Computer Logic Design Fundamentals

Chapter 2 – Combinational Logic Circuits

Part 2 – Circuit Optimization

Prof. Yueming Wang

ymingwang@zju.edu.cn

College of Computer Science and Technology,
Zhejiang University

Overview

- Part 1 – Gate Circuits and Boolean Equations
  - Binary Logic and Gates
  - Boolean Algebra
  - Standard Forms
- Part 2 – Circuit Optimization
  - Two-Level Optimization
  - Map Manipulation
  - Practical Optimization (Espresso)
  - Multi-Level Circuit Optimization
- Part 3 – Additional Gates and Circuits
  - Other Gate Types
  - Exclusive-OR Operator and Gates
  - High-Impedance Outputs
  - Propagation Delay

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
- Distinct cost criteria we will use:
  - Literal cost (L)
  - Gate input cost (G)
  - Gate input cost with NOTs (GN)

Literal Cost

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

Literal Cost

- **Another Example:**
  - $F = ABCD + \overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$
  - $F = (A + \overline{B})(B + \overline{C})(C + \overline{D})(D + \overline{A})$
  - **Which solution is best?**

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 single literal terms,(G) and
  - optionally, the number of distinct complemented single literals (GN).
- **Example:**
  - $F = BD + A\overline{B}C + A\overline{C}\,\overline{D}$ G = 11, GN = 14
  - $F = BD + A\overline{B}C + A\overline{B}\,\overline{D} + AB\overline{C}$ G = , GN =
  - $F = (A + \overline{B})(A + D)(B + C + \overline{D})(\overline{B} + \overline{C} + D)$ G = , GN =
  - Which solution is best?

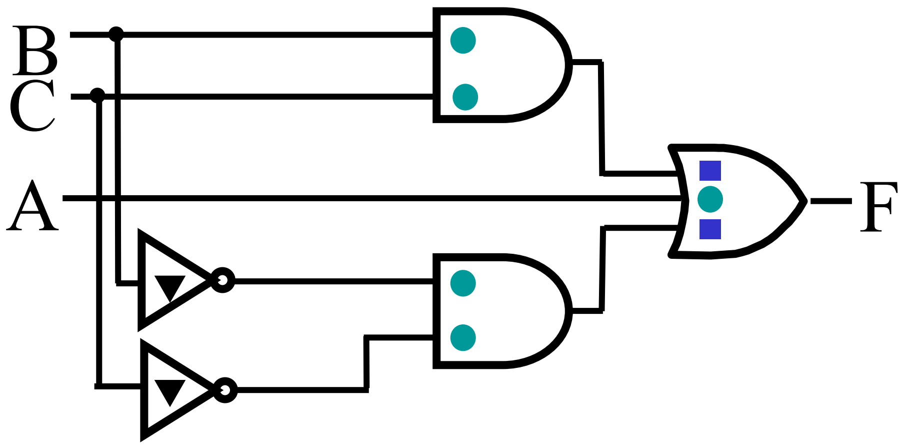
Cost Criteria (continued)

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

$GN = G + 2 = 9$

$L = 5$

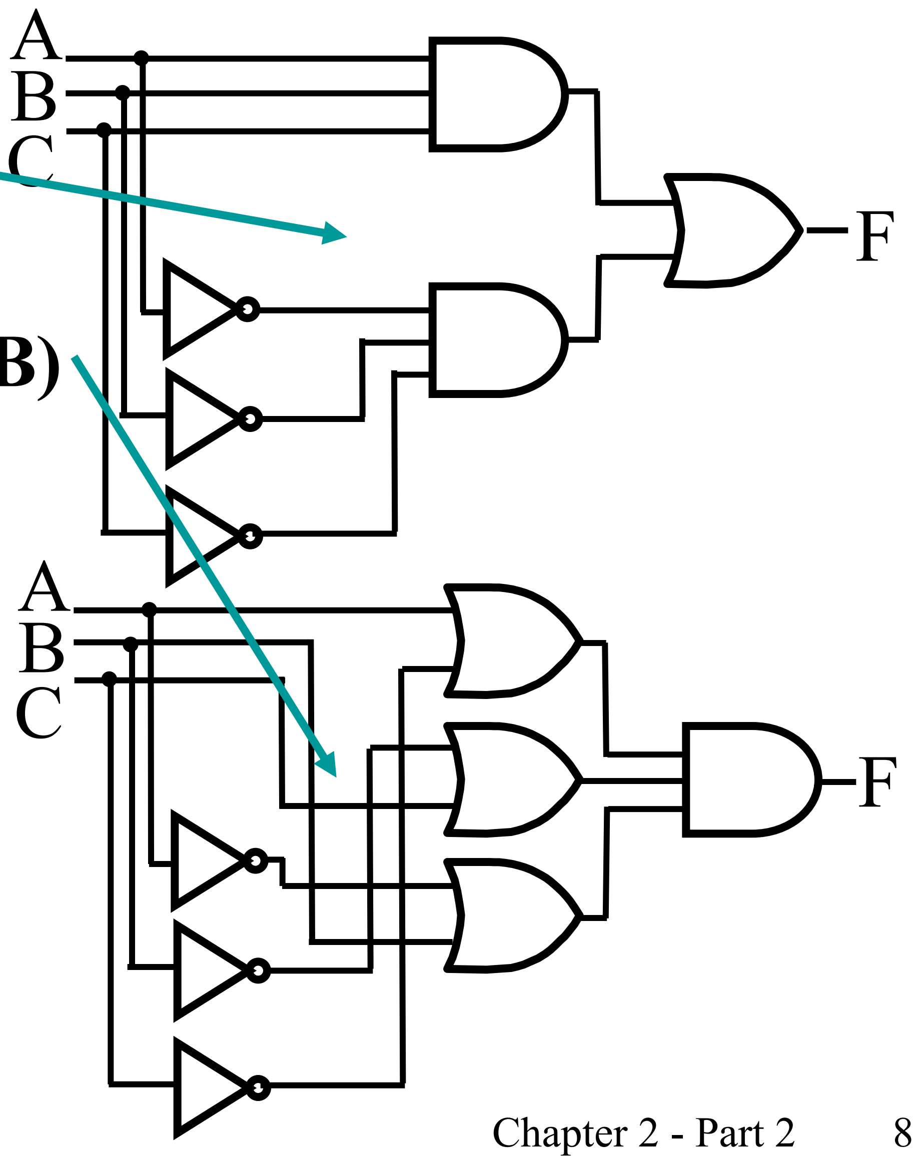
$G = L + 2 = 7$



- 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 (continued)

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



Boolean Function Optimization

- Minimizing the gate input (or literal) cost of a (a set of) Boolean equation(s) reduces 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 m0 and minterm m1 are “adjacent” and differ in the value of the variable y
  - Similarly, minterm m0 and minterm m2 differ in the x variable.
  - Also, m1 and m3 differ in the x variable as well.
  - Finally, m2 and m3 differ in the value of the variable y

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

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:

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

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

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

$$G(x,y) = \left(x\,\overline{y} + x\,y\right) + \left(xy + \overline{x}\,y\right) = x + y$$

$\boxed{\textbf{Duplicate } xy}$

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

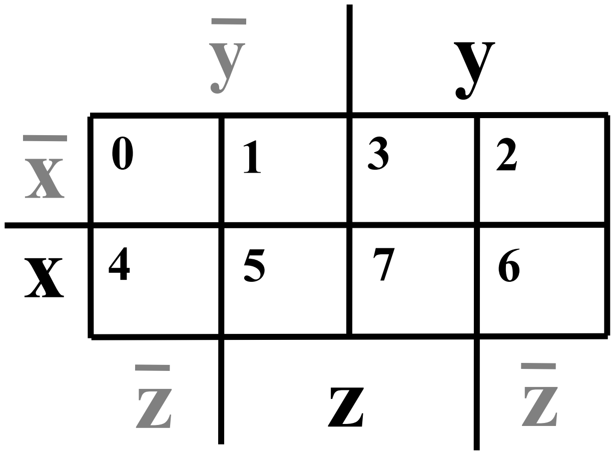
- Where each minterm corresponds to the product terms:

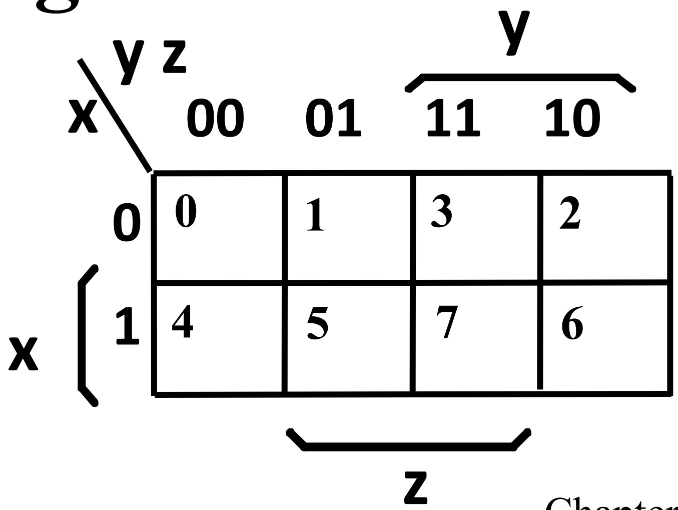
	yz=00	yz=01	yz=11	yz=10
x=0	$\overline{x}\,\overline{y}\,\overline{z}$	$\overline{x}\,\overline{y}\,z$	$\overline{x}\,y\,z$	$\overline{x}\,y\,\overline{z}$
x=1	$x\,\overline{y}\,\overline{z}$	$x\,\overline{y}\,z$	$x\,y\,z$	$x\,y\,\overline{z}$

- Note that if the binary value for an <u>index</u> 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:

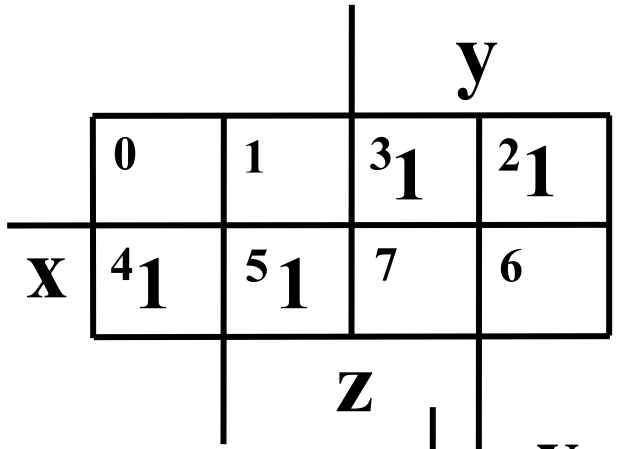




Example Functions

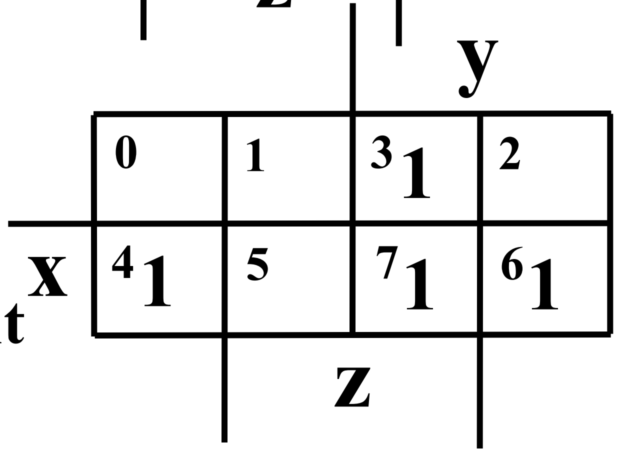
- By convention, we represent the minterms of F by a "1" in the map and leave the minterms of $\overline{F}$ blank
- Example:

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



- Example:

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



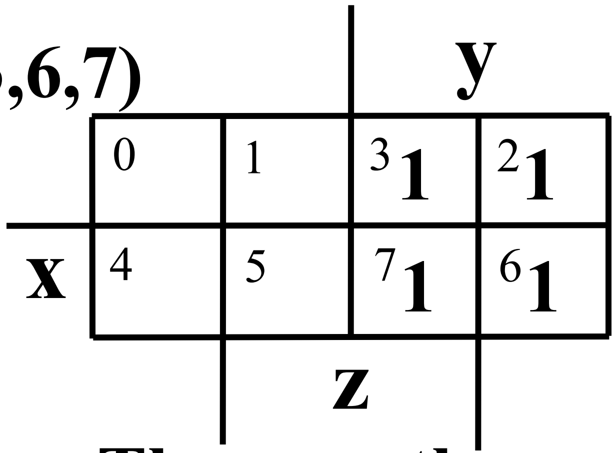
- Learn the locations of the 8 indices based on the variable order shown (x, most significant and z, least significant) on the map boundaries

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
  - Four "adjacent" terms represent a product term with one variable
  - Eight "adjacent" terms is the function of all ones (no variables) = 1.

