MUTHAYAMMAL ENGINEERING COLLEGE

Rasipuram - 637 408

COURSE CODE & TITLE - 19GES24 /DIGITAL PRINCIPLES AND SYSTEM DESIGN

UNIT-1

BOOLEAN ALGEBRA AND LOGIC GATES

Presentation by
Mrs.V.Hema
AP-ECE

SYLLABUS

- **UNIT I: BOOLEAN ALGEBRA AND LOGIC GATES**

  Review of Number Systems –Arithmetic Operations -Binary Codes–Boolean Algebra and Theorems –Boolean Functions–Simplification of Boolean Functions using Karnaugh Map and Tabulation Methods –Logic Gates–NAND and NOR Implementations.

- **UNIT II :COMBINATIONAL LOGIC**

  Combinational Circuits –Analysis and Design Procedures–Circuits for Arithmetic Operations, Code Conversion –Decoders and Encoders –Multiplexers and Demultiplexers –Introduction to HDL –HDL Models of Combinational circuits.

- **UNIT III:SYNCHRONOUS SEQUENTIAL LOGIC**

  Sequential Circuits –Latches and Flip Flops –Analysis and Design Procedures –State Reduction and State Assignment –Shift Registers–Counters –HDL for Sequential Logic Circuits.

SYLLABUS

- **UNIT IV:ASYNCHRONOUS SEQUENTIAL LOGIC**

  Analysis and Design of Asynchronous Sequential Circuits–Reduction of State and Flow Tables –Race-free State Assignment–Hazards.

- **UNIT V:MEMORY AND PROGRAMMABLE LOGIC**

  RAM and ROM –Memory Decoding –Error Detection and Correction –Programmable Logic Array –Programmable Array Logic –Sequential Programmable Devices –Application Specific Integrated Circuits.

- **TEXT BOOKS:**

  1.“Digital Design”, Pearson Education Publication, IV Edition 2008 by Morris Mano M. and Michael D. Ciletti

  2. “Digital Design Principles and Practices”, Pearson Education Publication, IV Edition 2008 by John F. Wakerly

INTRODUCTION

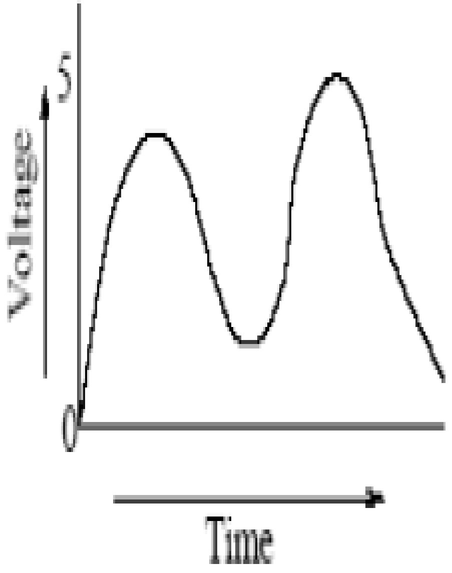
Basically there are two types of signals in electronics,

i) Analog

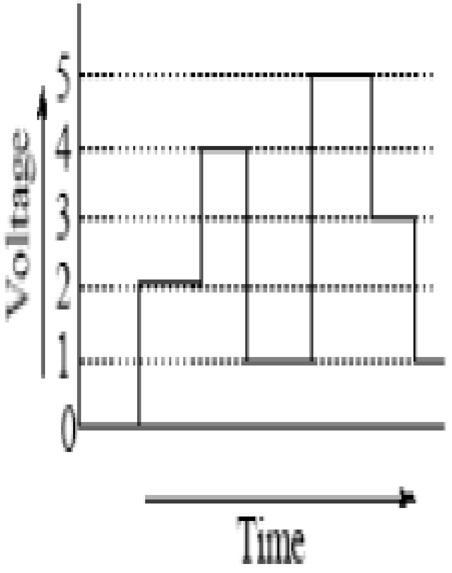
ii) Digital

- The term digital refers to any process that is accomplished using discrete units
- Digital computer is the best example of a digital system.
- Almost all digital circuits are really logic circuits because it is much easier to manipulate and process multiple voltage levels

