Week No. 03
Digital Logic Design

Nouman M Durrani

Parity Method for Error Detection

A parity bit tells if the number of 1s is odd or even.

Many systems use a parity bit as a means for bit **error detection.** Any group of bits contain either an even or an odd number of 1s. A parity bit is attached to a group of bits to make the total number of 1s in a group always even or always odd. An even parity bit makes the total number of 1s even, and an odd parity bit makes the total odd.

TABLE 2–10

The BCD code with parity bits.

EVEN PARITY		ODD PARITY	
P	BCD	P	BCD
0	0000	1	0000
1	0001	0	0001
1	0010	0	0010
0	0011	1	0011
1	0100	0	0100
0	0101	1	0101
0	0110	1	0110
1	0111	0	0111
1	1000	0	1000
0	1001	1	1001

Detecting an Error

A parity bit provides for the detection of a single bit error but cannot check for two errors in one group.

An odd parity system receives the following code groups: 10110, 11010, 110011, 110101110100, and 1100010101010. Determine which groups, if any, are in error.

Solution Since odd parity is required, any group with an even number of 1s is incorrect. The following groups are in error: **110011** and **1100010101010.**

Assign the proper even parity bit to the following code groups:

(a) 1010 **(b)** 111000 **(c)** 101101

(d) 1000111001001 **(e)** 101101011111

Solution Make the parity bit either 1 or 0 as necessary to make the total number of 1s even. The parity bit will be the left-most bit (color).

(a) 01010 **(b)** 1111000 **(c)** 0101101

(d) 0100011100101 **(e)** 1101101011111

The Hamming Error Correction Code

- A single parity bit allows for the detection of single-bit errors in a code word.
- A single parity bit can indicate that there is an error in a certain group of bits.
- In order to correct a detected error, more information is required because the position of the bit in error must be identified before it can be corrected.
- More than one parity bit must be included in a group of bits to be able to correct a detected error.
- The Hamming code provides for single-error correction.

Number of Parity Bits If the number of data bits is designated d, then the number of parity bits, p, is determined by the following relationship:

$$2^p \geq d + p + 1 \qquad \text{Equation 2–1}$$

For example, if we have four data bits, then p is found by trial and error with Equation 2–1. Let $p = 2$. Then

$$2^p = 2^2 = 4$$

and

$$d + p + 1 = 4 + 2 + 1 = 7$$

Since 2^p must be equal to or greater than $d + p + 1$, the relationship in Equation 2–1 is *not* satisfied. We have to try again. Let $p = 3$. Then

$$2^p = 2^3 = 8$$

and

$$d + p + 1 = 4 + 3 + 1 = 8$$

- The parity bits are located in the positions that are numbered corresponding to ascending powers of two (1, 2, 4, 8, . . .), as indicated;
  P_1, P_2, D_1, P_3, D_2, D_3 , D_4
- The symbol P_n designates a particular parity bit, and D_n designates a particular data bit.

Determine the Hamming code for the BCD number 1001 (data bits), using even parity.

Solution **Step 1:** Find the number of parity bits required. Let $p = 3$. Then

$$2^p = 2^3 = 8$$
$$d + p + 1 = 4 + 3 + 1 = 8$$

Three parity bits are sufficient.

$$\text{Total code bits} = 4 + 3 = 7$$

Step 2: Construct a bit position table, as shown in Table 2–12, and enter the data bits. Parity bits are determined in the following steps.

▼ TABLE 2–12

BIT DESIGNATION	P_1	P_2	D_1	P_3	D_2	D_3	D_4
BIT POSITION	1	2	3	4	5	6	7
BINARY POSITION NUMBER	001	010	011	100	101	110	111
Data bits			1		0	0	1
Parity bits	0	0		1			

Step 3: Determine the parity bits as follows:

Bit P_1 checks bit positions 1, 3, 5, and 7 and must be a 0 for there to be an even number of 1s (2) in this group.

