Section

3

Boolean Algebra

The Building Blocks of Digital Logic Design

Section Overview

Binary Operations (AND, OR, NOT), Basic laws, Proof by Perfect Induction, De Morgan's Theorem, Canonical and Standard Forms (SOP, POS), Gates as SSI Building Blocks (Buffer, NAND, NOR, XOR)

Source: UCI Lecture Series on Computer Design — Gates as Building Blocks, Digital Design Sections 1-9 and 2-1 to 2-7, Digital Logic Design CECS 201 Lecture Notes by Professor Allison Section II — Boolean Algebra and Logic Gates, Digital Computer Fundamentals Chapter 4 — Boolean Algebra and Gate Networks, Principles of Digital Computer Design Chapter 5 — Switching Algebra and Logic Gates, Computer Hardware Theory Section 6.3 — Remarks about Boolean Algebra, An Introduction To Microcomputers pp. 2-7 to 2-10 — Boolean Algebra and Computer Logic.

Sessions: Four(4)

Topics:

1) Binary Operations and Their Representation

2) Basic Laws and Theorems of Boolean Algebra

3) Derivation of Boolean Expressions (Sum-of-products and Product-of-sums)

4) Reducing Algebraic Expressions

5) Converting an Algebraic Expression into Logic Gates

6) From Logic Gates to SSI Circuits

Binary Operations and Their Representation

The Problem

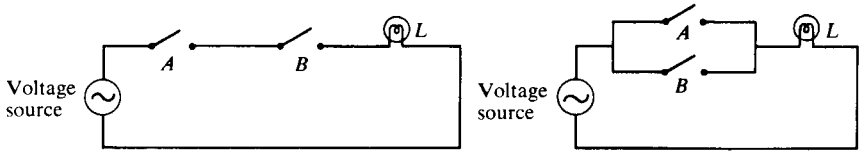
Modern digital computers are designed using techniques and symbology from a field of mathematics called modern *algebra.* Algebraists have studied for a period of over a hundred years mathematical systems called Boolean algebras. Nothing could be more simple and normal to human reasoning than the rules of a Boolean algebra, for these

originated in studies of how we reason, what lines of reasoning are valid, what constitutes proof, and other allied subjects. The name Boolean algebra honors a fascinating English mathematician, George Boole, who in 1854 published a classic book, "An Investigation of the Laws of Thought, on Which Are Founded the Mathematical Theories of Logic and Probabilities." Boole's stated intention was to perform a mathematical analysis of logic. The work of Boole is perhaps best introduced by the following quotation from the first chapter:

The design of the following treatise is to investigate the fundamental laws of those operations of the mind by which reasoning is performed; to give expression to them in the symbolical language of a Calculus, and upon this foundation to establish the science of Logic and construct its method; to make that method itself the basis of a general method for the application of the mathematical doctrine of Probabilities; and, finally, to collect from the various elements of truth brought to view in the course of these inquiries some probable intimations concerning the nature and constitution of the human mind.

Starting with his investigation of the laws of thought, Boole constructed a "logical algebra."

Boolean algebra was first brought to bear on problems which had arisen in the design of relay switching circuits in 1938 by Claude E. Shannon, a research assistant in the department of electrical engineering at the Massachusetts Institute of Technology. A version of Shannon's thesis, written at MIT for the degree of Master of Science, was published under the title, "A Symbolic Analysis of Relay and Switching Circuits." This paper presented a method for representing any circuit consisting of combinations of switches and relays by a set of mathematical expressions, and a calculus was developed for manipulating these expressions. The calculus used was shown to be based on the rules of Boolean algebra.



(a) Switches in series – logic AND (b) Switches in parallel – logic OR

There are several advantages in having a mathematical technique for the description of the internal workings of a computer. For one thing, it is often far more convenient to calculate with expressions used to represent switching circuits than it is to use schematic or even logical diagrams. Just as an ordinary algebraic expression may be simplified by means of the basic theorems, the expression describing a given switching circuit network may also be reduced or simplified. This enables the logical designer to simplify the circuitry used, achieving economy of construction and reliability of operation. Boolean algebra also provides an economical and straightforward way of describing the circuitry used in computers.