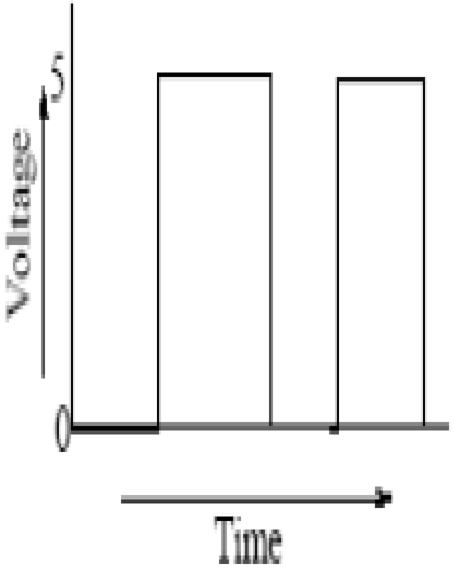
Digital versus Analog (contd.)



Analog Signal



Digital Signal



Logic (Binary) Signal

ADVANTAGES AND DISADVANTAGES

The usual advantages of digital circuits when compared to analog circuits are:

- Information storage can be **easier**
- Robustness

The Disadvantages of digital circuits are:

- Digital circuits are sometimes more **expensive**
- Digital systems must translate from continuous analog signals to discrete digital signals. This causes **quantization errors.**

NUMBER SYSTEMS

- A system for representing number of certain type is called "Number System".
- Integers are normally written using positional numbering number system, in which each digit represents the coefficient in a power series.

$$N = a_{n-1}r^{n-1} + a_{n-2}r^{n-2} + \cdots + a_2r^2 + a_1r^1 + a_0$$

- Where n is the number of digit, r is the radix or base and a_i is the coefficient

$$0 \leq a_i < r$$

REVIEW OF NUMBER SYSTEMS

- Many number systems are in use in digital technology.
- The decimal system is clearly the most familiar to us because it is tools that we use every day.
- Types of Number Systems are

➢ Decimal Number system

➢ Binary Number system

➢ Octal Number system

➢ Hexadecimal Number system

DECIMAL NUMBER SYSTEM

- Decimal system is composed of 10 numerals or symbols. These 10 symbols are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
- Using these symbols as digits of a number, we can express any quantity.
- The decimal system is also called the base-10 system because it has 10 digits.
- Even though the decimal system has only 10 symbols, any number of any magnitude can be expressed by using our system of positional weighting.

Example: 3.14_{10} , 52_{10} ,1024_{10}

BINARY SYSTEM

- In the binary system, there are only two symbols or possible digit values, 0 and 1.
- This base-2 system can be used to represent any quantity that can be represented in decimal or other base system.
- Binary quantities can be represented by any device that has only two operating states or possible conditions.
- E.g.. A switch is only open or closed. We arbitrarily (as we define them) let an open switch represent binary 0 and a closed switch represent binary 1. Thus we can represent any binary number by using series of switches.