Example: Combining Squares

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



- Applying the Minimization Theorem three times:

$$\begin{aligned} F(x,y,z) &= \bar{x}\,y\,z + x\,y\,z + \bar{x}\,y\,\bar{z} + x\,y\,\bar{z} \\ &= yz + y\bar{z} \\ &= y \end{aligned}$$

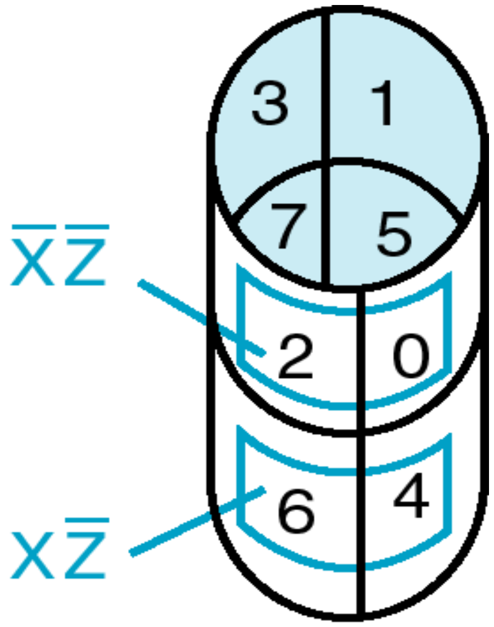
- Thus the four terms that form a 2×2 square correspond to the term "y".

Three-Variable Maps

- **Reduced literal product terms for SOP standard forms correspond to rectangles on K-maps containing cell counts that are powers of 2.**
- **Rectangles of 2 cells represent 2 adjacent minterms; of 4 cells represent 4 minterms that form a "pairwise adjacent" ring.**
- **Rectangles can contain non-adjacent cells as illustrated by the "pairwise adjacent" ring above.**

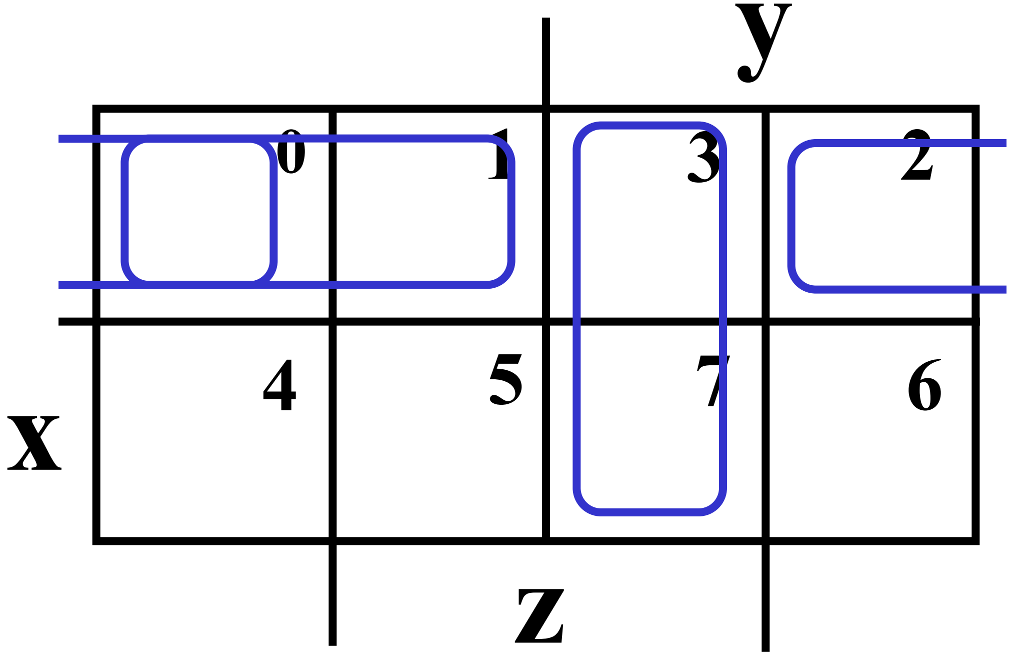
Three-Variable Maps

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



Three-Variable Maps

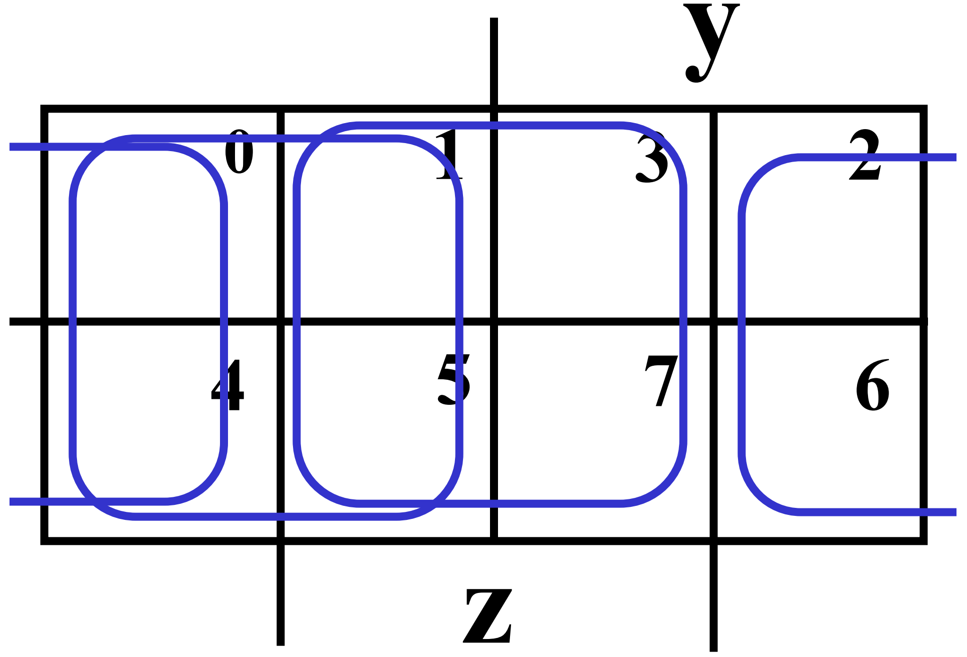
- Example Shapes of 2-cell Rectangles:



- Read off the product terms for the rectangles shown

Three-Variable Maps

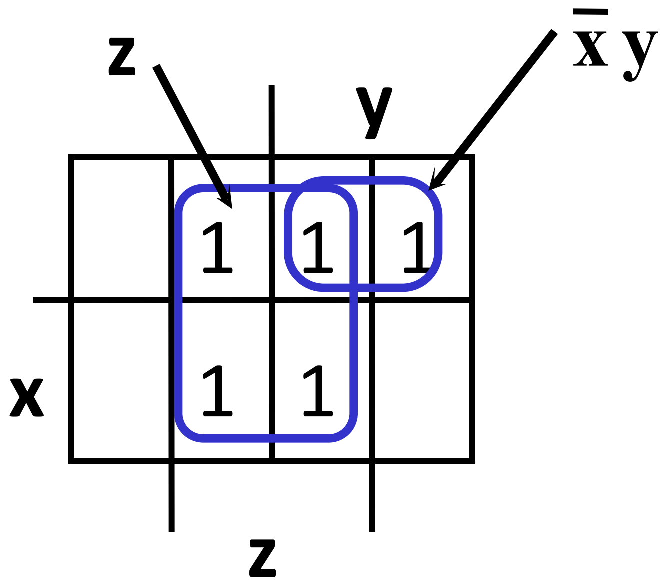
- Example Shapes of 4-cell Rectangles:



