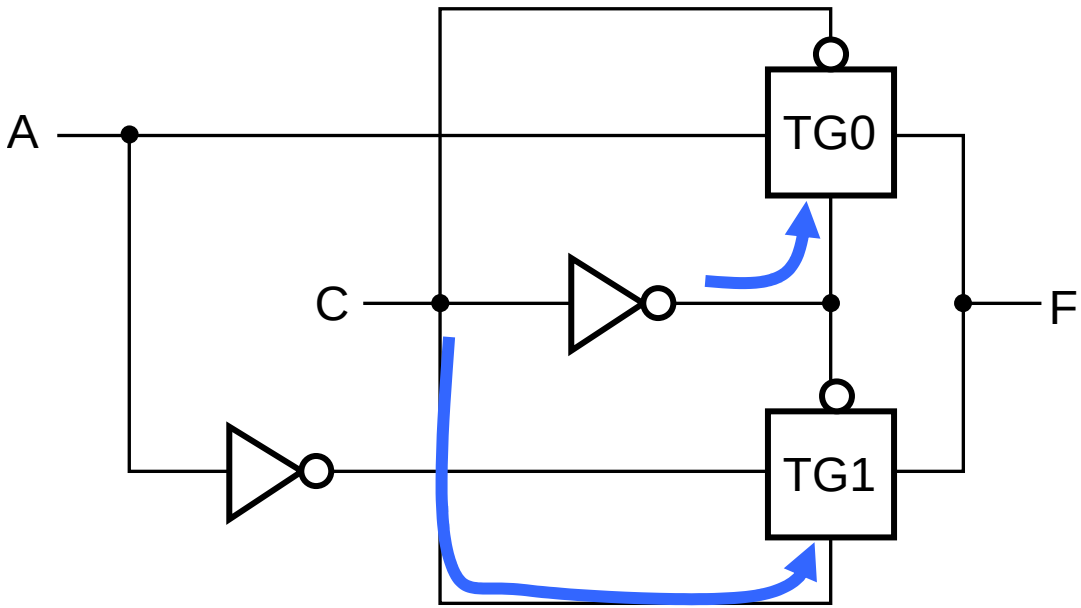
Transmission Gates (2/2)

- In many cases, X can be regarded as a data input and Y as an output. C and $\overline{C}$, with complementary values applied, is a control input.
- With these definitions, the transmission gate, provides a 3-state output:
  - C = 1, Y = X (X = 0 or 1)
  - C = 0, Y = Hi-Z
- Care must be taken when using the TG in design, however, since X and Y as input and output are interchangeable, and signals can pass in both directions.



Circuit Example Using TG

- Exclusive OR **F = A ⊕ C**



(a)

A	C	TG1	TG0	F
0	0	No path	Path	0
0	1	Path	No path	1
1	0	No path	Path	1
1	1	Path	No path	0

(b)

- The basis for the function implementation is TG-controlled paths to the output

Terms of Use

-
- The following terms of use apply in addition to the standard Pearson Education Legal Notice.
- Permission is given to incorporate these materials into classroom presentations and handouts only to instructors adopting Logic and Computer Design Fundamentals as the course text.
- Permission is granted to the instructors adopting the book to post these materials on a protected website or protected ftp site in original or modified form. All other website or ftp postings, including those offering the materials for a fee, are prohibited.
- You may not remove or in any way alter this Terms of Use notice or any trademark, copyright, or other proprietary notice, including the copyright watermark on each slide.
- Return to Title Page