Boolean Variables

Boolean algebra allows the concise description and manipulation of binary variables; although it by no means restricted to base 2 systems. Variables in Boolean algebra have a unique characteristic; they may assume only one of two possible values.

Value (Bit)	Alternate Names
0	F, False, No, OFF, LOW
1	T, True, Yes, ON, HIGH

Therefore if $x \neq 0$, then $x = 1$
and if $x \neq 1$, then $x = 0$

Boolean Operations

Boolean Algebra

Boolean algebra operates with three functional operators — the building blocks of digital logic design — Complement, OR, and AND. These building blocks are comparable to taking the negative, adding, and multiplying in ordinary algebra.

Operator Name	Alternate Name	Example	Alternate Representations
NOT	complement inversion	$\bar{x}$	x'
OR	union logical addition	$x + y$	$x \bigcup y,\ x \vee y,\ x \vee y$
AND	intersection logical multiplication	xy	$x \cdot y,\ x \bigcap y,\ x \& y$

Truth Table

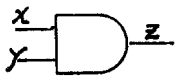
The possible input and output combinations can be arranged in tabular form, called a Truth Table.

INPUT		OUTPUT		
		NOT	OR	AND
x	y	$\bar{x}$	$x + y$	xy
0	0	1	0	0
0	1		1	0
1	0	0	1	0
1	1		1	1

Gates

The OR and AND Boolean operations are physically realized by two types of electronic circuits called OR gates and AND gates. A third circuit called a Buffer, does nothing to the logic.

AND Gate



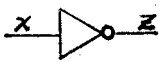
OR Gate

Buffer

NOT

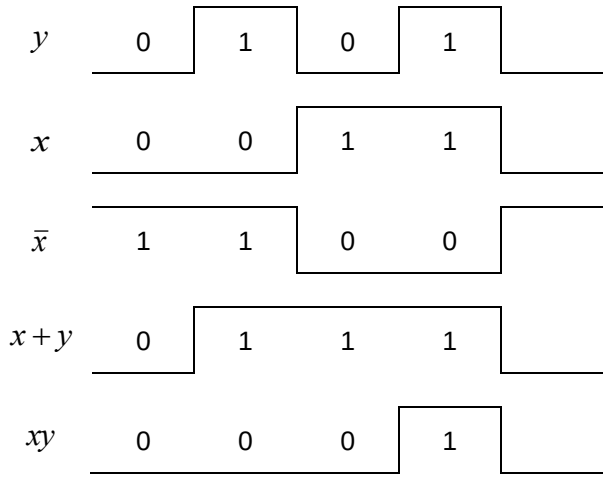
For any of the above functions, inversion — the Boolean NOT operation — of a input or output is denoted by a circle. Inversion is indicated in a Boolean expression by a line over a variable or expression.

Inverter



Timing Diagram

The possible input and output combinations can also be presented as a timing diagram. A timing diagram is a picture that shows the input and output waveforms of a digital circuit.



Basic Laws and Theorems of Boolean Algebra

The last section presented Boolean variables and the three basic operations. Although simple in appearance, these rules may be used to construct a Boolean algebra, determining all the relationships which follow. The following list of useful relationships may be proved using the "proof by perfect induction."

Law		Dual (D)	
1	$\overline{\overline{x}} = x$		Involution
	OR Laws	**AND Laws**	
2	$0 + x = x$	$1 \cdot x = x$	Identity element under addition is 0 and under multiplication it is 1
3	$1 + x = 1$	$0 \cdot x = 0$	Dominance
4	$x + x = x$	$x \cdot x = x$	Idempotent
5	$x + \overline{x} = 1$	$x \cdot \overline{x} = 0$	Complements *not valid under normal algebra*
	Commutative		
6	$x + y = y + x$	$x \cdot y = y \cdot x$	Commutative law of addition and multiplication — order does not affect result.
	Associative		can be derived from the above rules
7	$x + (y + z) = (x + y) + z$	$x(yz) = (xy)z$	Associative law of addition and multiplication — any two may be summed or multiplied together followed by the third
	Distributive		
8	$x(y + z) = xy + xz$	$x + yz = (x + y)(x + z)$	product of a monomial (x) multiplied by a polynomial (y+z) is equal to the sum of the products of the monomial multiplied by each term of the polynomial. *The dual is not valid under normal algebra*
Theorem	**Simplification**		
9	$x + xy = x$	$x(x + y) = x$	Absorption
10	$x + \overline{x}y = x + y$	$x(\overline{x} + y) = xy$	Degenerate-Reflect
	De Morgan's		
11	$\overline{x + y} = \overline{x} \cdot \overline{y}$	$\overline{x \cdot y} = \overline{x} + \overline{y}$	