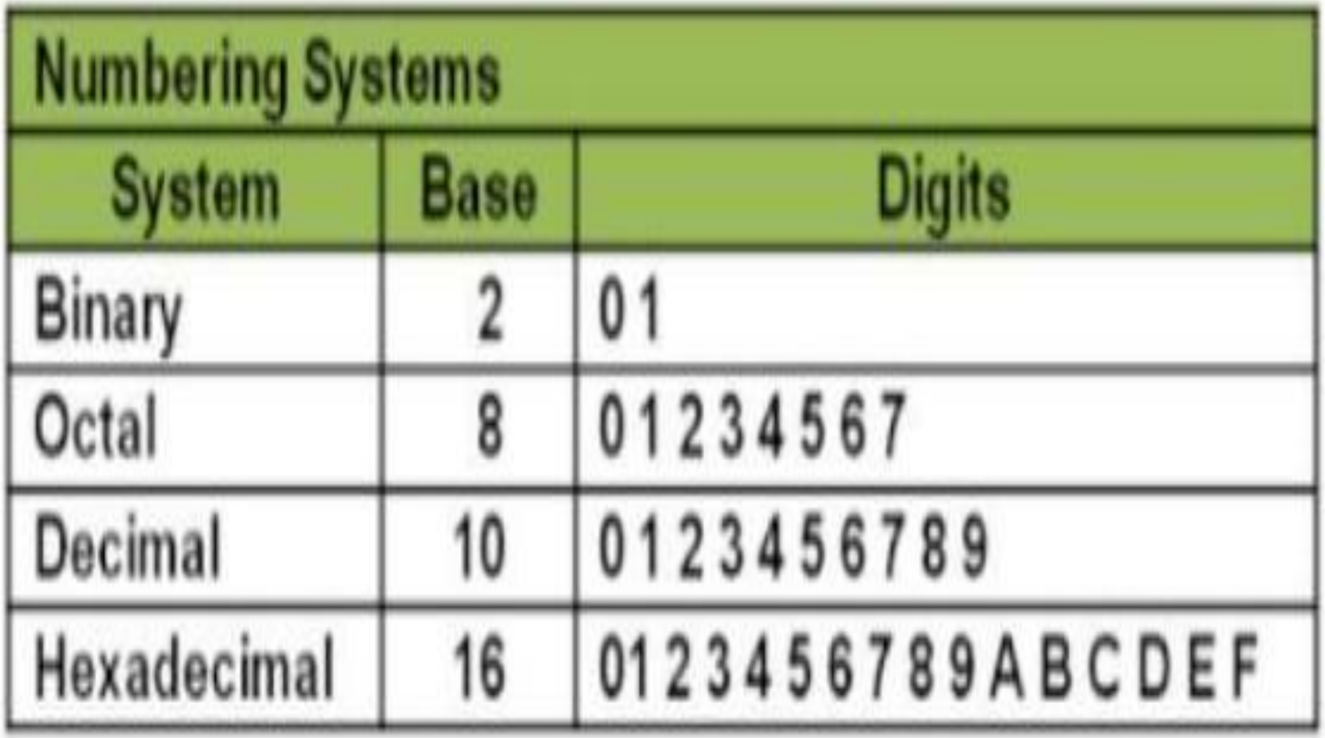
OCTAL NUMBER SYSTEM

- The octal number system has a base of eight, meaning that it has eight possible digits: 0,1,2,3,4,5,6,7.
- An older computer-based number system is "octal" or base eight. The digits in octal math are 0, 1, 2, 3, 4, 5, 6, and 7. The value "eight" is written as "1 eight and 0 ones" or 10_8.
- Since, the octal numbers uses less number of digits as compared to decimal numbers and hexadecimal numbers therefore it is easy to do computations in fewer steps and also less chances of occurrence of error.

HEXADECIMAL NUMBER SYSTEM

- The hexadecimal system uses base 16. Thus, it has 16 possible digit symbols. It uses the digits 0 through 9 plus the letters A, B, C, D, E, and F as the 16 digit symbols.
- Hexadecimal numbering system is often used by programmers to simplify the binary numbering system. Since 16 is equivalent to 2^4, there is a linear relationship between the numbers 2 and 16.
- Computers use binary numbering system while humans use hexadecimal numbering system to shorten binary and make it easier to understand.

Fig: Number system and their Base value



Numbering Systems		
System	Base	Digits
Binary	2	0 1
Octal	8	0 1 2 3 4 5 6 7
Decimal	10	0 1 2 3 4 5 6 7 8 9
Hexadecimal	16	0 1 2 3 4 5 6 7 8 9 A B C D E F