- Read off the product terms for the rectangles shown

Three Variable Maps

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



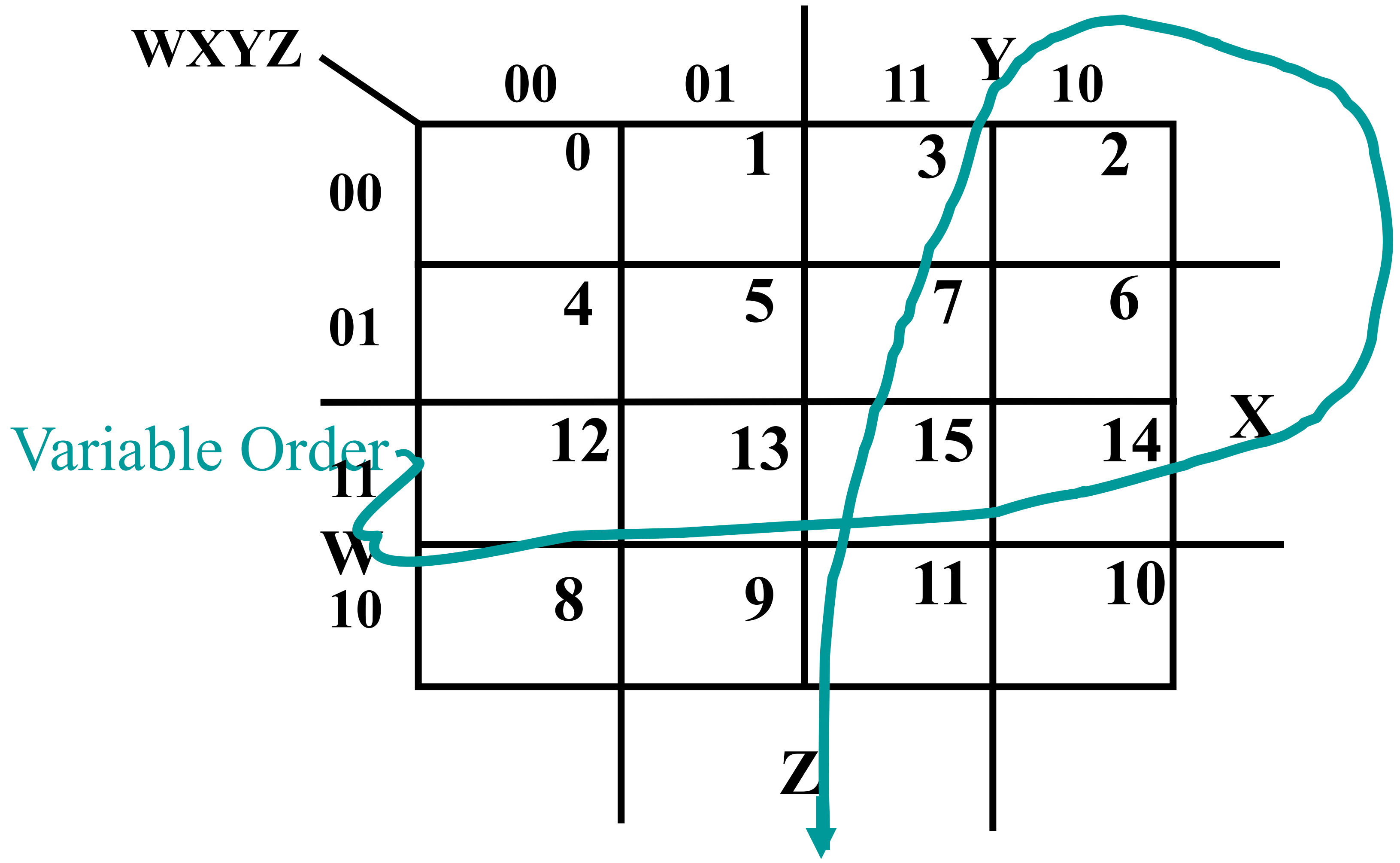
$$F(x, y, z) = z + \bar{x}\,y$$

Three-Variable Map Simplification

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

Four Variable Maps

- **Map and location of minterms:**

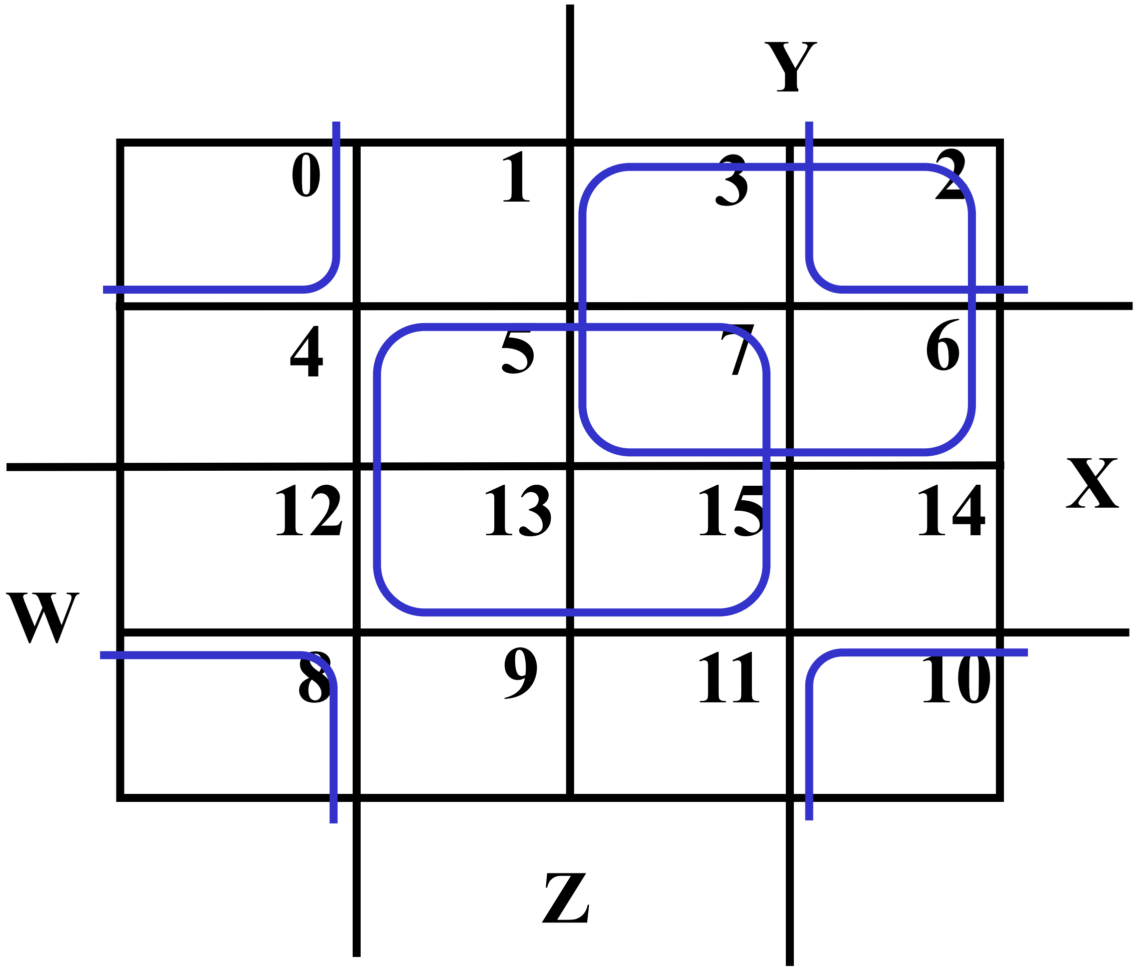


Four Variable Terms

- 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")

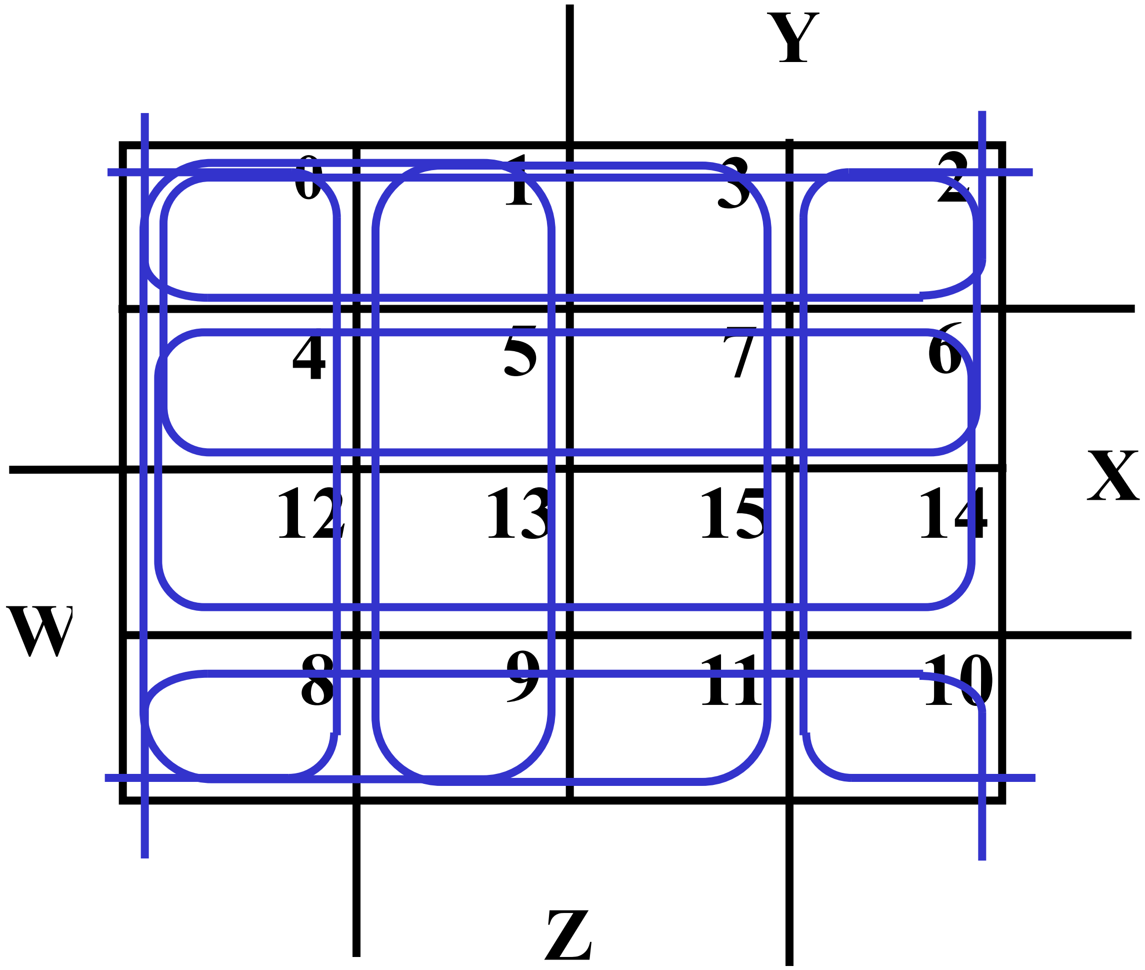
Four-Variable Maps

- **Example Shapes of Rectangles:**



Four-Variable Maps

- **Example Shapes of Rectangles:**

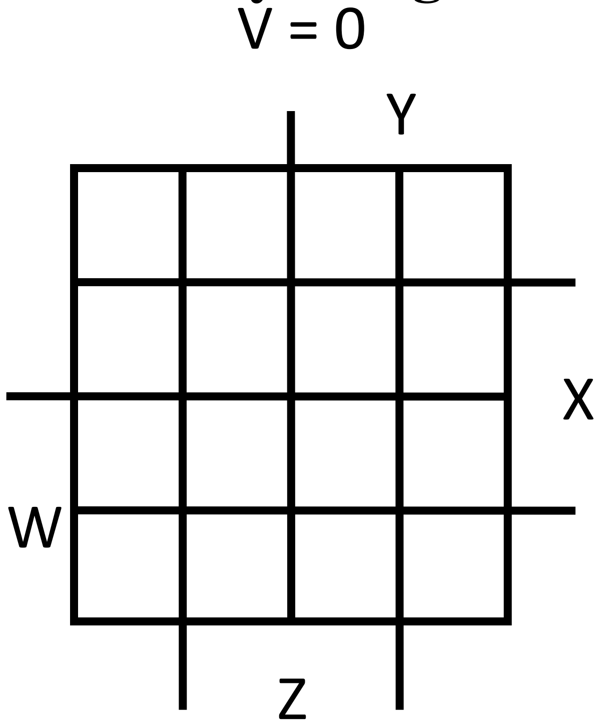


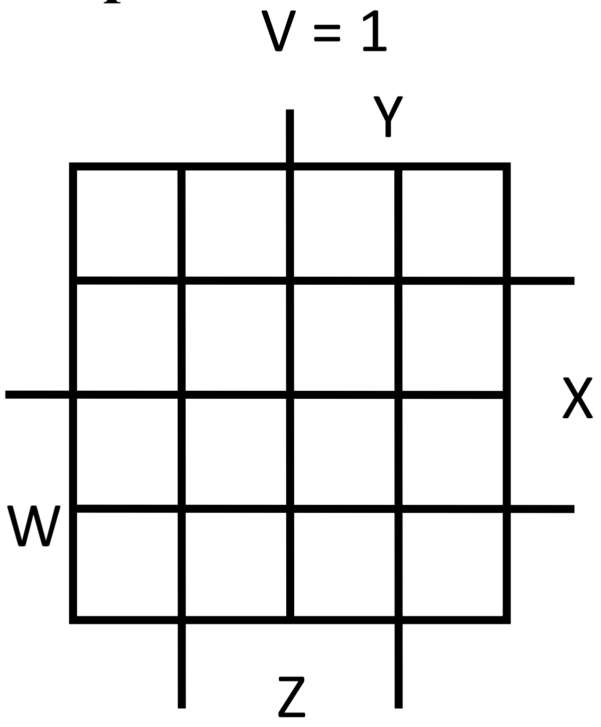
Four-Variable Map Simplification

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

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.**





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**
- **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 <u>never occur</u>, so the output values for these codes are “x” to represent “don’t cares.”**

Don't Cares in K-Maps

- **Example 2: A circuit that represents a very common situation that occurs in computer design has two distinct sets of input variables:**
  - **A, B, and C which take on all possible combinations, and**
  - **Y which takes on values 0 or 1.**