Principle of Duality

There exists a basic duality which underlies all Boolean algebra. The laws and theorems which have been presented can all be divided into pairs. In proving the laws and theorems, it is then necessary only to prove one theorem, and the "dual" of the theorem follows necessarily. To form the dual of an algebraic expression you simply need to:

1) Interchange AND and OR operators

2) Replace 1's with 0's and 0's with 1's

Proof by Perfect Induction

The rules given may be used to simplify Boolean expressions, just as the rules of normal algebra may be used to simplify expressions.

Proof of the Dual of the Distributive Law

Notice that, among others, rule 8D, does not apply to "normal" algebra. The rule however can be obtained from the preceding rules as follows:

$x + yz = (x + y)(x + z)$

Algebraic Proof A

$(x + y)(x + z) = x(x + y) + y(x + z)$	8. Distributive Law by Extension
$xx + xy + yx + yz$	8. Distributive Law
$x + xy + yx + yz$	4D. Idempotent
$x + yx + yz$	9. Absorption
$x + yz$	9. Absorption

Algebraic Proof B

$x + yz = x + yz$	
$x(1 + y) + yz$	2. Identity under multiplication
$x + xy + yz$	8. Distributive Law
$x(1 + z) + xy + yz$	3. Dominance
$xx + xz + xy + yz$	8. Distributive Law and 4D. Idempotent
$x(x + z) + yx + yz$	8. Distributive Law and 6D. Commutative Law
$x(x + z) + y(x + z)$	8. Distributive Law
$(x + z)x + (x + z)y$	6D. Commutative Law
$(x + z)(x + y)$	8. Distributive Law (use substitution or by extension)
$(x + y)(x + z)$	6D. Commutative Law

Proof by Perfect Induction

Since rule 8D does not apply to normal algebra, it is interesting to test the rule using the "proof by perfect induction." It will therefore be necessary to construct truth tables for the right hand and left hand sides of the equation and compare the results.

Step 1

x	y	z	yz	$x+yz$
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	1
1	0	0	0	1
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Step 2

x	y	z	$x+y$	$x+z$	$(x+y)(x+z)$
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	1	0	0
0	1	1	1	1	1
1	0	0	1	1	1
1	0	1	1	1	1
1	1	0	1	1	1
1	1	1	1	1	1

Notice that the last columns of the two tables are identical. This proves (by means of the proof by perfect induction) that the expressions are equivalent

Proof of Simplification Theorems

$x + xy = x$

Algebraic Proof

$x + xy = x \cdot 1 + xy$ — 2. Identity under multiplication

$x(1 + y)$ — 8. Distributive Law

x — 3. Dominance

Proof by Perfect Induction

x	y	xy	$x+xy$
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

Notice that the first and last columns of the two tables are identical. This proves (by means of the proof by perfect induction) that the expressions are equivalent

---

$x(x + y) = x$

Algebraic Proof

$x(x + y) = xx + xy$ — 8. Distributive Law

$x + xy$ — 4D. Idempotent

x — 9. Absorption (see last proof)

Proof by Perfect Induction

x	y	$x+y$	$x(x+y)$
0	0	0	0
0	1	1	0
1	0	1	1
1	1	1	1

Notice that the first and last columns of the two tables are identical. This proves (by means of the proof by perfect induction) that the expressions are equivalent

---

$x + \bar{x}y = x + y$

Algebraic Proof

$x + \bar{x}y = (x + y)(x + \bar{x})$	8D. Distributive (use substitution)
$x + y$	5. Complements