Bit P_2 checks bit positions 2, 3, 6, and 7 and must be a 0 for there to be an even number of 1s (2) in this group.

Bit P_3 checks bit positions 4, 5, 6, and 7 and must be a 1 for there to be an even number of 1s (2) in this group.

Step 4: These parity bits are entered in Table 2–12, and the resulting combined code is 0011001.

Determine the Hamming code for the data bits 10110 using odd parity.

Solution **Step 1:** Determine the number of parity bits required. In this case the number of data bits, d, is five. From the previous example we know that $p = 3$ will not work. Try $p = 4$:

$$2^p = 2^4 = 16$$
$$d + p + 1 = 5 + 4 + 1 = 10$$

Four parity bits are sufficient.

$$\text{Total code bits} = 5 + 4 = 9$$

Step 2: Construct a bit position table, Table 2–13, and enter the data bits. Parity bits are determined in the following steps. Notice that P_4 is in bit position 8.

▼ TABLE 2–13

BIT DESIGNATION	P_1	P_2	D_1	P_3	D_2	D_3	D_4	P_4	D_5
BIT POSITION	1	2	3	4	5	6	7	8	9
BINARY POSITION NUMBER	0001	0010	0011	0100	0101	0110	0111	1000	1001
Data bits			1		0	1	1		0
Parity bits	1	0		1				1	

Step 3: Determine the parity bits as follows:

Bit P_1 checks bit positions 1, 3, 5, 7, and 9 and must be a 1 for there to be an odd number of 1s (3) in this group.

Bit P_2 checks bit positions 2, 3, 6, and 7 and must be a 0 for there to be an odd number of 1s (3) in this group.

Bit P_3 checks bit positions 4, 5, 6, and 7 and must be a 1 for there to be an odd number of 1s (3) in this group.

Bit P_4 checks bit positions 8 and 9 and must be a 1 for there to be an odd number of 1s (1) in this group.

Step 4: These parity bits are entered in the Table 2–13, and the resulting combined code is 101101110.

EXAMPLE 2–43

Assume that the code word in Example 2–41 (0011001) is transmitted and that 0010001 is received. The receiver does not "know" what was transmitted and must look for proper parities to determine if the code is correct. Designate any error that has occurred in transmission if even parity is used.

Solution First, make a bit position table, as indicated in Table 2–14.

TABLE 2–14

BIT DESIGNATION	P_1	P_2	D_1	P_3	D_2	D_3	D_4
BIT POSITION	1	2	3	4	5	6	7
BINARY POSITION NUMBER	001	010	011	100	101	110	111
Received code	0	0	1	0	0	0	1

First parity check:
Bit P_1 checks positions 1, 3, 5, and 7.
There are two 1s in this group.
Parity check is good. ⟶ 0 (LSB)

Second parity check:
Bit P_2 checks positions 2, 3, 6, and 7.
There are two 1s in this group.
Parity check is good. ⟶ 0

Third parity check:
Bit P_3 checks positions 4, 5, 6, and 7.
There is one 1 in this group.
Parity check is bad. ⟶ 1 (MSB)

Result:
The error position code is 100 (binary four). This says that the bit in position 4 is in error. It is a 0 and should be a 1. The corrected code is 0011001, which agrees with the transmitted code.

Related Problem Repeat the process illustrated in the example if the received code is 0111001.

BOOLEAN ALGEBRA

- Boolean algebra is the mathematics of digital systems. A basic knowledge of Boolean algebra is indispensable to the study and analysis of logic circuits.
- A variable is a symbol used to represent a logical quantity. Any single variable can have a 1 or a 0 value.
- The complement is the inverse of a variable and is indicated by a bar over the variable (overbar). For example,

$$\text{If } A = 1, \text{ then } \overline{A} = 0. \text{ If } A = 0, \text{ then } \overline{A} = 1.$$

- A literal is a variable or the complement of a variable.

Laws of Boolean Algebra

- Commutative Laws The commutative law of addition for two variables is written as

  A+B=B+A

- This law states that the order in which the variables are ORed makes no difference.