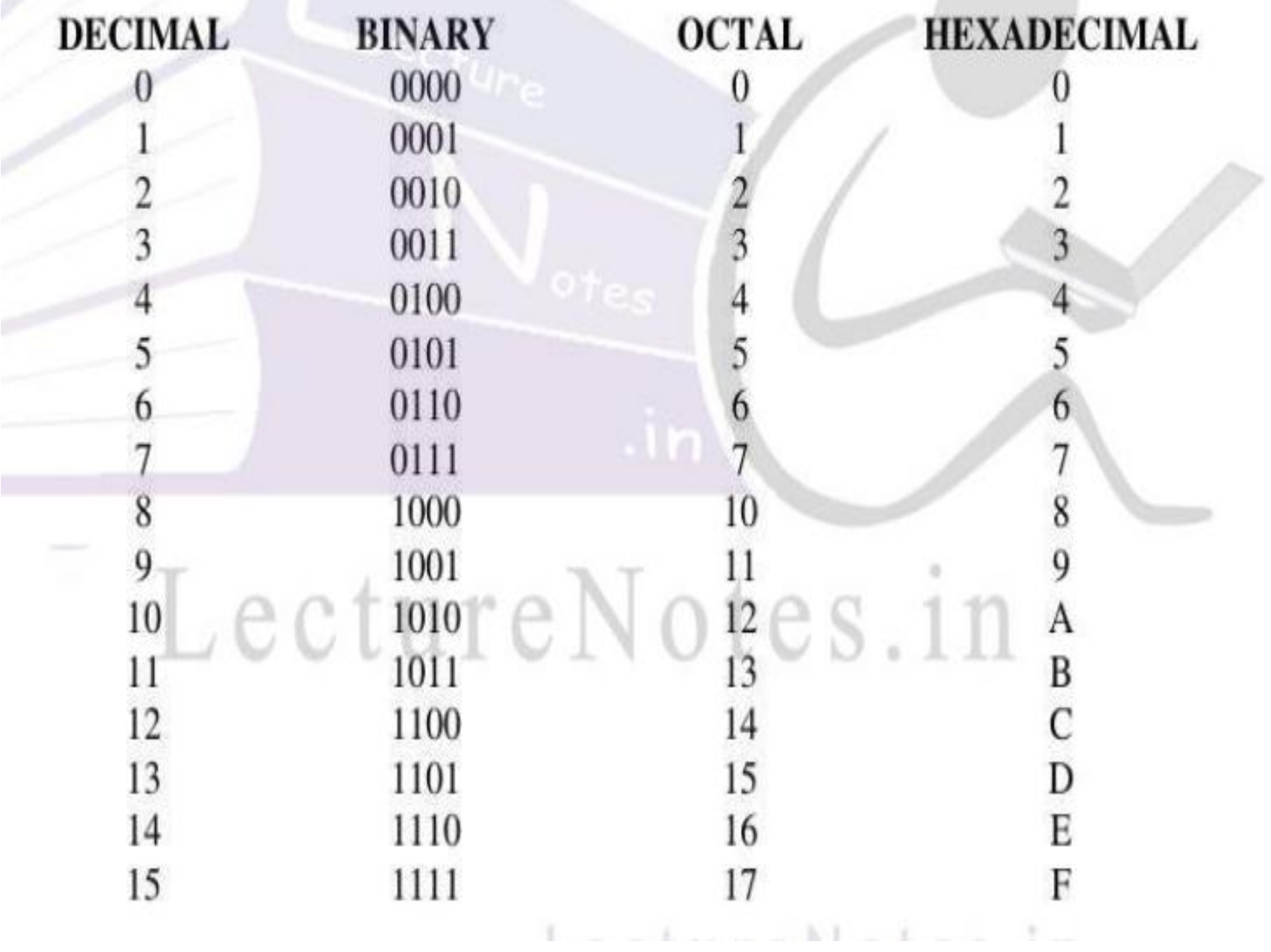


Fig: Types of Number Systems

DECIMAL	BINARY	OCTAL	HEXADECIMAL
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

8421 CODE

8 + 4 + 2 + 1 (8421 code)