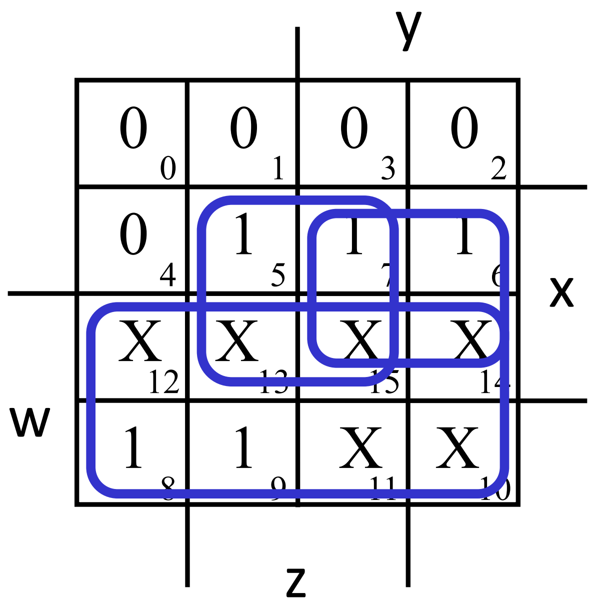
  and a single output Z. The circuit that receives the input Y observes it only for combinations of A, B, and C such A = 1 and B = 1 or C = 0, otherwise ignoring it. Thus, Z is specified only for those combinations, and for all other combinations of A, B, and C, Z is a don't care. Specifically, Z must be specified for $AB + \overline{C} = 1$, and is a don't care for :

  $$\overline{AB + \overline{C}} = (\overline{A} + \overline{B})C = \overline{A}C + \overline{B}C = 1$$

- **Ultimately, each don't care "x" entry may take on either a 0 or 1 value in resulting solutions**
- **For example, an "x" may take on value "0" in an SOP solution and value "1" in a POS solution, or vice-versa.**
- **Any minterm with value "x" *need not* be covered by a prime implicant.**

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:



$F1\,(w,x,y,z) = w + x\,z + x\,y \quad G = 7$

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

$F_2(w, x, y, z) = \overline{w}\,x\,z + \overline{w}\,x\,y + w\overline{x}\,\overline{y} \quad G = 12$

- For this particular function, cost G for the POS solution for $F_1(w,x,y,z)$ is not changed by using the don't cares.

Product of Sums Example

- Find the optimum <u>POS</u> solution:

$F(A, B, C, D) = \Sigma_m(3,9,11,12,13,14,15) + \Sigma_d(1,4,6)$

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

Systematic Simplification

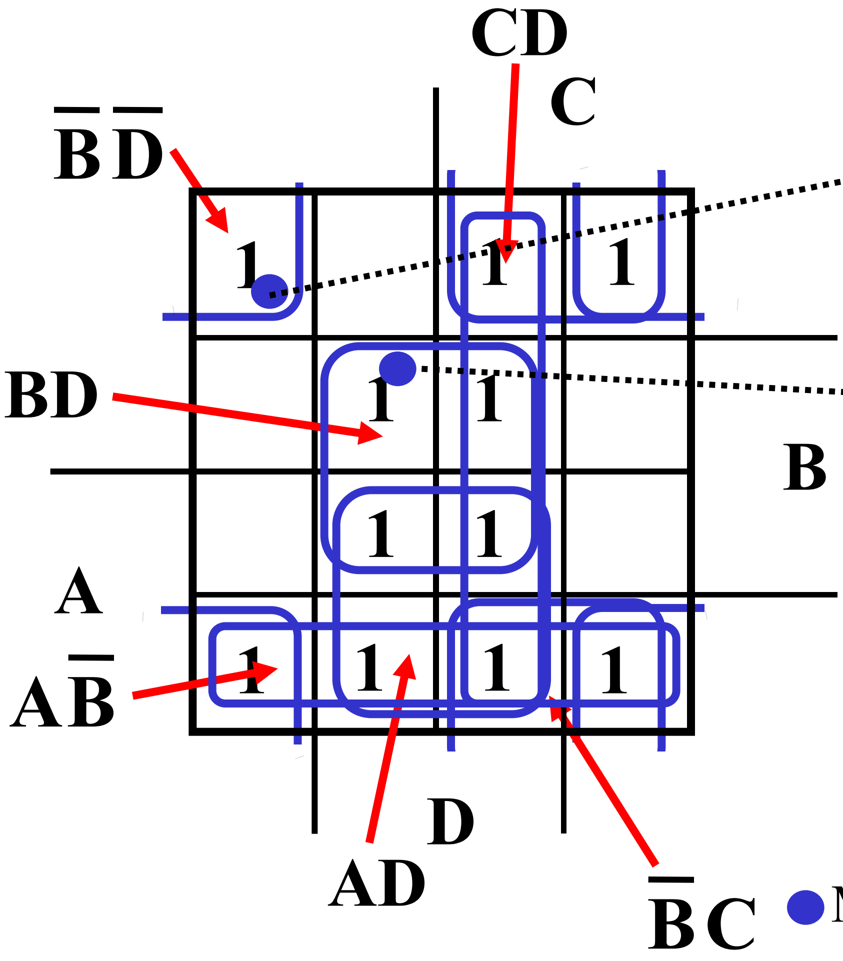
- $F(W, X, Y, Z) = \Sigma_m(3,4,5,7,9,13,14,15)$

Systematic Simplification

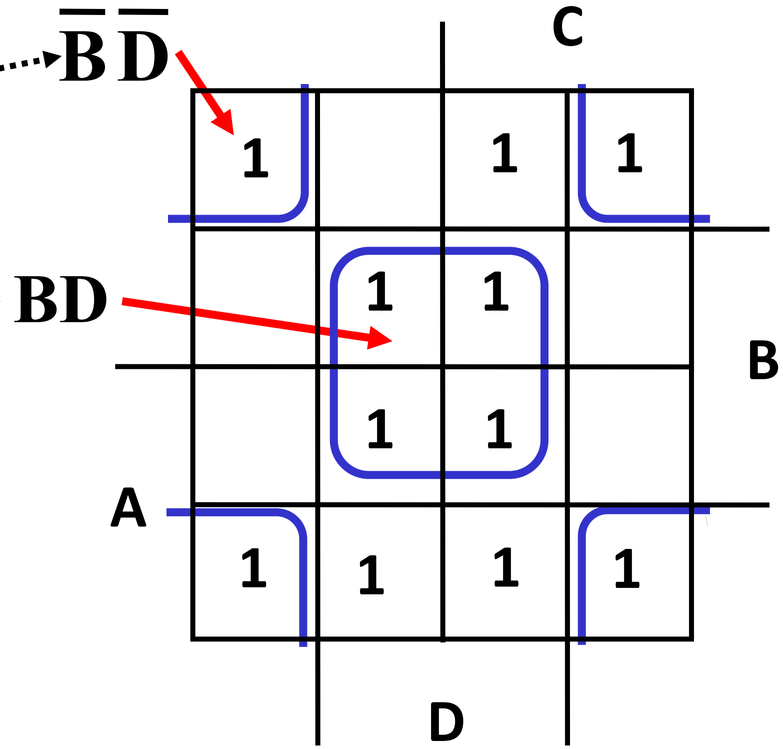
- A *Prime Implicant* is a product term obtained by combining the maximum possible number of adjacent squares in the map into a rectangle with the number of squares a power of 2.
- A prime implicant is called an *Essential Prime Implicant* if it is the only prime implicant that covers (includes) one 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



ESSENTIAL Prime Implicants



● Minterms covered by single prime implicant

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!

Optimization Algorithm

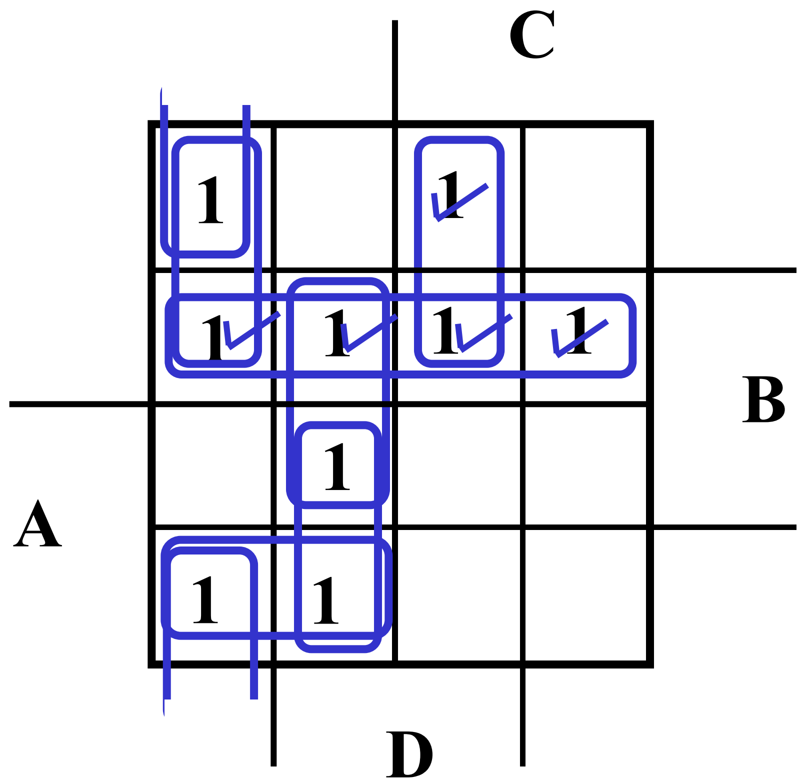
- Find <u>all</u> prime implicants.
- Include <u>all</u> essential prime implicants in the solution
- Select a minimum cost set of non-essential prime implicants to cover all minterms not yet covered:
  - Obtaining an optimum solution: See Reading Supplement - More on Optimization
  - Obtaining a good simplified solution: Use the Selection Rule

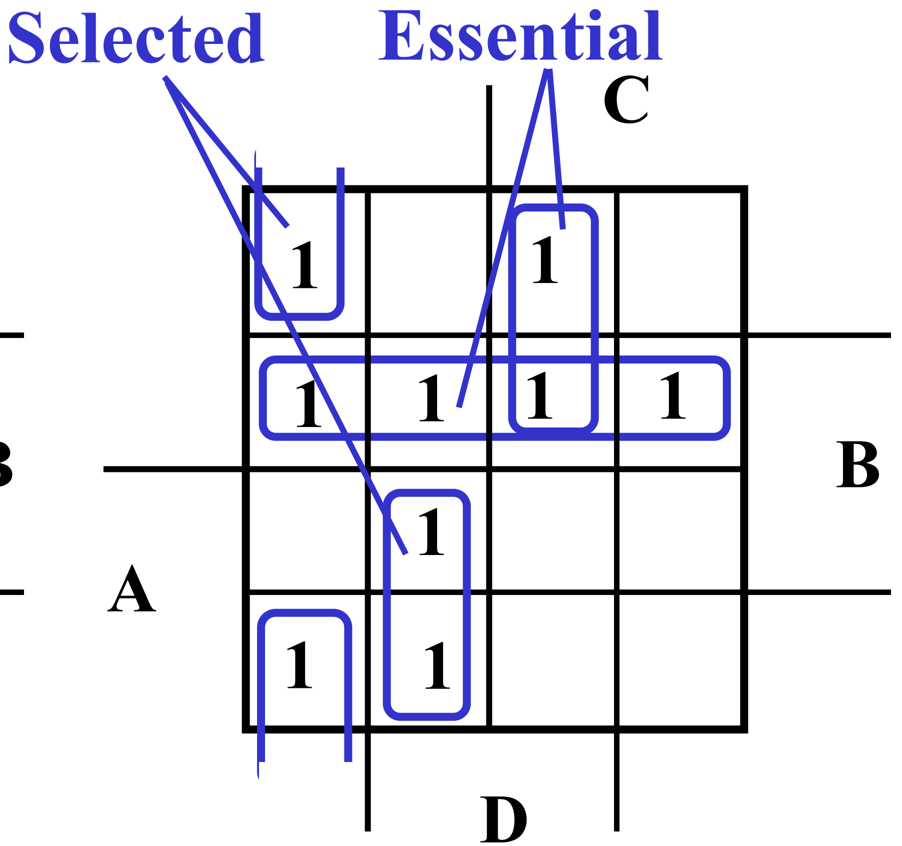
Prime Implicant Selection Rule

- **Minimize the overlap among prime implicants as much as possible. In particular, in the final solution, make sure that each prime implicant selected includes at least one minterm not included in any other prime implicant selected.**