Proof by Perfect Induction

Step 1

x	y	$\bar{x}y$	$x + \bar{x}y$
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	1

Step 2

x	y	$x+y$
0	0	0
0	1	1
1	0	1
1	1	1

Notice that the first and last columns of the two tables are identical. This proves (by means of the proof by perfect induction) that the expressions are equivalent

---

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

Algebraic Proof

$x(\bar{x} + y) = x\bar{x} + xy$	8. Distributive
$0 + xy$	5D. Complements
xy	2. Identity under addition

Proof by Perfect Induction

Step 1

x	y	$(\bar{x} + y)$	$x(\bar{x} + y)$
0	0	1	0
0	1	1	0
1	0	0	0
1	1	1	1

Step 2

x	y	xy
0	0	0
0	1	0
1	0	0
1	1	1

Notice that the last columns of the two tables are identical. This proves (by means of the proof by perfect induction) that the expressions are equivalent

Although used here to prove theorems, the algebraic techniques demonstrated are most practically applied to the simplification of circuit designs, where proof by perfect induction provides little or no help. Use Homework to develop your skills

De Morgan's Theorems

The complement of any Boolean expression, or part of any expression, may be found by means of De Morgan's Theorem. Two steps are used to form a complement.

1) OR symbols are replaced with AND symbols or AND symbols with OR symbols.

2) Each of the variables (terms) in the expression is complemented

Example 1 — Complement of a function

$f = x\bar{y} + z$ Find the complement (NOT) of f

$\bar{f} = \overline{x\bar{y} + z}$

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

Example 2 — Simplify a function

$f = a + b(\overline{c + \bar{d}\bar{e}})$ Simplify

$f = a + b(\overline{c + \bar{d} + \bar{e}})$ First, demorganize $\overline{de}$

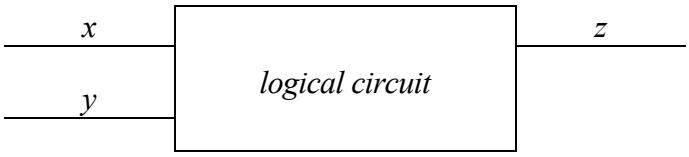
$f = a + b\bar{c}de$

Derivation of Boolean Expressions

When designing a logical circuit, the logic designer works from two sets of known values:

1) the various states which the *inputs* (x, y) to the logical network can take, and

2) the desired *outputs* (z) for each input condition.



These inputs and output conditions may be defined as a detailed set of requirements (**Word Problem**), **Waveforms**, or in tabular form as a **Truth Table**.

The Boolean expression, and by extension the logic circuit, is derived from these sets of values.

Sum-of-Products

An important consideration in dealing with logic circuits and their algebraic counterparts is the form of the Boolean algebra expression and the resulting form of the logic circuit. Certain types of Boolean algebra expressions lead to logic circuits which are more desirable from an implementation viewpoint. We will look at two: Sum-of-products and Product-of-sums. We will consider the most popular form first.

Inputs		**Output**	
x	y	z	*Product Terms*
0	0	1	$\bar{x} \cdot \bar{y}$
0	1	0	$\bar{x} \cdot y$
1	0	1	$x \cdot \bar{y}$
1	1	1	$x \cdot y$

The last column is a list of “product terms” obtained from the values of the input variables. This column contains each of the input variables listed in each row of the table, with the letter representing the respective input complemented when the input value of this variable is 0, and not complemented when the input value is 1. The terms obtained in this manner are designated as product terms or min terms (m_i).

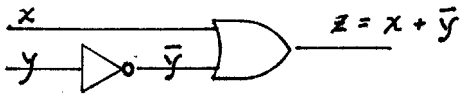
To create a sum-of-products (SOP) expression, add the product terms where the output (*z*) equals 1.

$$z = \bar{x} \cdot \bar{y} + x \cdot \bar{y} + x \cdot y$$

The right hand side of this expression may be simplified as follows:

$z = \bar{x} \cdot \bar{y} + x(\bar{y} + y)$	8. Distributive Law
$z = \bar{x} \cdot \bar{y} + x$	5. Complements
$z = \bar{x} \cdot \bar{y} + x$	5. Complements
$z = x + \bar{y}$	6. Commutative, 10. Degenerate-Reflect

Now that we have the simplest algebraic expression possible, how do we translate it into a logic circuit? To answer this question, you need to take every variable and consider it an input to a gate with every term implemented with a gate. For now, create the compliment of a variable with an inverter. Thus the expression $z = x + \bar{y}$ is realized with a single inverter and an OR gate.