5 = 0 + 1 + 0 + 1 = 0101

9 = 1 + 0 + 0 + 1 = 1001

15 = 1 + 1 + 1 + 1 = 1111

CODE CONVERSION

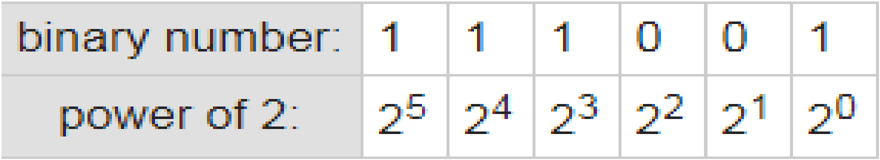
- Coding is the process of translating the input information which can be understandable by the machine or a particular device.
- Coding can be used for security purpose to protect the information from steeling or interrupting.
- Converting from one code form to another code form is called code conversion, like converting from binary to decimal or converting from hexadecimal to decimal.

BINARY TO DECIMAL COVERSION

- Any binary number can be converted to its decimal equivalent simply by summing together the weights of the various positions in the binary number which contain a 1.
- The decimal number is equal to the sum of binary digits (d_n) times their power of 2 (2^n):
- decimal = $d_0 \times 2^0 + d_1 \times 2^1 + d_2 \times 2^2 + \ldots$

Example

Find the decimal value of 111001_2:



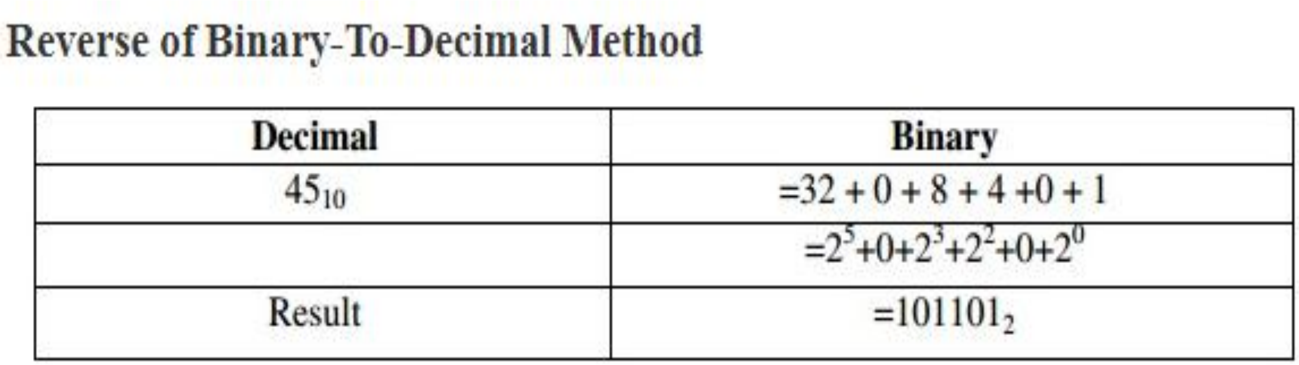
binary number:	1	1	1	0	0	1
power of 2:	2^5	2^4	2^3	2^2	2^1	2^0

- $111001_2 = 1\cdot2^5+1\cdot2^4+1\cdot2^3+0\cdot2^2+0\cdot2^1+1\cdot2^0 = 57_{10}$

