Binary Logic and Gates

Introduction

- Our objective is to learn how to design ***digital*** circuits.
- These circuits use ***binary*** systems.
- **Signals** in such binary systems may represent only one of ***2 possible values*** *0 or 1*
- *Physically*, these signals are electrical *voltage* signals
- **These signals may assume either a high** or a **Low** voltage value.
- The high voltage value typically equals the voltage of the power supply **(**e.g. 5 volts or 3.3 volts), and the **Low** voltage value is typically 0 volts (or Ground).
- When a signal is at the **High** voltage value, we say that the signal has a *Logic 1* value.
- When a signal is at the **Low** voltage value, we say that the signal has a *Logic 0* value.
- Hence, the *physical* value of a signal is the actual voltage value it carries, while its *Logic* value is either **1** (**High**) or **0** (**Low**).
- Digital circuits process (or manipulate) input binary signals and produce the required output binary signals as shown in **Figure 1**

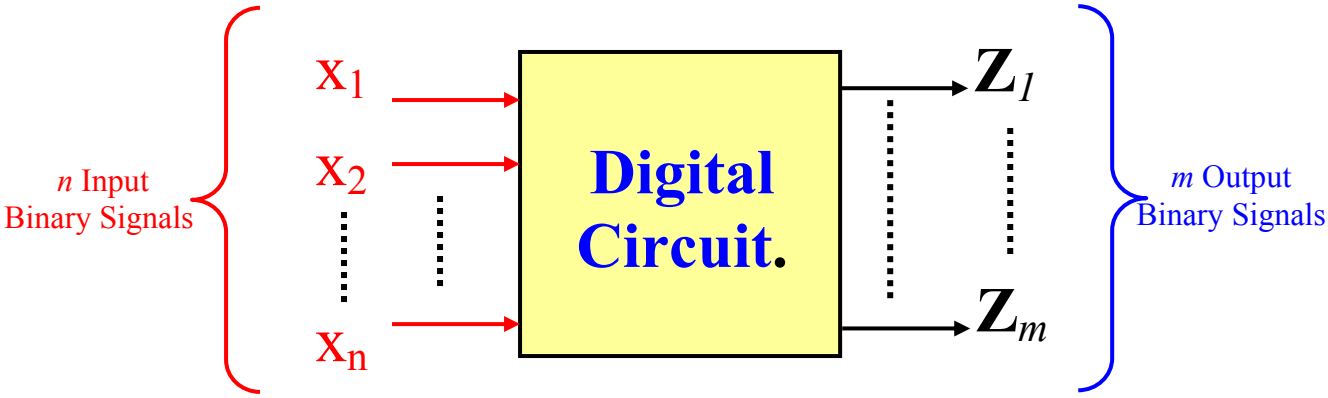


Figure 1 A Digital Circuit with *n* Input Signals and *m* Output Signals

- Generally, the circuit will have a number of input signals (say *n* of them) as shown in the Figure x_1, x_2, up to x_n, and a number of output signals (say m) Z_1, Z_2, up to Z_m.
- The value assumed by the i^{th} output signal Z_i depends on the values of the input signals x_1, x_2, up to x_n.
- In other words, we can say that Z_i is a function of the n input signals x_1, x_2, up to x_n. *Or we can write:*

$$Z_i = F_i(x_1, x_2, \ldots\ldots, x_n) \qquad \textit{for } i = 1, 2, 3, \ldots m$$

- The m output functions (F_i) are functions of binary signals and produce a single binary output signal.
- Thus, these functions are binary functions and require binary logic algebra for their derivation and manipulation. This binary system algebra is commonly referred to as ***Boolean Algebra*** after the mathematician George Boole. The functions are known as *Boolean functions* while the binary signals are represented by *Boolean variables*.
- To be able to design a digital circuit, we must learn how to *derive* the Boolean function implemented by this circuit.

Notes:

1. The two values of binary variables may be equivalently referred to as **0** and **1** or **False** (0) and **True** (1)or as **Low** (0) and **High**(1).
2. Whether we use 0 and 1 or False and True or Low and High, all these are referred to as Logic Values.
3. Systems manipulating Binary Logic Signals are commonly referred to as **Binary Logic systems.**
4. Digital circuits implementing a particular Binary (Boolean) function are commonly known as **Logic Circuits**.