Selection Rule Example

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

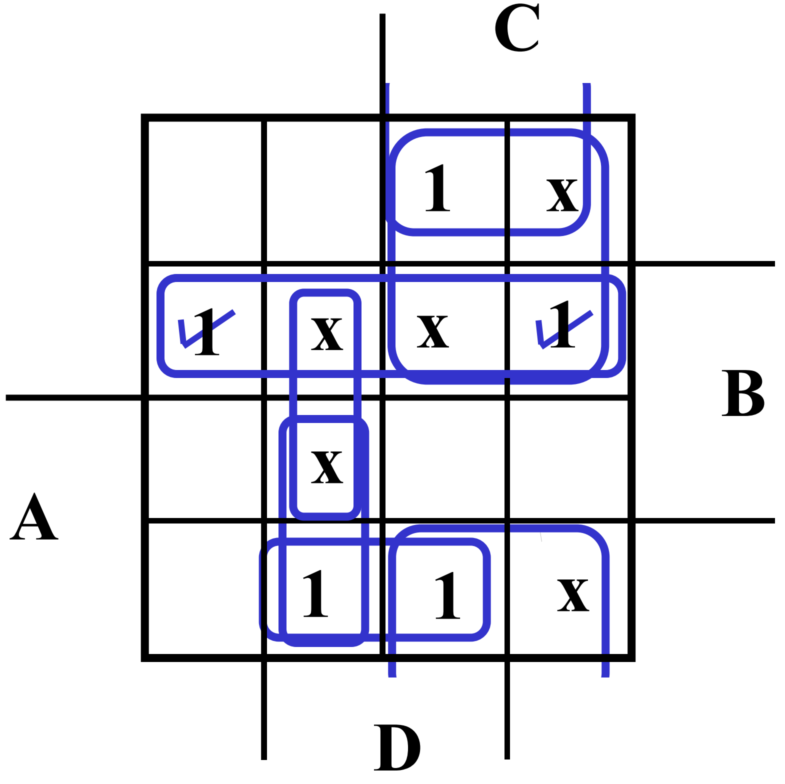


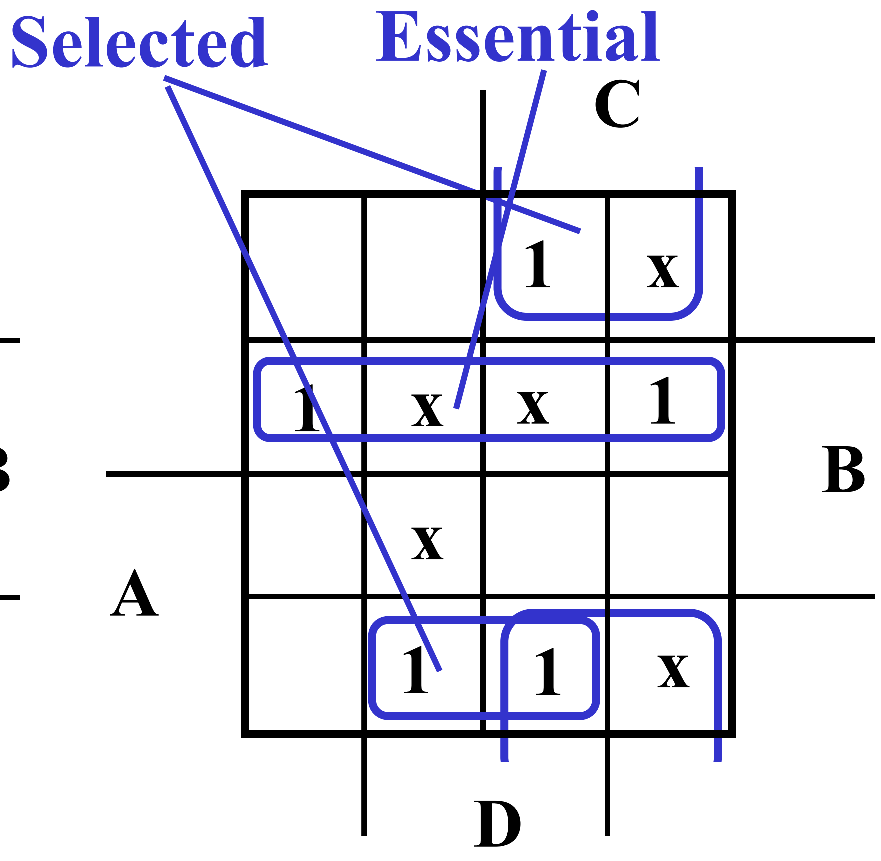


✓ Minterms covered by essential prime implicants

Selection Rule Example with Don't Cares

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





✓ Minterms covered by essential prime implicants

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

Transformation Examples

- **Algebraic Factoring**

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

  - **Factoring:**

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

  - **Factoring again:**

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

  - **Factoring again:**

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

Overview

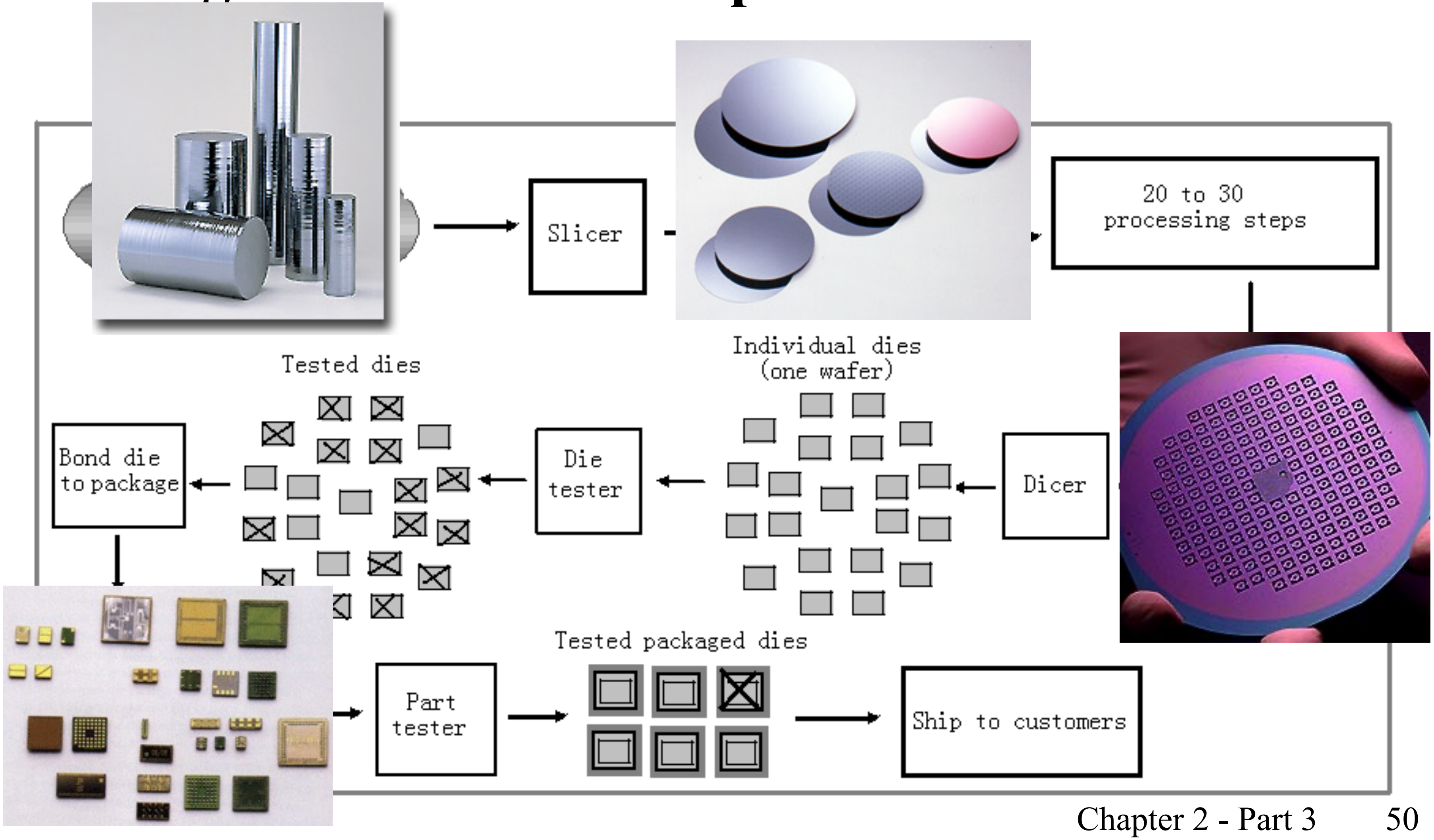
- **Part 1 – Gate Circuits and Boolean Equations**
  - Binary Logic and Gates
  - Boolean Algebra
  - Standard Forms
- **Part 2 – Circuit Optimization**
  - Two-Level Optimization
  - Map Manipulation
  - Practical Optimization (Espresso)
  - Multi-Level Circuit Optimization
- **Part 3 – Additional Gates and Circuits**
  - Other Gate Types
  - Exclusive-OR Operator and Gates
  - High-Impedance Outputs
  - Propagation Delay

Integrated Circuits

- **Integrated circuit (informally, a "chip") is a semiconductor crystal (most often silicon) containing the electronic components for the digital gates and storage elements which are interconnected on the chip.**
- **Terminology - Levels of chip integration**
  - ***SSI*** **(*small-scale integrated*) - fewer than 10 gates**
  - ***MSI*** **(*medium-scale integrated*) - 10 to 100 gates**
  - ***LSI*** **(*large-scale integrated*) - 100 to thousands of gates**
  - ***VLSI*** **(*very large-scale integrated*) - thousands to 100s of millions of gates**

Integrated Circuits

- **Design and manufacture procedure**



Technology Parameters

- Specific gate implementation technologies are characterized by the following parameters:
  - *Fan-in* – the number of inputs available on a gate
  - *Fan-out* – the number of standard loads driven by a gate output
  - *Logic Levels* – the signal value ranges for 1 and 0 on the inputs and 1 and 0 on the outputs (see Figure 1-1)
  - *Noise Margin* – the maximum external noise voltage superimposed on a normal input value that will not cause an undesirable change in the circuit output
  - *Cost for a gate* - a measure of the contribution by the gate to the cost of the integrated circuit
  - *Propagation Delay* – The time required for a change in the value of a signal to propagate from an input to an output
  - *Power Dissipation* – the amount of power drawn from the power supply and consumed by the gate

Fan-out

- **Fan-out can be defined in terms of a standard load**
  - **Example: 1 standard load equals the load contributed by the input of 1 inverter.**
  - ***Transition time* -the time required for the gate output to change from H to L, t_{HL}, or from L to H, t_{LH}**
  - **The *maximum fan-out* that can be driven by a gate is the number of standard loads the gate can drive without exceeding its specified *maximum transition time***