DECIMAL TO BINARY CONVERSION

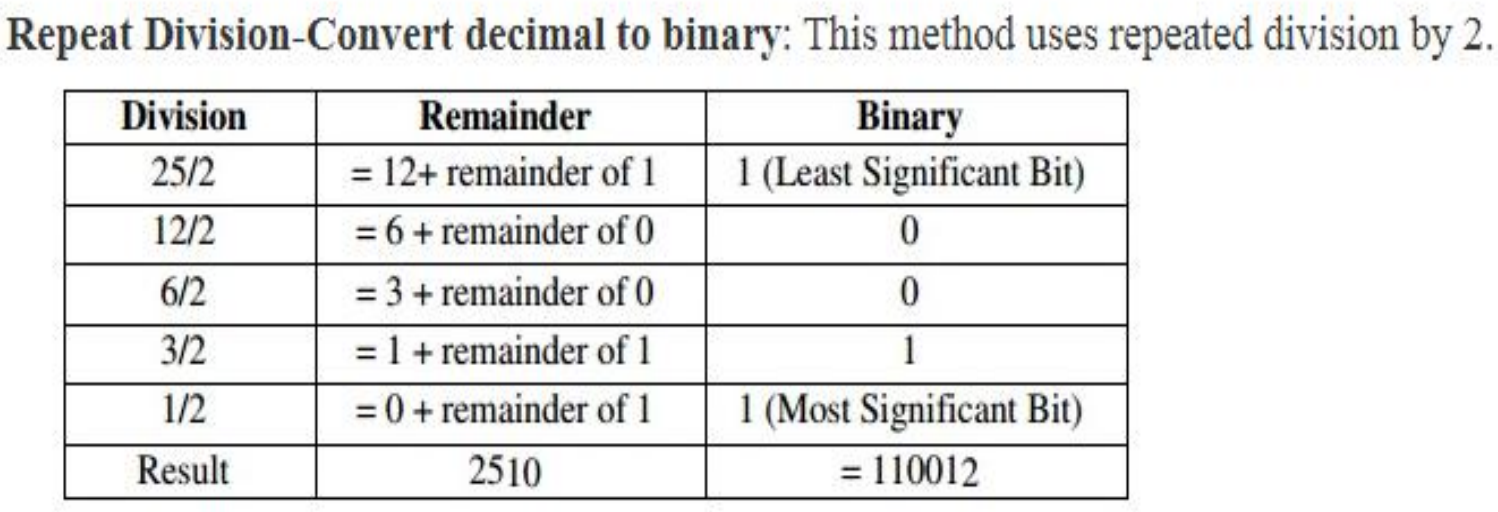
There are 2 methods:

- Reverse of Binary-To-Decimal Method
- Repeat Division



Reverse of Binary-To-Decimal Method

Decimal	Binary
45_{10}	=32 + 0 + 8 + 4 +0 + 1
	$=2^5+0+2^3+2^2+0+2^0$
Result	$=101101_2$



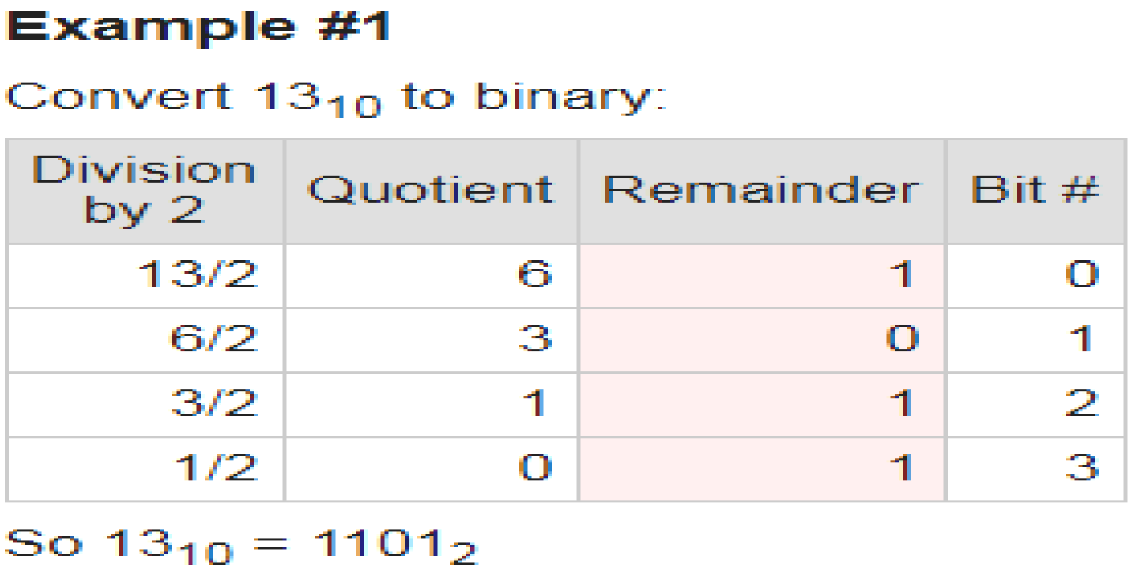
Repeat Division-Convert decimal to binary: This method uses repeated division by 2.