CHAPTER OBJECTIVES

- **Learn** ***Binary Logic*** **and** ***BOOLEAN Algebra***
- **Learn How to Map a** ***Boolean Expressions*** **into Logic** ***Circuit Implementations***
- **Learn How To Manipulate** ***Boolean Expressions*** **and** ***Simplify Them***

Elements of Boolean Algebra (Binary Logic)

As in standard algebra, Boolean algebra has 3 main elements:

1. Constants,
2. Variables, and
3. Operators.

Logically

- ***Constant*** *Values* are either 0 or 1 *Binary* ***Variables*** $\in \{0, 1\}$
- *3 Possible* ***Operators*** The **AND operator**, the **OR** operator, and the **NOT** operator

Physically

- **Constants** $\Rightarrow$ Power Supply Voltage (Logic 1)
  $\Rightarrow$ Ground Voltage (Logic 0)
- **Variables** $\Rightarrow$ Signals (**H**igh = **1**, **L**ow = **0**)
- **Operators** $\Rightarrow$ Electronic Devices (**Logic Gates**)
  1. • **AND** - Gate
  2. • **OR** - Gate
  3. • **NOT** - Gate (***Inverter***)

Logic Gates & Logic Operations

The AND Operation

- If ***X*** and ***Y*** are two *binary variables*, the result of the operation ***X*** **AND** ***Y*** is **1** *if and only if* **both** ***X*** **= 1 and** ***Y*** **= 1**, and is **0 otherwise**.
- In Boolean expressions, the **AND** operation is represented either by a "*dot*" or by the absence of an operator. Thus, ***X*** **AND** ***Y*** is written as ***X.Y*** or just ***XY***
- This is summarized in the following table (commonly called *truth table*):

Table 1 Truth Table of the AND operation

X	*Y*	*Z = X AND Y* *Z=XY*
F	F	F
F	T	F
T	F	F
T	T	T

Table 1 Truth Table of the AND operation

X	*Y*	*Z = X AND Y* *Z=XY*
0	0	0
0	1	0
1	0	0
1	1	1

- The electronic device which performs the AND operation is called the **AND gate**. Figure 2 shows the symbol of a 2-input AND gate which has two inputs (***X*** and ***Y***) and gives one output ***Z=XY***

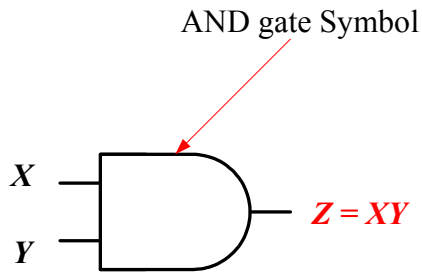


Figure 2 Two-Input AND gate

- The AND logic can be further illustrated using what is known as the Venn diagram
- AND gates may have more than 2 inputs. Figure 3 shows a 3-input AND gate.

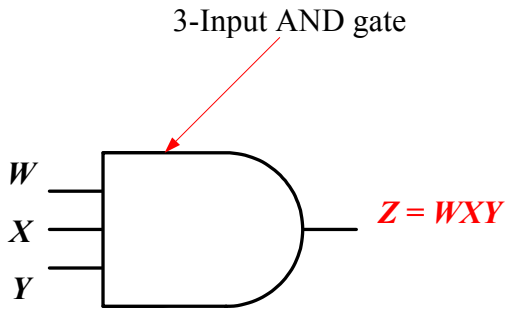


Figure 3 Three-Input AND gate

- The truth table of the output variable Z=WXY of the 3-input AND gate is given in Table 2

Table 2 Truth Table of 3-Input AND gate

W	*X*	*Y*	*Z=WXY*
0	**0**	**0**	**0**
0	**0**	**1**	**0**
0	**1**	**0**	**0**
0	**1**	**1**	**0**
1	**0**	**0**	**0**
1	**0**	**1**	**0**
1	**1**	**0**	**0**
1	**1**	**1**	**1**

Notes

- The output of an AND gate is 1 *if and only if* **ALL** its input signals are 1's, otherwise it is 0.
- A function of two input binary variables will have a truth table of 4 rows since each variable may assume any one of two possible values (0 or 1).
- A function of three input variables will have a truth table of 8 rows since each variable may assume any one of two possible values (0 or 1).
- In general, ***n*** input variables have 2^n possible combinations. Accordingly, a function of ***n*** input variables, will have a truth table of 2^n rows.