FIGURE 4–1

Application of commutative law of addition.

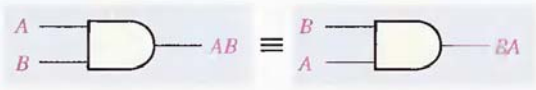
The *commutative law of multiplication* for two variables is

$$AB = BA$$

This law states that the order in which the variables are ANDed makes no difference. Figure 4–2 illustrates this law as applied to the AND gate.

FIGURE 4–2

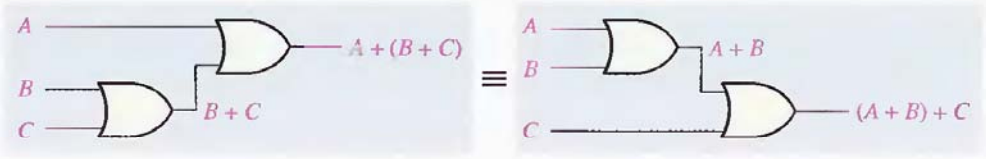
Application of commutative law of multiplication.



Associative Laws The *associative law of addition* is written as follows for three variables:

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

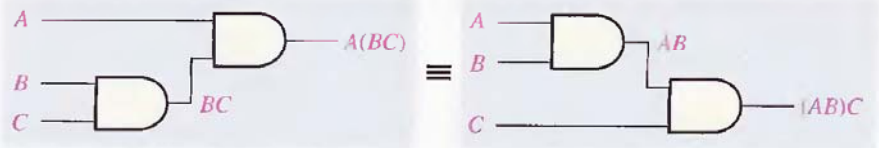
This law states that when ORing more than two variables, the result is the same regardless of the grouping of the variables. Figure 4–3 illustrates this law as applied to 2-input OR gates.



The *associative law of multiplication* is written as follows for three variables:

$$A(BC) = (AB)C$$

This law states that it makes no difference in what order the variables are grouped when ANDing more than two variables. Figure 4–4 illustrates this law as applied to 2-input AND gates.



Distributive Law The distributive law is written for three variables as follows:

$$A(B + C) = AB + AC$$

This law states that ORing two or more variables and then ANDing the result with a single variable is equivalent to ANDing the single variable with each of the two or more varibles and then ORing the products:

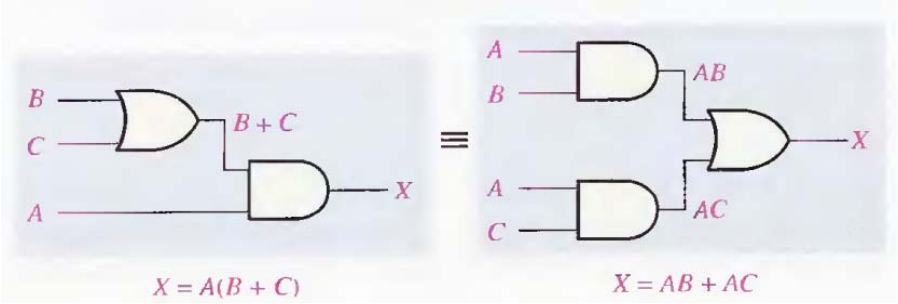


TABLE 4–1

Basic rules of Boolean algebra.

1. $A + 0 = A$	**7.** $A \cdot A = A$
2. $A + 1 = 1$	**8.** $A \cdot \overline{A} = 0$
3. $A \cdot 0 = 0$	**9.** $\overline{\overline{A}} = A$
4. $A \cdot 1 = A$	**10.** $A + AB = A$
5. $A + A = A$	**11.** $A + \overline{A}B = A + B$
6. $A + \overline{A} = 1$	**12.** $(A + B)(A + C) = A + BC$

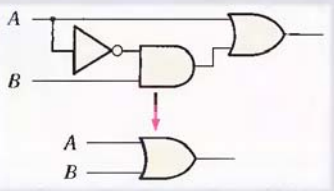
A, B, or *C* can represent a single variable or a combination of variables.