Division	Remainder	Binary
25/2	= 12+ remainder of 1	1 (Least Significant Bit)
12/2	= 6 + remainder of 0	0
6/2	= 3 + remainder of 0	0
3/2	= 1 + remainder of 1	1
1/2	= 0 + remainder of 1	1 (Most Significant Bit)
Result	2510	= 110012

Contd,.

Conversion steps:

- Divide the number by 2.
- Get the integer quotient for the next iteration.
- Get the remainder for the binary digit.
- Repeat the steps until the quotient is equal to 0.



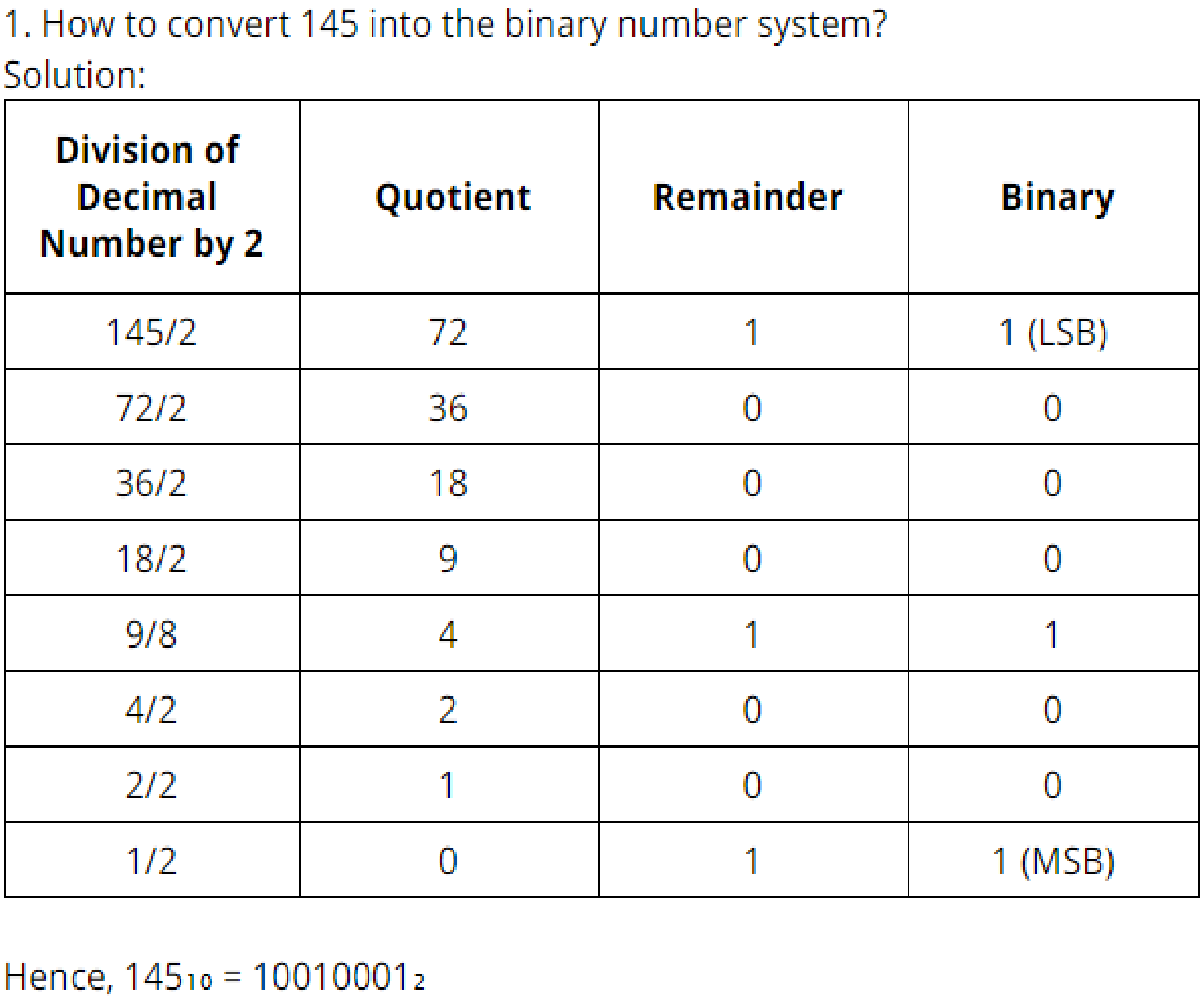
Example #1

Convert 13_{10} to binary:

Division by 2	Quotient	Remainder	Bit #
13/2	6	1	0
6/2	3	0	1
3/2	1	1	2
1/2	0	1	3

So $13_{10} = 1101_2$

EXAMPLE



1. How to convert 145 into the binary number system?

Solution:

Division of Decimal Number by 2	Quotient	Remainder	Binary
145/2	72	1	1 (LSB)
72/2	36	0	0
36/2	18	0	0
18/2	9	0	0
9/8	4	1	1
4/2	2	0	0
2/2	1	0	0
1/2	0	1	1 (MSB)

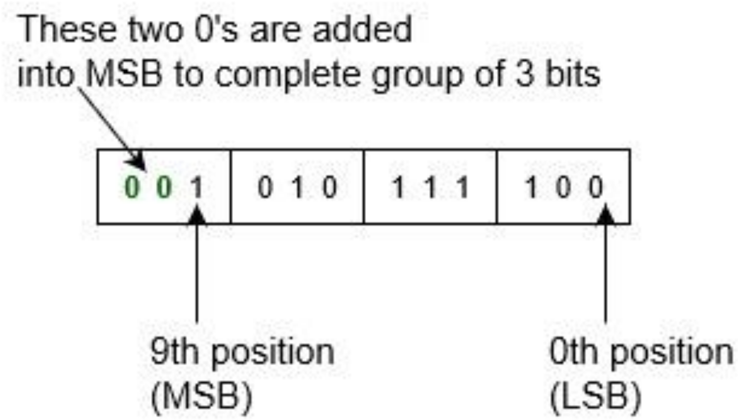