The OR Operation

- If ***X*** and ***Y*** are two *binary variables*, the result of the operation ***X* OR *Y*** is **1** *if and only if* **either *X* = 1 or *Y* = 1 or both X & Y are 1's**, but it is **0 otherwise**.
- In other words, ***X* OR *Y*** is 0 if and only if both ***X* = 0 and *Y* = 0**, but is **1 otherwise**.
- In Boolean expressions, the **OR** operation is represented by a "*plus*" sign. Thus, ***X* OR *Y*** is written as ***X*+*Y***
- This is summarized in the Table 3.

Table 3 Truth Table of the OR operation

X	*Y*	*Z = X OR Y* *Z=X+Y*
0	**0**	**0**
0	**1**	**1**
1	**0**	**1**
1	**1**	**1**

- The electronic device which performs the OR operation is called the **OR gate**. Figure 4 shows the symbol of a 2-input OR gate which has two inputs (***X*** and ***Y***) and gives one output ***Z=X+Y***

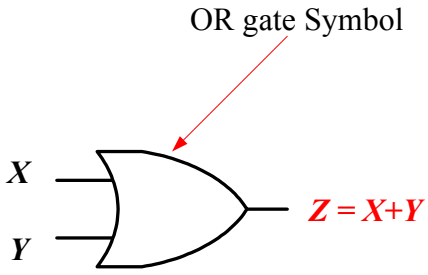


Figure 4 Two-Input OR gate

- The OR logic can be further illustrated using the Venn diagram
- OR gates may have more than 2 inputs. Figure 5 shows a 3-input OR gate.

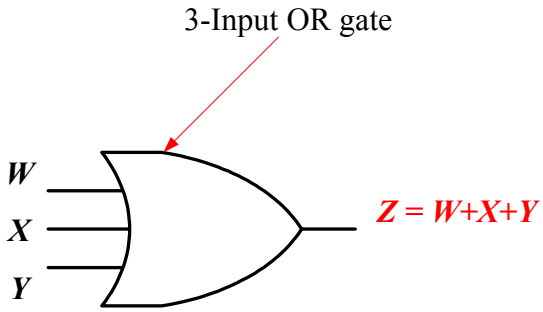


Figure 5 Three-Input OR gate

- The truth table of a 3 input OR gate Z=W+X+Y is given in **Table** 4

Table 4 Truth Table of

3-Input OR gate

W	*X*	*Y*	*Z=WXY*
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

- In general, the output of an OR gate is 1 *unless* **ALL** its input signals are 0's.

The NOT Operation

- NOT is a "*unary*" operator.
- IF ***Z*** = **NOT** ***X***, then the value of ***Z*** will always be the complement of the value of ***X***. In other words, if ***X*** = 0 then ***Z*** = 1, and if ***X*** = 1 then ***Z*** =0.
- In Boolean expressions, the **NOT** operation is represented by either a bar on top of the variable (e.g. $Z = \overline{X}$) or a prime (e.g. $Z = X'$).
- This is summarized in Table 5.

Table 5 Truth Table of the

NOT operation

X	*Z=X'*
0	1
1	0

- The electronic device which performs the NOT operation is called the **NOT gate, or simply** ***INVERTER***. Figure 5 shows the inverter symbol.

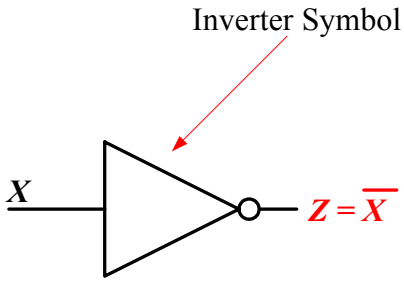


Figure 5 An Inverter

- If $Z = \overline{X}$, Z is commonly referred to as the *Complement* of X. Alternatively, we say that *Z equals X-complemented*
- The NOT operation can be further illustrated using the Venn diagram

Boolean Algebra

Logic Circuits and Boolean Expressions

- A Boolean expression (or a Boolean function) is a combination of Boolean *variables*, *AND*-operators, *OR*-operators, and *NOT* operators.
- • *Boolean Expressions* (*Functions*) are fully defined by their *truth tables*Each Boolean function (expression) can be implemented by a digital *logic circuit* which consists of logic gates.
    - Variables of the function correspond to signals in the logic circuit,
    - Operators of the function are converted into corresponding logic gates in the logic circuit.