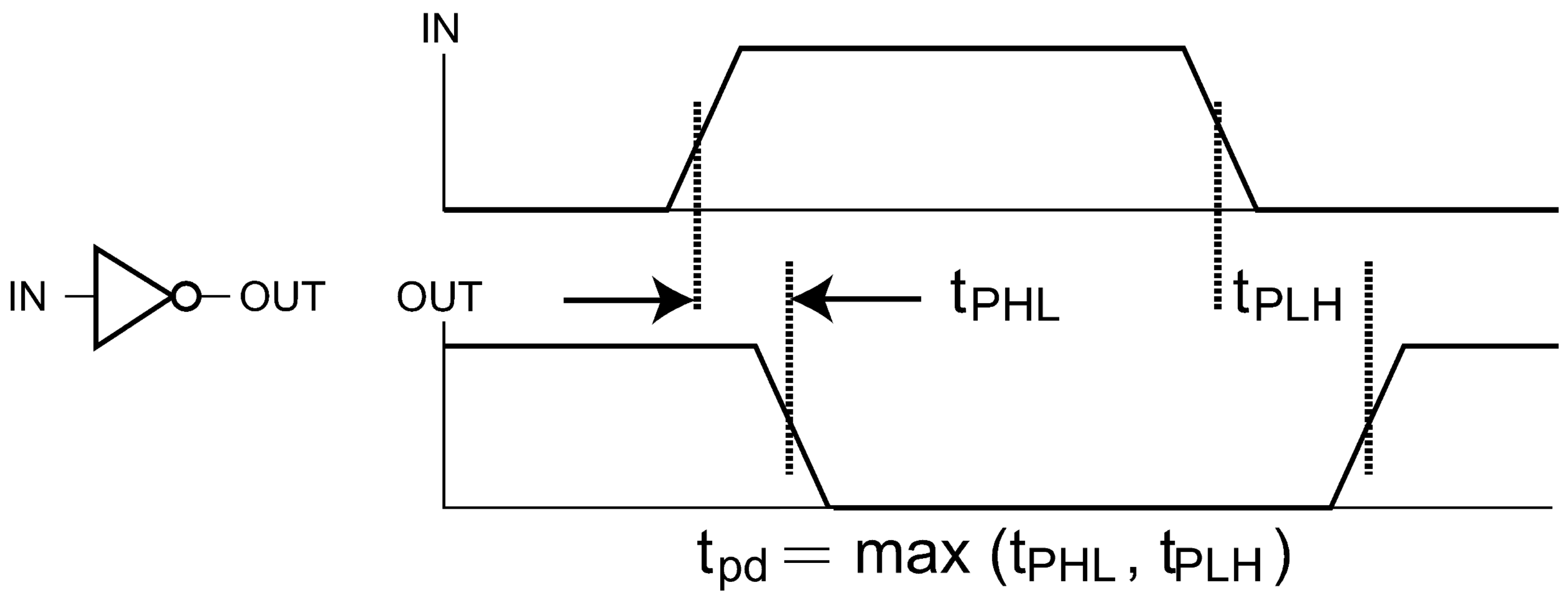
Cost

- **In an integrated circuit:**
  - **The cost of a gate is proportional to the <u>chip area</u> occupied by the gate**
  - **The gate area is roughly proportional to the <u>number and size of the transistors</u> and the <u>amount of wiring</u> connecting them**
  - **Ignoring the wiring area, the gate area is roughly proportional to the <u>gate input count</u>**
  - **So gate input count is a rough measure of gate cost**
- **If the actual chip layout area occupied by the gate is known, it is a far more accurate measure**

Propagation Delay

- *Propagation delay* is the time for a change on an input of a gate to propagate to the output.
- Delay is usually measured at the 50% point with respect to the H and L output voltage levels.
- High-to-low (t_{PHL}) and low-to-high (t_{PLH}) output signal changes may have different propagation delays.
- High-to-low (HL) and low-to-high (LH) transitions are defined with respect to the output, <u>not</u> the input.
- An HL input transition causes:
  - an LH output transition if the gate inverts and
  - an HL output transition if the gate does not invert.

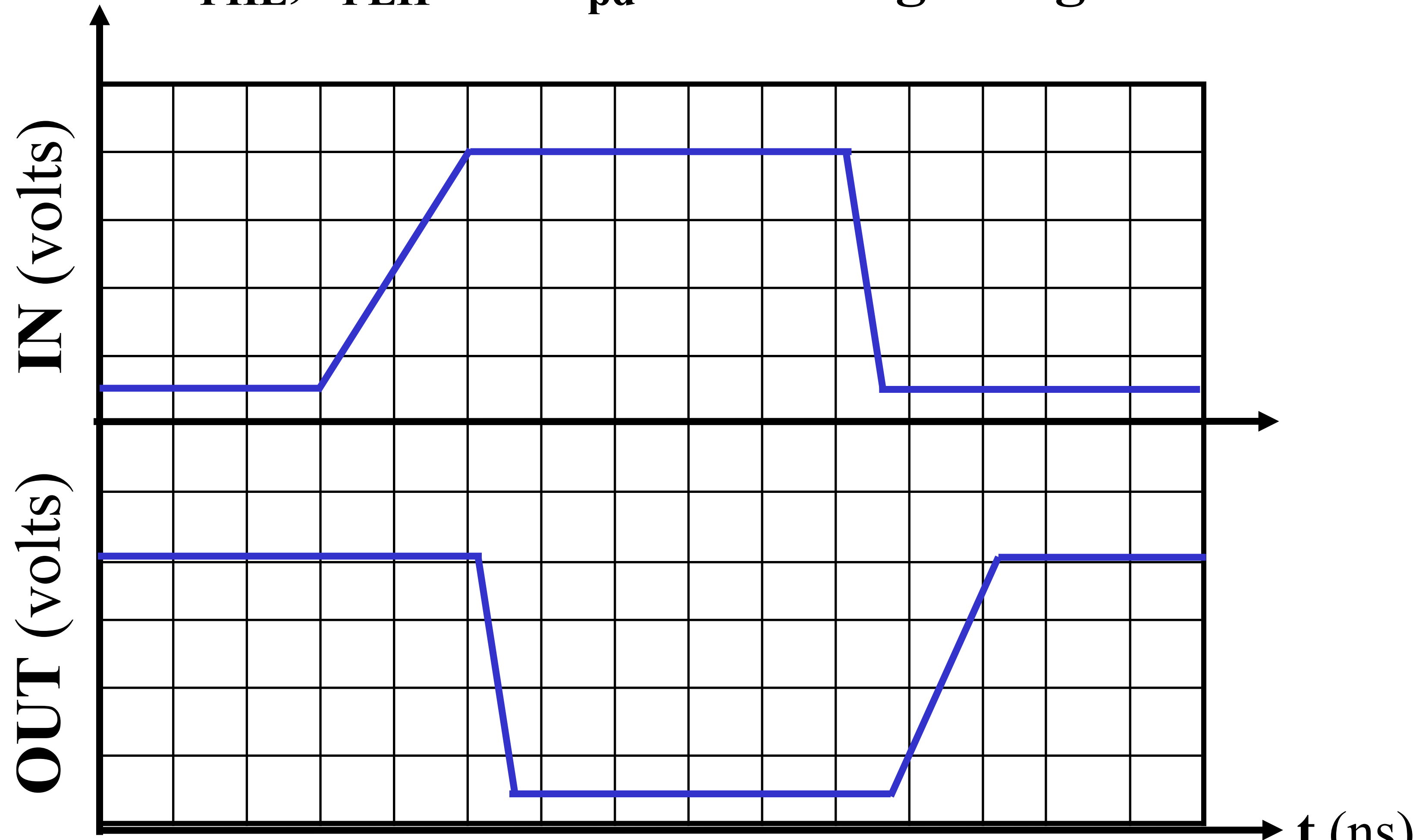
Propagation Delay (continued)



- Propagation delays measured at the midpoint between the L and H values
- What is the expression for the t_{PHL} delay for:
  - a string of *n* identical buffers?
  - a string of *n* identical inverters?

Propagation Delay Example

- Find t_{PHL}, t_{PLH} and t_{pd} for the signals given

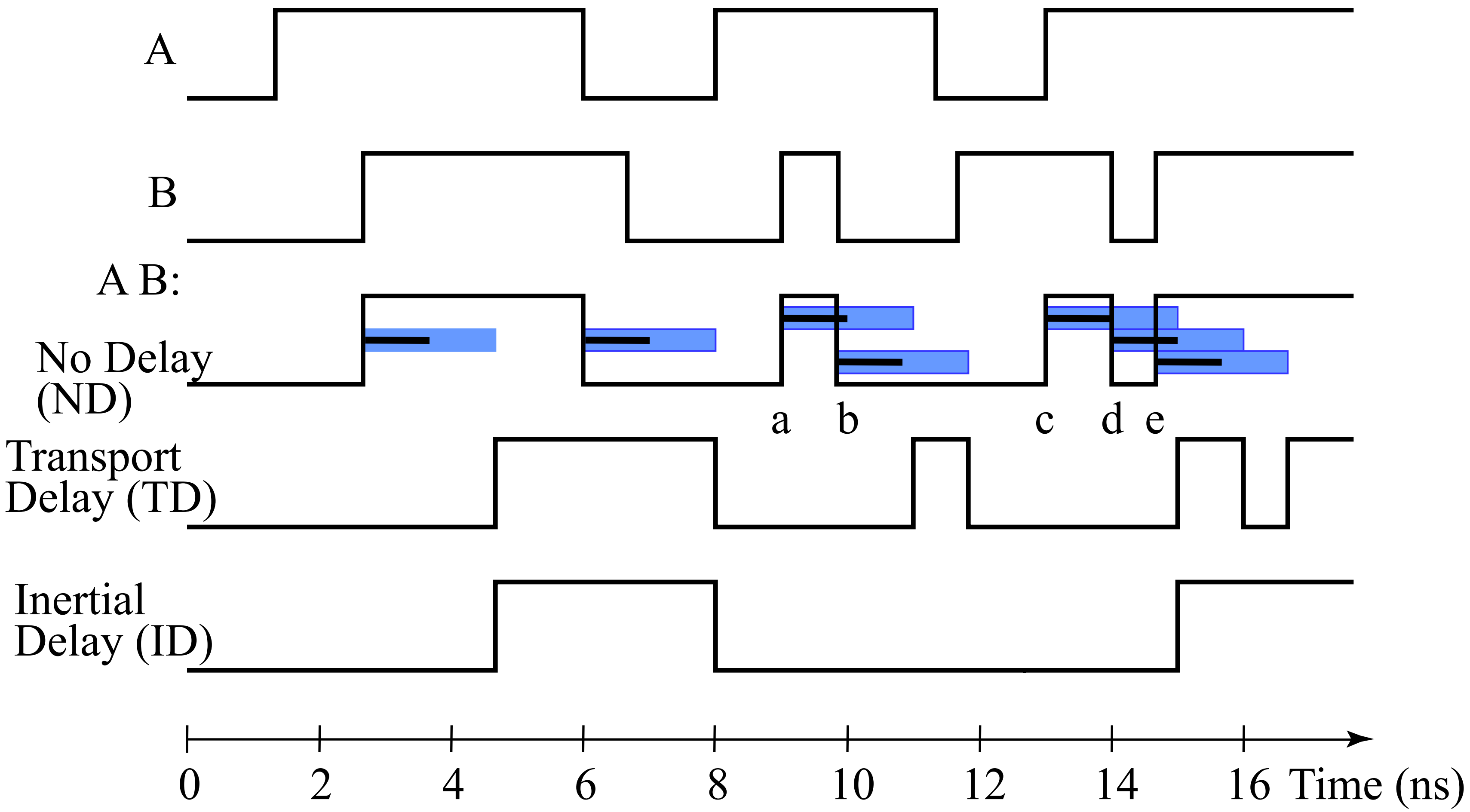


1.0 ns per division

Delay Models

- ***Transport delay*** **- a change in the output in response to a change on the inputs occurs after a fixed specified delay**
- ***Inertial delay*** **- similar to transport delay, except that if the input changes such that the output is to change twice in a time interval less than the** ***rejection time*****, the output changes do not occur. Models typical electronic circuit behavior, namely, rejects narrow "pulses" on the outputs**