Terminology

Key words: product term, min term, sum-of-products, levels of logic, standard form, canonical form.

Technically, a *product term* contains AND operators and a *min term* is a product term containing all the input variables. The *sum-of-products* expression $z = \bar{x} \cdot \bar{y} + x$, contains one min term and one product term of one variable. A sum-of-product expression has only *two-levels* of logic (AND followed by OR) and is by definition in *standard form*. The expression $z = \bar{x} \cdot \bar{y} + x(\bar{y} + y)$ contains three-levels of logic and is therefore not a sum-of-products. The sum-of-products expression $z = \bar{x} \cdot \bar{y} + x \cdot \bar{y} + x \cdot y$ contains three product terms, which are in turn also min terms. When a sum-of-product expression is expressed as a sum of min terms, it is said to be in *canonical form*.

Steps to Solution:

1) From the problem statement, a truth table is formed. The problem may be in the form of a word problem, waveforms, or tables. In any event, the problem is synthesized into a set of input and corresponding output conditions in tabular form (a truth table).

2) A column is added to the truth table and named *product terms*. For each row whose output is 1, a product term is formed from the input columns.

3) A sum-of-products expression is built from these product terms.

4) The algebraic expression is simplified.

5) A logical circuit is designed.

In this example, you have three inputs (designated *x, y, z*) with two outputs f_1 and f_2.

Steps 1 and 2 — Truth Table and Product Terms

Inputs			Outputs		
x	y	z	f_1	f_2	*Product Terms*
0	0	0	0	0	
0	0	1	0	1	$\bar{x} \cdot \bar{y} \cdot z$
0	1	0	0	0	
0	1	1	0	0	
1	0	0	0	1	$x \cdot \bar{y} \cdot \bar{z}$
1	0	1	0	1	$x \cdot \bar{y} \cdot z$
1	1	0	1	1	$x \cdot y \cdot \bar{z}$
1	1	1	0	1	$x \cdot y \cdot z$

Step 3 — Sum-Of-Products

$$f_1 = xy\bar{z}$$

$$f_2 = \bar{x} \cdot \bar{y} \cdot z + x \cdot \bar{y} \cdot \bar{z} + x \cdot \bar{y} \cdot z + x \cdot \bar{y} \cdot z + x \cdot y \cdot \bar{z} + x \cdot y \cdot z$$

Step 4 — Reduce Algebraic Expression

No further simplification is possible with the first output.

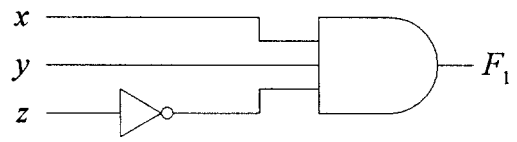
$$f_2 = \bar{x} \cdot \bar{y} \cdot z + x \cdot \bar{y} \cdot \bar{z} + x \cdot \bar{y} \cdot z + x \cdot y \cdot \bar{z} + x \cdot y \cdot z$$

$\bar{x} \cdot \bar{y} \cdot z + x\bar{y}(\bar{z} + z) + xy(\bar{z} + z)$	8. Distributive
$\bar{x} \cdot \bar{y} \cdot z + x\bar{y} \cdot 1 + xy \cdot 1$	5. Complements
$\bar{x} \cdot \bar{y} \cdot z + x\bar{y} + xy$	2D. Identity
$\bar{x} \cdot \bar{y} \cdot z + x(\bar{y} + y)$	8. Distributive
$\bar{x} \cdot \bar{y} \cdot z + x \cdot 1$	5. Complements
$\bar{x} \cdot \bar{y} \cdot z + x$	2D. Identity
$x + \bar{x} \cdot \bar{y} \cdot z$	6. Commutative
$x + \bar{y} \cdot z$	10. Degenerate-Reflect