Example

Consider the expression $F = X + (\overline{Y}.Z)$ The diagram of the logic circuit corresponding to this function is shown in Figure 6

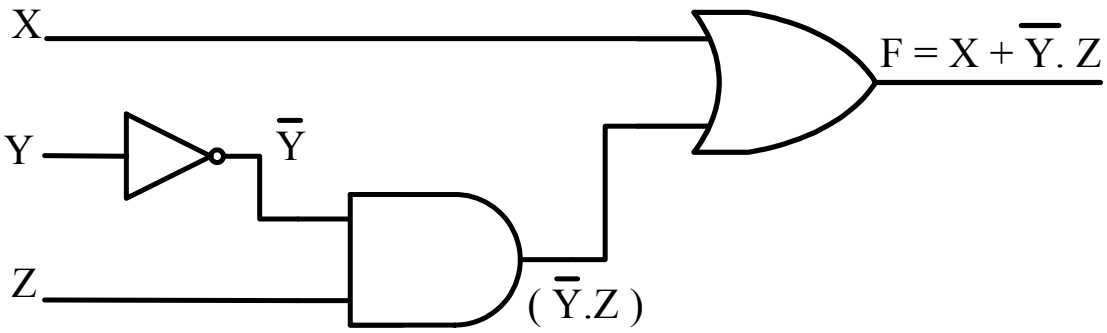


Figure 6 Logic Circuit Diagram of $F = X + (\overline{Y}.Z)$

The truth table of this function is shown in Table 6

Table .6 Truth Table of $F = X + (\overline{Y}.Z)$

X	Y	Z	Y`	Y`Z	F= X + Y`Z
0	0	0	1	0	0
0	0	1	1	1	1
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	1	0	1
1	0	1	1	1	1
1	1	0	0	0	1
1	1	1	0	0	1

- Since F is function of 3 variables (X, Y, Z), the truth table has 2^3 or 8 rows.

Basic Identities of Boolean Algebra

AND Identities

From the truth table of the AND operation, shown here for reference, we can derive some basic identities. These identities can be easily verified by showing that they are valid for both possible values of X (0 and 1).

AND Truth Table		
X	***Y***	***Z=XY***
0	**0**	**0**
0	**1**	**0**
1	**0**	**0**
1	**1**	**1**

1. $0 \cdot X = 0$

AND Truth Table		
X	***Y***	***Z=XY***
0	**0**	**0**
0	**1**	**0**
1	**0**	**0**
1	**1**	**1**

2. $1 \cdot X = X$

AND Truth Table		
X	***Y***	***Z=XY***
0	**0**	**0**
0	**1**	**0**
1	**0**	**0**
1	**1**	**1**

3. $X \cdot X = X$

AND Truth Table		
X	***Y***	***Z=XY***
0	**0**	**0**
0	**1**	**0**
1	**0**	**0**
1	**1**	**1**

4. $X.\overline{X} = 0$

AND Truth Table		
X	***Y***	***Z=XY***
0	**0**	**0**
0	**1**	**0**
1	**0**	**0**
1	**1**	**1**

OR Identities

From the truth table of the OR operation, shown here for reference, we can derive some basic identities. These identities can be easily verified by showing that they are valid for both possible values of X (0 and 1).

1. $1 + X = 1$

OR Truth Table		
X	***Y***	***Z=X+Y***
0	**0**	**0**
0	**1**	**1**
1	**0**	**1**
1	**1**	**1**

2. $0 + X = X$

OR Truth Table		
X	***Y***	***Z=X+Y***
0	**0**	**0**
0	**1**	**1**
1	**0**	**1**
1	**1**	**1**

3. $X + X = X$

OR Truth Table		
X	***Y***	***Z=X+Y***
0	**0**	**0**
0	**1**	**1**
1	**0**	**1**
1	**1**	**1**

4. $X + \overline{X} = 1$

OR Truth Table		
X	***Y***	***Z=X+Y***
0	**0**	**0**
0	**1**	**1**
1	**0**	**1**
1	**1**	**1**

Summary of the basic identity

AND Identities

1. $0 . X = 0$
2. $1 . X = X$
3. $X . X = X$
4. $X . \overline{X} = 0$

OR Identities

5. $1 + X = 1$
6. $0 + X = X$
7. $X + X = X$
8. $X + \overline{X} = 1$

Duality Principle

- Given a Boolean expression, its *dual* is obtained by replacing each 1 with a 0, each 0 with a 1, each AND (.) with an OR (+), and each OR (+) with an AND(.).
- The dual of an identity is also an identity. This is known as the duality principle.