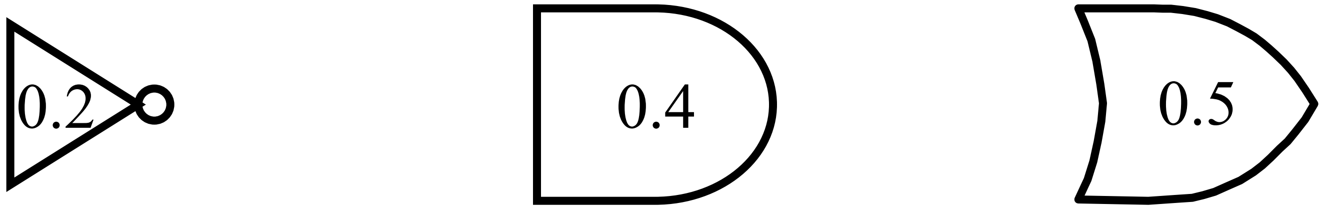
Delay Model Example



Propagation Delay = 2.0 ns Rejection Time = 1 .0 ns

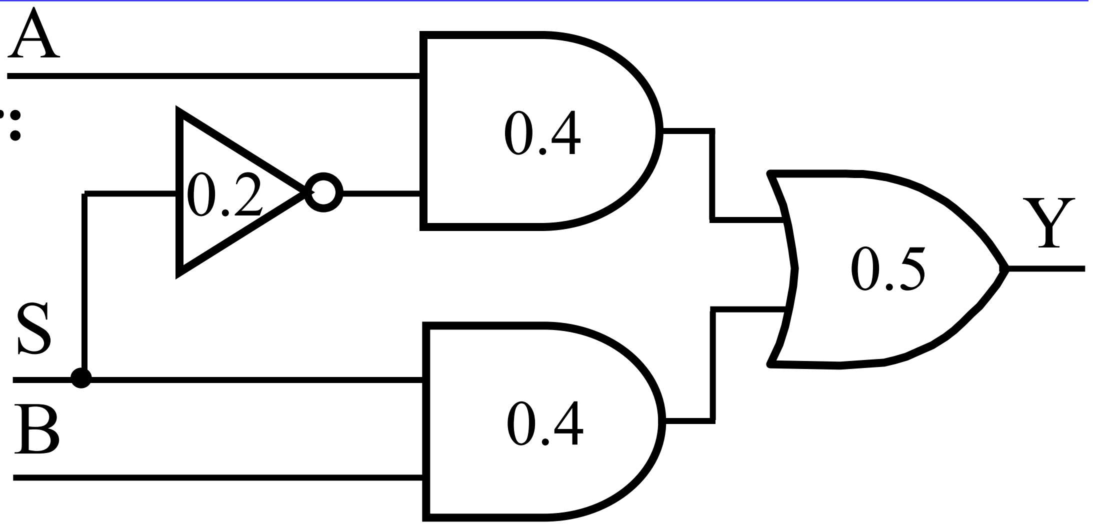
Circuit Delay

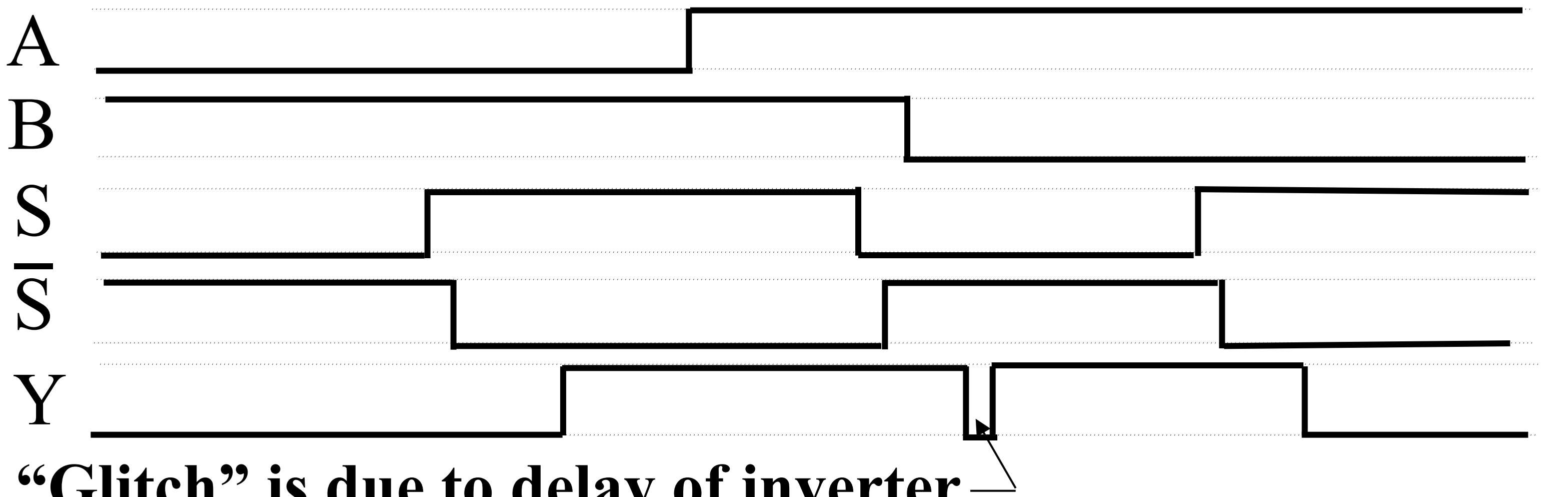
- **Suppose gates with delay *n* ns are represented for *n* = 0.2 ns, *n* = 0.4 ns, *n* = 0.5 ns, respectively:**



Circuit Delay

- **Consider a simple 2-input multiplexer:**
- **With function:**
  - **Y = A for S = 0**
  - **Y = B for S = 1**





- **"Glitch" is due to delay of inverter**

Fan-out and Delay

- The fan-out loading a gate's output affects the gate's propagation delay
- Example:
  - One realistic equation for t_{pd} for a NAND gate with 4 inputs is:

    $$t_{pd} = 0.07 + 0.021 \text{ SL ns}$$

  - SL is the number of standard loads the gate is driving, i. e., its fan-out in standard loads
  - For SL = 4.5, $t_{pd} = 0.165$ ns
- If this effect is considered, the delay of a gate in a circuit takes on different values depending on the circuit load on its output.

Cost/Performance Tradeoffs

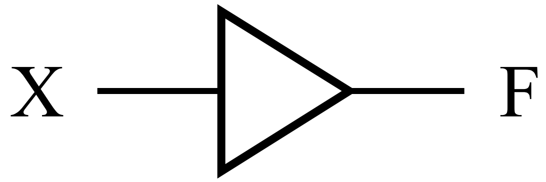
- **Gate-Level Example:**
  - **NAND gate G with 20 standard loads on its output has a delay of 0.45 ns and has a normalized cost of 2.0**
  - **A buffer H has a normalized cost of 1.5. The NAND gate driving the buffer with 20 standard loads gives a total delay of 0.33 ns**
  - **In which if the following cases should the buffer be added?**
    1. **The cost of this portion of the circuit cannot be more than 2.5**
    2. **The delay of this portion of the circuit cannot be more than 0.40 ns**
    3. **The delay of this portion of the circuit must be less than 0.40 ns and the cost less than 3.0**
- **Tradeoffs can also be accomplished much higher in the design hierarchy**
- **Constraints on cost and performance have a major role in making tradeoffs**

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
- Primitive gates will be covered first

Buffer

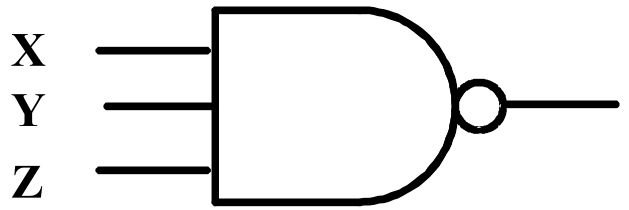
- **A buffer is a gate with the function F = X:**



- **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

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

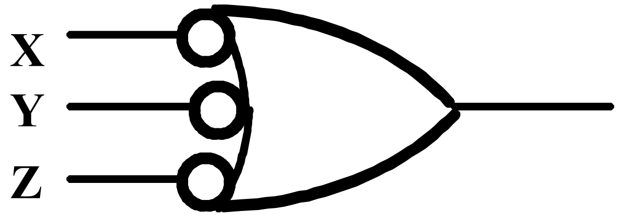


$F(X,Y,Z) = \overline{X\ Y\ Z}$

- NAND represents <u>NOT</u> <u>AND</u>, 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 (continued)

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



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

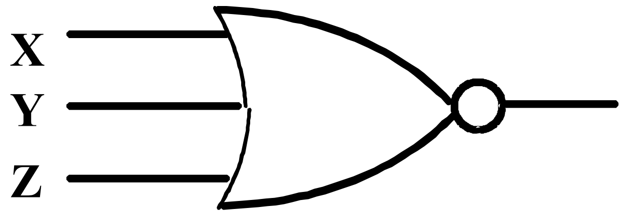
- **This NAND symbol is called Invert-OR, since inputs are inverted and then ORed together.**
- **AND-Invert and Invert-OR both represent the NAND gate. Having both makes visualization of circuit function easier.**
- **A NAND gate with one input degenerates to an inverter.**

NAND Gates (continued)

- **The NAND gate is the natural implementation for CMOS technology in terms of chip area and speed.**
- ***Universal gate*** **- a gate type that can implement any Boolean function.**
- **The NAND gate is a universal gate as shown in Figure 2-24 of the text.**
- **NAND usually does not have a operation symbol defined since**
  - **the NAND operation is not associative, and**
  - **we have difficulty dealing with non-associative mathematics!**

NOR Gate

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

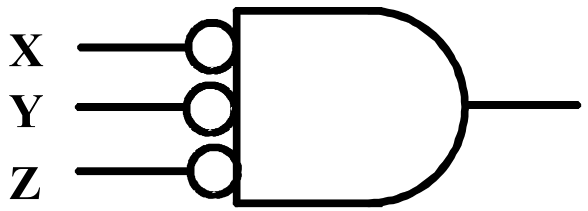


$F(X,Y,Z) = \overline{X+Y+Z}$

- **NOR represents <u>NOT - OR</u>, 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 (continued)

- 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 (continued)

- **The NOR gate is a natural implementation for some technologies other than CMOS in terms of chip area and speed.**
- **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!**

Exclusive OR/ Exclusive NOR

- 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.