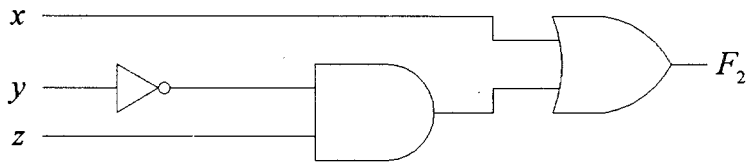
Check Your Answer

Inputs			Outputs	
x	y	z	$\bar{y} \cdot z$	$x + \bar{y} \cdot z$
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	0	1
1	0	1	1	1
1	1	0	0	1
1	1	1	0	1

Step 5 — Convert Algebra to Logic



and



Product-of-Sums

In a fashion analogous to the sum-of-products construction of logic circuits (actually its dual), we can look at a Boolean expression, and its corresponding logic circuit, as the product (AND) of sums (OR).

Inputs			Output
x	y	z	*Sum Terms*
0	0	1	$x + y$
0	1	0	$x + \bar{y}$
1	0	1	$\bar{x} + y$
1	1	1	$\bar{x} + \bar{y}$

The last column is a list of "sum terms" obtained from the values of the input variables. This column contains each of the input variables listed in each row of the table, with the letter representing the respective input complemented when the input value of this variable is 1, and not complemented when the input value is 0. The terms obtained in this manner are designated as sum terms or Max terms (M_i).

To create a product-of-sums (POS) expression, multiply (OR) the sum terms where the output (z) equals 0.

$$z = x + \bar{y}$$

Notice that the Boolean expression, and corresponding logic circuit, derived by the product-of-sums method is the same as developed using the sum-of-products method after simplification.

Terminology

Key words: sum term, Max term, product-of-sums, levels of logic, standard form, canonical form.

Technically, a *sum term* contains OR operators and a *Max term* is a sum term containing all the input variables. The *product-of-sums* expression $z = (x + y) \cdot \bar{x}$, contains one Max term and one sum term of one variable. A product-of-sums expression has only *two-levels* of logic (OR followed by AND) and is by definition in *standard form*. The expression $z = \bar{x} \cdot \bar{y} + x(\bar{y} + y)$ contains three-levels of logic and is therefore not a product-of-sums. The product-of-sums expression $z = (x + y)(\bar{x} + y)(\bar{x} + \bar{y})$ contains three sum terms, which are in turn also Max terms. When a product-of-sums expression is expressed as a sum of Max terms, it is said to be in *canonical form*.

Some texts use the term "conventional" in place of "standard," P_i in place of m_i and S_i in place of M_i.

Steps to Solution:

The "Steps to Solution" using the product-of-sums methodology is the dual of the sum-of-products method.

1) From the problem statement, a truth table is formed. The problem may be in the form of a word problem, waveforms, or tables. In any event, the problem is synthesized into a set of input and corresponding output conditions in tabular form (a truth table).

2) A column is added to the truth table and named *sum terms*. For each row whose output is 0, a sum term is formed from the input columns.

3) A products-of-sums expression is built from these sum terms.

4) The algebraic expression is simplified.

5) A logical circuit is designed.

In this example, you have three inputs (designated *x, y, z*) with two outputs f_1 and f_2.

Steps 1 and 2 — Truth Table and Product Terms

Inputs			Outputs		
x	y	z	f_1	f_2	*Sum Terms*
0	0	0	0	0	$x+y+z$
0	0	1	0	1	$x+y+\bar{z}$
0	1	0	0	0	$x+\bar{y}+z$
0	1	1	0	0	$x+\bar{y}+\bar{z}$
1	0	0	0	1	$\bar{x}+y+z$
1	0	1	0	1	$\bar{x}+y+\bar{z}$
1	1	0	1	1	
1	1	1	0	1	$\bar{x}+\bar{y}+\bar{z}$