It can be easily shown that the AND basic identities and the OR basic identities are duals as shown in Table 7

Table 7 Duality of the AND and OR Basic Identities

AND Identities		Dual Identities (OR Udentities)
0 . X = 0	**0 → 1** **. → +**	**1 + X = 1**
1 . X = X	**1 → 0** **. → +**	**0 + X = X**
X . X = X	**. → +**	**X + X = X**
X . $\overline{X}$ = 0	**0 → 1** **. → +**	**X + $\overline{X}$ = 1**

Another Important Identity

$$\left(\overline{\overline{X}}\right) = X$$

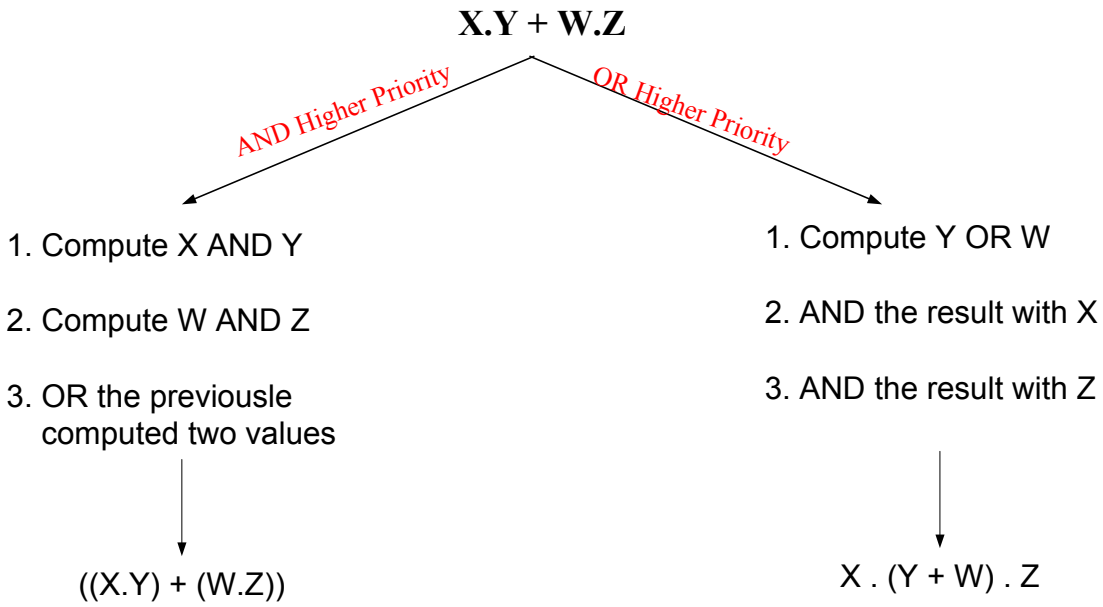
- This can be simply proven from the truth table of the NOT operation as shown.

NOT operation Truth Table

X	$\overline{X}$	$\left(\overline{\overline{X}}\right)$
0	**1**	**0**
1	**0**	**1**

Operator Precedence

Given the Boolean expression **X.Y + W.Z** the order of applying the operators will affect the final value of the expression.



For Boolean Algebra, the precedence rules for various operators are given below , in a decreasing order of priority:

1- *Parentheses* → *Highest Priority*

2- *Not* operator (*Complement*)

3- *AND* operator,

4- *OR* operator → *Lowest Priority*

Properties of Boolean Algebra

Important properties of Boolean Algebra are shown in Table

		Property	Dual Property
1	Commutative	X + Y = Y + X	X . Y = Y . X
2	Distributive	X.(Y + Z) = X.Y + X.Z	X+(Y.Z) = (X+Y).(X+Z)
3	DeMorgan	(X + Y) ` = X`.Y`	(X.Y) ` = X` + Y`
4	Extended DeMorgan	(A+B+C+ ….+Z)` = A`.B`.C`…..Z`	(A.B.C….Z)` = A`+B`+C`+….+Z`
5	Generalized DeMorgan	$[F(x_1,x_2,\ldots,x_n,0,1,+,.)]` = F(x`_1,x`_2,\ldots,x`_n,1,0,.,+)$	