Exclusive OR/ Exclusive NOR

- **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.**

Truth Tables for XOR/XNOR

- **Operator Rules: XOR XNOR**

X	Y	$X \oplus Y$
0	0	0
0	1	1
1	0	1
1	1	0

X	Y	$\overline{(X \oplus Y)}$ or $X \equiv Y$
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 ≡?**

XOR/XNOR (Continued)

- 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:

$$X \oplus 0 = X \qquad X \oplus 1 = \overline{X}$$

$$X \oplus X = 0 \qquad X \oplus \overline{X} = 1$$

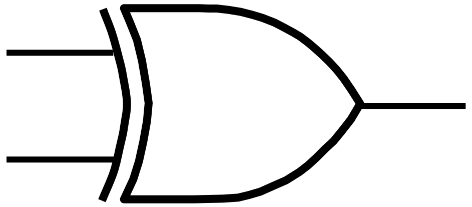
$$X \oplus Y = Y \oplus X$$

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

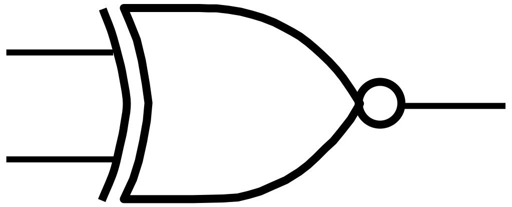
$$X \oplus \overline{Y} = \overline{X \oplus Y}$$

Symbols For XOR and XNOR

- XOR symbol:



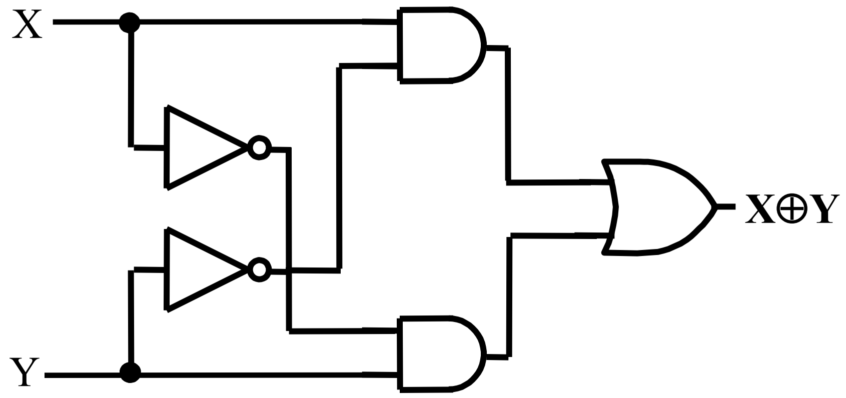
- XNOR symbol:



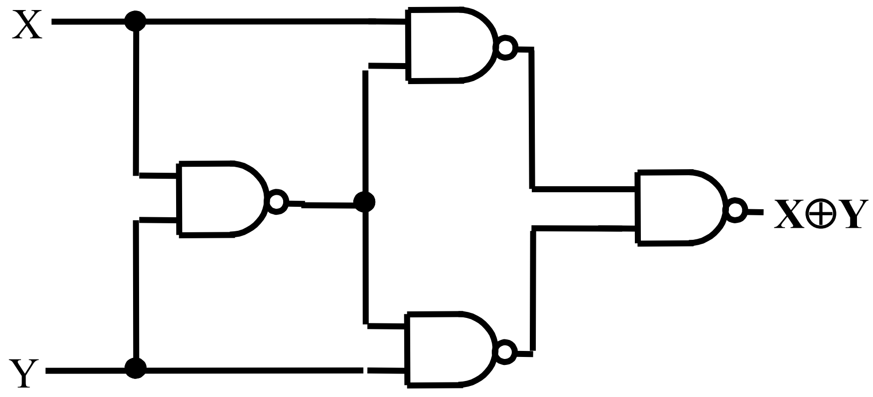
- Shaped symbols exist only for two inputs

XOR Implementations

- The simple SOP implementation uses the following structure:



- A NAND only implementation is:

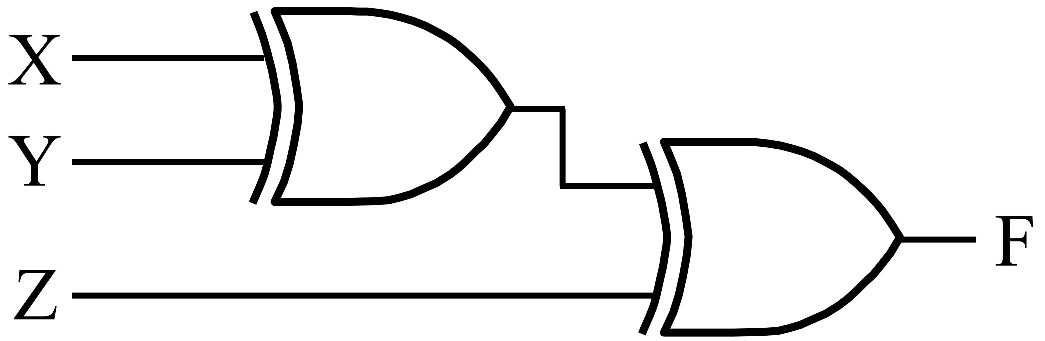


Odd and Even Functions

- 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 four 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

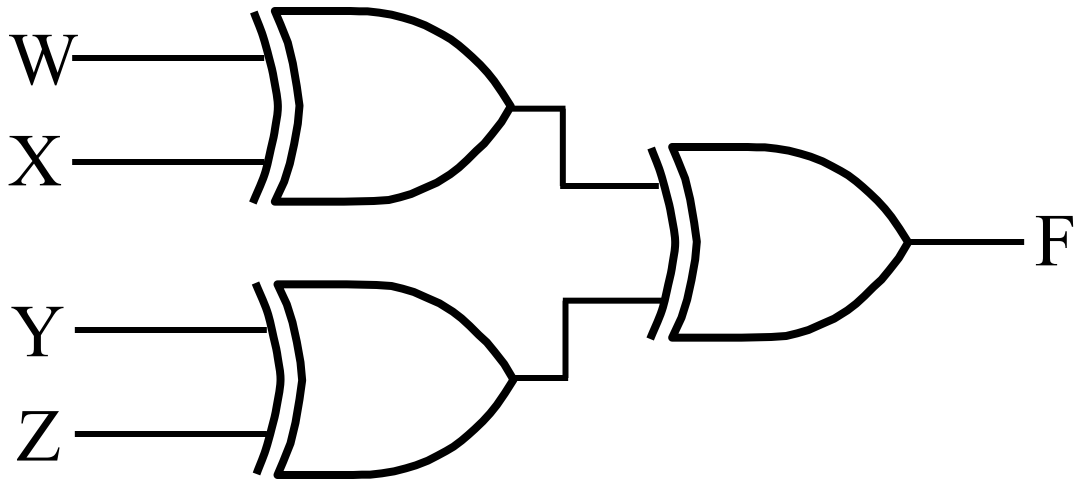
Example: Odd Function Implementation

- Design a 3-input odd function $F = X \oplus Y \oplus Z$ with 2-input XOR gates
- Factoring, $F = (X \oplus Y) \oplus Z$
- The circuit:



Example: Even Function Implementation

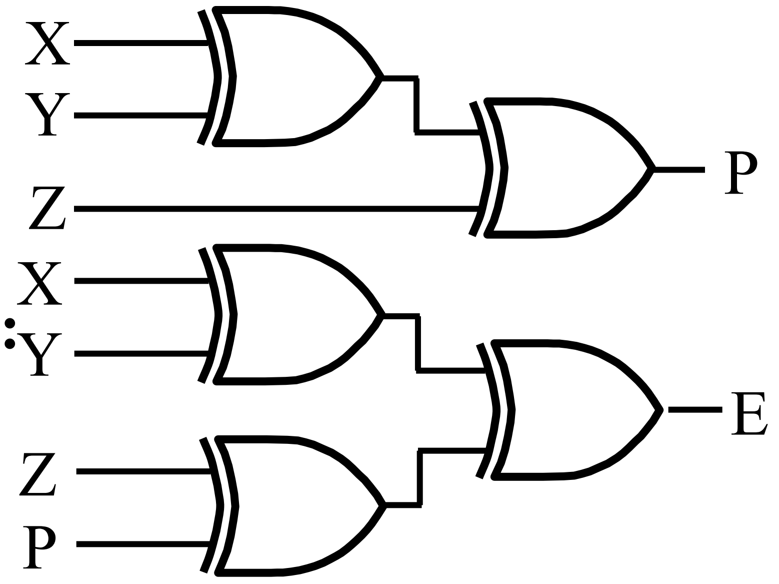
- Design a 4-input odd function $F = W \oplus X \oplus Y \oplus Z$ with 2-input XOR and XNOR gates
- Factoring, $F = (W \oplus X) \oplus (Y \oplus Z)$
- The circuit:



- How to get an even function?

Parity Generators and Checkers

- In Chapter 1, a parity bit added to n-bit code to produce an n + 1 bit code:
  - Add odd parity bit to generate code words with even parity
  - Add even parity bit to generate code words with odd parity
  - Use odd parity circuit to check code words with even parity
  - Use even parity circuit to check code words with odd parity
- Example: n = 3. Generate even parity code words of length four with odd parity generator:
- Check even parity code words of length four with odd parity checker:
- Operation: (X,Y,Z) = (0,0,1) gives (X,Y,Z,P) = (0,0,1,1) and E = 0. If Y changes from 0 to 1 between generator and checker, then E = 1 indicates an error.



Hi-Impedance Outputs

- **Logic gates introduced thus far**
  - **have 1 and 0 output values,**
  - **<u>cannot</u> have their outputs connected together, and**
  - **transmit signals on connections in <u>only one</u> 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 (continued)

- **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**
  - **Optional: a transmission gate (See Reading Supplement: More on CMOS Circuit-Level Design),**

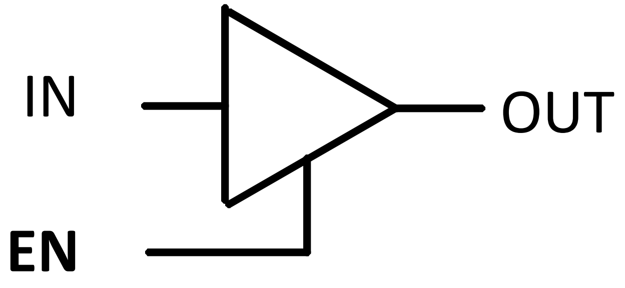
each of which has one data input and one control input.

The 3-State Buffer

- **For the symbol and truth table, IN is the <u>data input,</u> and EN, the <u>control input</u>.**
- **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?

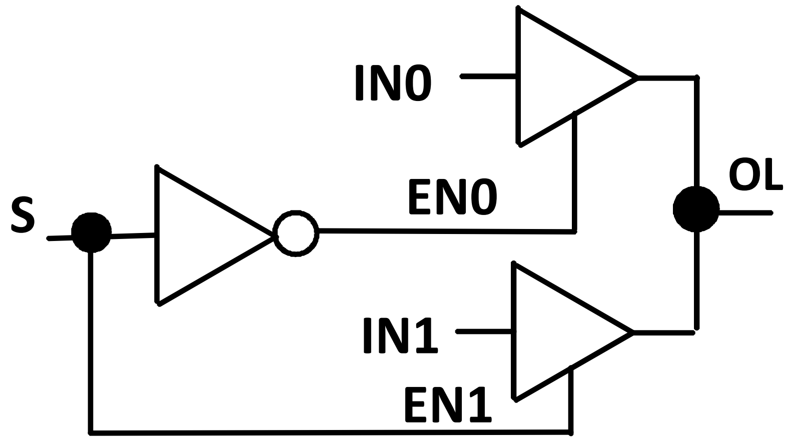
Resolution Table

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:**

EN0	IN0	EN1	IN1	OL
0	X	1	0	0
0	X	1	1	1
1	0	0	X	0
1	1	0	X	1
0	X	0	X	X



- **Since EN0 = $\overline{S}$ and EN1 = S, one of the two buffer outputs is always Hi-Z plus the last row of the table never occurs.**

More Complex Gates

- The remaining complex gates are SOP or POS structures with and without an output inverter.
- The names are derived using:
  - A - AND
  - O - OR
  - I - Inverter
  - Numbers of inputs on first-level “gates” or directly to second-level “gates”

More Complex Gates (continued)

- **Example: AOI - AND-OR-Invert consists of a single gate with AND functions driving an OR function which is inverted.**
- **Example: 2-2-1 AO has two 2-input ANDS driving an OR with one additional OR input**
- **These gate types are used because:**
  - **the number of transistors needed is fewer than required by connecting together primitive gates**
  - **potentially, the circuit delay is smaller, increasing the circuit operating speed**

Assignments

- 2-15c; 2-17b; 2-19a; 2-22a; 2-25b; 2-29; 2-30; 2-31