Rule 11. $A + \overline{A}B = A + B$ This rule can be proved as follows:

$A + \overline{A}B = (A + AB) + \overline{A}B$	Rule 10: $A = A + AB$
$= (AA + AB) + \overline{A}B$	Rule 7: $A = AA$
$= AA + AB + A\overline{A} + \overline{A}B$	Rule 8: adding $A\overline{A} = 0$
$= (A + \overline{A})(A + B)$	Factoring
$= 1 \cdot (A + B)$	Rule 6: $A + \overline{A} = 1$
$= A + B$	Rule 4: drop the 1

A	B	$\overline{A}B$	$A + \overline{A}B$	$A + B$
0	0	0	0	0
0	1	1	1	1
1	0	0	1	1
1	1	0	1	1

equal

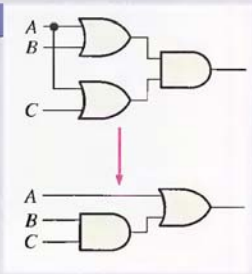


Rule 12. $(A + B)(A + C) = A + BC$ This rule can be proved as follows:

$(A + B)(A + C)$	$= AA + AC + AB + BC$	Distributive law
	$= A + AC + AB + BC$	Rule 7: $AA = A$
	$= A(1 + C) + AB + BC$	Factoring (distributive law)
	$= A \cdot 1 + AB + BC$	Rule 2: $1 + C = 1$
	$= A(1 + B) + BC$	Factoring (distributive law)
	$= A \cdot 1 + BC$	Rule 2: $1 + B = 1$

A	B	C	A + B	A + C	(A + B)(A + C)	BC	A + BC
0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	1	1	1	1
1	0	0	1	1	1	0	1
1	0	1	1	1	1	0	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

equal



DEMORGAN'S THEOREMS

- DeMorgan's theorems provide mathematical verification of the equivalency of the NAND and negative-OR gates and the equivalency of the NOR and negative-AND gates.

The complement of two or more ANDed variables is equivalent to the OR of the complements of the individual variables.

The formula for expressing this theorem for two variables is

$$\overline{XY} = \overline{X} + \overline{Y}$$ Equation 4–6

DeMorgan's second theorem is stated as follows:

The complement of a sum of variables is equal to the product of the complements of the variables.

Stated another way,

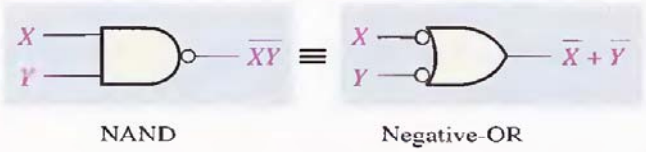
The complement of two or more ORed variables is equivalent to the AND of the complements of the individual variables.

The formula for expressing this theorem for two variables is

$$\overline{X + Y} = \overline{X}\,\overline{Y}$$ Equation 4–7

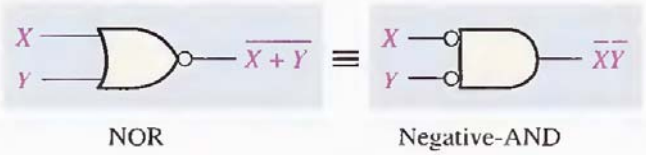
Figure 4–15 shows the gate equivalencies and truth tables for Equations 4–6 and 4–7.

Figure 4–15 shows the gate equivalencies and truth tables for Equations 4–6 and 4–7.



NAND ≡ Negative-OR

Inputs		Output	
X	Y	$\overline{XY}$	$\overline{X} + \overline{Y}$
0	0	1	1
0	1	1	1
1	0	1	1
1	1	0	0



NOR ≡ Negative-AND

Inputs		Output	
X	Y	$\overline{X + Y}$	$\overline{\overline{X}\,\overline{Y}}$
0	0	1	1
0	1	0	0
1	0	0	0
1	1	0	0