Notes

- The above properties can be easily proved using truth tables.
- The only difference between the *dual* of an expression and the *complement* of that expression is that in the dual *variables* are not complemented while in the complement expression, all variables are complemented.
- Using the above properties, complex *Boolean expressions* can be *manipulated* into a *simpler forms* resulting in simpler logic circuit implementations.
- Simpler expressions are generally implemented by simpler logic circuits which are both faster and less expensive. This represents a great advantage since *cost* and *speed* are prime factors in the success and profitability of any product.

Algebraic Manipulation

- The objective here is to acquire some skills in manipulating Boolean expressions into simpler forms for more efficient implementations.
- Properties of Boolean algebra will be utilized for this purpose.

Example Prove that $X + XY = X$

Proof: $X + XY = X.(1 + Y) = X.1 = X$

Example Prove that $X + \overbrace{\quad}^{=1} = \overbrace{\quad}^{=X}$ → *This an important identity that is useful in simplifying more complex expressions*

Proof: This will be proved in two ways

(1) $X + X`Y = \overbrace{(X + X`)}^{=1}(X + Y)$

$= 1.(X + Y)$

$= X + Y$

(2) $X + X`Y = X.1 + X`Y =$

$= X.\overbrace{(1+Y)} + X`Y$

$= \overbrace{X + XY} + X`Y$

$= X + (XY + X`Y)$

$= X + Y(\overbrace{X + X`}^{=1})$

$= X + Y$

Example ``Consensus Theory``

Show that $XY + X`Z + YZ = XY + X`Z$

Proof:

LHS $= XY + X`Z + YZ$

$= XY + X`Z + YZ.1$ (1 = 1)

$= XY + X`Z + YZ.(X + X`)$

$= XY + X`Z + YZX + YZX`$

$= XY + YZX + X`Z + YZX`$

$= XY(1 + Z) + X`Z(1 + Y)$ (1 = 1, 1 = 1)

$= XY.1 + X`Z.1 = XY + X`Z =$ **LHS**

Example

Simplify the following function

$$F_1 = \overline{(A + \overline{B} + A\overline{B})}\ \ \overline{(AB + \overline{A}C + BC)}$$

Solution:

$$\mathbf{F1} = \overline{(A + \overline{B} + A\overline{B})}\ \ \overline{(AB + \overline{A}C + BC)}$$

Using De-Morgan theorem

- $\overline{(A + \overline{B} + A\overline{B})} = A` . B. (A` + B) = A` . B + A` . B = A` . B$
- $\overline{(AB + \overline{A}C + BC)} = (A` + B`).(A + C` ).(B` + C`)$

$$\mathbf{F1} = \overline{(A + \overline{B} + A\overline{B})}\ \ \overline{(AB + \overline{A}C + BC)}$$

$$= A` . B. (A` + B`).(A + C`).(B` + C`)$$

Since X= X.X=X.X.X, we can rewrite the previous expression as follows

$$\mathbf{F1} = (A`.B). (A` . B). (A` . B). (A` + B).(A + C` ).(B` + C`)$$

$$= (A`.B). (A` + B`). (A` . B) .(A + C` ). (A` . B). (B` + C`)$$

$$= (A`.B + 0 ) . (0 + A` . B.C` ) . (A`.B + A`.B.C`)$$

$$= (A`.B) . (A` . B.C` ) . (A`.B)$$

$$= A` . B.C`$$

Example

Simplify the following function

i. $\mathbf{G} = \overline{\left(\overline{(A + \overline{B} + C)} . (\overline{AB} + \overline{C}\,\overline{D}) + \overline{ACD} \right)}$

Solution:

$$G = \overline{\left(\overline{(A + \overline{B} + C)} . (\overline{AB} + \overline{C}\,\overline{D}) + \overline{ACD} \right)}$$

$$= \left((A + \overline{B} + C) + \left(AB . (C + D) \right) \right) . ACD$$

$$= (A + \overline{B} + C) . ACD + \left(AB . (C + D) \right) . ACD$$

$$= (ACD + ACD\overline{B}) + (ACDB + ACDB)$$

$$= ACD + ACDB$$

$$= ACD$$