Step 3 — Product-Of-Sums

$$f_1 = (x+y+z)(x+y+\bar{z})(x+\bar{y}+z)(x+\bar{y}+\bar{z})(\bar{x}+y+z)(\bar{x}+y+\bar{z})(\bar{x}+\bar{y}+\bar{z})$$

$$f_2 = (x+y+z)(x+\bar{y}+z)(x+\bar{y}+\bar{z})$$

A Few Observations

Just as the first POS example lead directly to the simplest form of the Boolean equation, the SOP derived expression for f_1 (see last Chapter) leads directly to the simplest form of the Boolean equation. The POS method, on the other hand, leads to an overly complex form. The lesson here is to take time before you rush into a solution and study the problem. Often simply inverting the outputs (0 to 1, 1 to 0) and then using De Morgan's Theorem on the solution will lead to the quickest solution. Or as demonstrated here, maybe selecting the POS over the SOP method or visa-versa will provide the quickest and best solution .

Also notice that the logic circuit for a sum-of-products expression in "standard" form (an expression with at least two product terms with at least two variables in each product term) go directly into an AND-to-OR gate circuit, while "standard" product-of-sums expressions go directly into OR-to-AND gate networks.

Step 4 — Reduce Algebraic Expression

$f_2 = (x+y+z)(x+\bar{y}+z)(x+\bar{y}+\bar{z})$

$(x+(y+z)(\bar{y}+z))(x+\bar{y}+\bar{z})$	8D. Dual of Distributive Law
$x+(y+z)(\bar{y}+z)(\bar{y}+\bar{z})$	8D. Dual of Distributive Law
$x+(y+z)(\bar{y}+z\bar{z})$	8D. Dual of Distributive Law
$x+(y+z)(\bar{y}+0)$	5D. Complements
$x+(y+z)\bar{y}$	2. Identity
$x+\bar{y}(y+z)$	6D. Commutative Law of Multiplication
$x+\bar{y}z$	10D. Dual of Degenerate-Reflect

Notice that the Boolean expression, and corresponding logic circuit, derived by the product-of-sums method is the same as developed using the sum-of-products method after simplification.

From Logic Gates to SSI Circuits

Small scale integration (SSI) can be defined as an integrated circuit (IC) with a complexity of 20 or less gates. Medium scale integration (MSI) has a complexity factor of 20 to 100 gates. Above a complexity factor of 100 gates is the domain of Large scale integration (LSI). SSI, MSI, and LSI all use the three fundamental building blocks, no matter if the IC is implemented using RTL, DTL, TTL, ECL, NMOS, or CMOS technology.

Gates	Level of Integration
1 to 19	SSI
20 to 100	MSI
100 plus	LSI

The most popular commercial SSI and MSI logic family is known as the seventy-four hundred (7400) series. It's military counterpart is known as the fifty-four hundred series (5400). The 5400 series of parts have operating specifications for military applications, whereas the 7400 series have operating specifications suited for commercial applications. Thus, for example, a 5408 and 7408 (AND gate package) differ only in their operating spec's, not functionally. The part number of an IC which is a member of the seventy-four hundred series begins with 74 (what did you think) followed by one or more letters which denote the technology employed. Popular technologies include:

Logic Family	Technology	7400 Designator	Comments
TTL Transistor-transistor logic	Schottky Low-power Schottky Advanced Low-Power Schottky Fast	S LS ALS F	High Speed
CMOS Complementary metal-oxide semiconductor	High-speed CMOS High-speed CMOS with TTL compatibility	HC HCT	Low Power
ECL	Emitter-coupled logic		Very High Speed
MOS	Metal-oxide semiconductor		High Density

The above table provides only a partial listing. The absence of a letter indicates standard TTL logic (can you say obsolete). In many cases the 74 is preceded by letters to indicate the manufacturer. For example SN denotes this as a seventy-four hundred series part made by Texas Instruments. The next series of numbers indicate the part type (AND, OR, etc.). Depending on the manufacturer additional letters may be appended to denote version (A or B), the package code (N = Plastic DIP), temperature range, and special processing (commercial grade device with burn-in).

	Manufacturer	Family	Technology	Type	Package and Temperature
Part Number	SN	74	LS	08	N

Examples of Logic gates packaged into plastic 14 pin dual-inline packages (DIPs). See page 65 of the textbook for additional examples.

7408 AND Gates	7406 Inverters