Hence, $145_{10} = 10010001_2$

BINARY TO OCTAL

The following are the steps to convert a binary number into octal number.

- Take binary number
- Divide the binary digits into groups of three (starting from right) for integer part and start from left for fraction part.
- Convert each group of three binary digits to one octal digit.

Example-1 − Convert binary number 1010111100 into octal number. Since there is no binary point here and no fractional part. So,



These two 0's are added into MSB to complete group of 3 bits

0 0 1	0 1 0	1 1 1	1 0 0

9th position (MSB)

0th position (LSB)

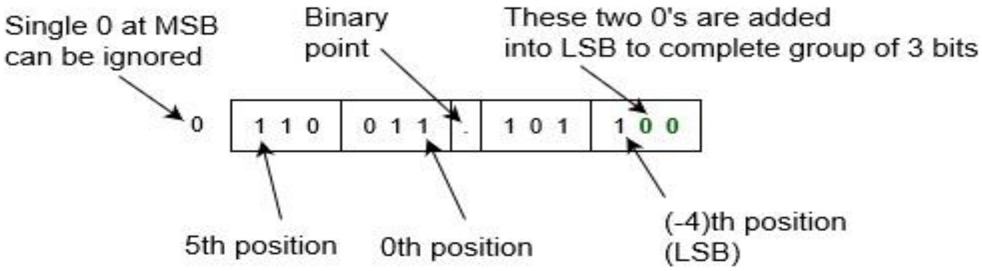
Contd,.

- Therefore, Binary to octal is.

$= (001\quad 010\quad 111\quad 100)_2$

$= (\quad 1\qquad 2\qquad 7\qquad 4)_8$

Example-2 Convert binary number 0110 011.1011 into octal number. Since there is binary point here and fractional part. So,



- Therefore, Binary to octal is.

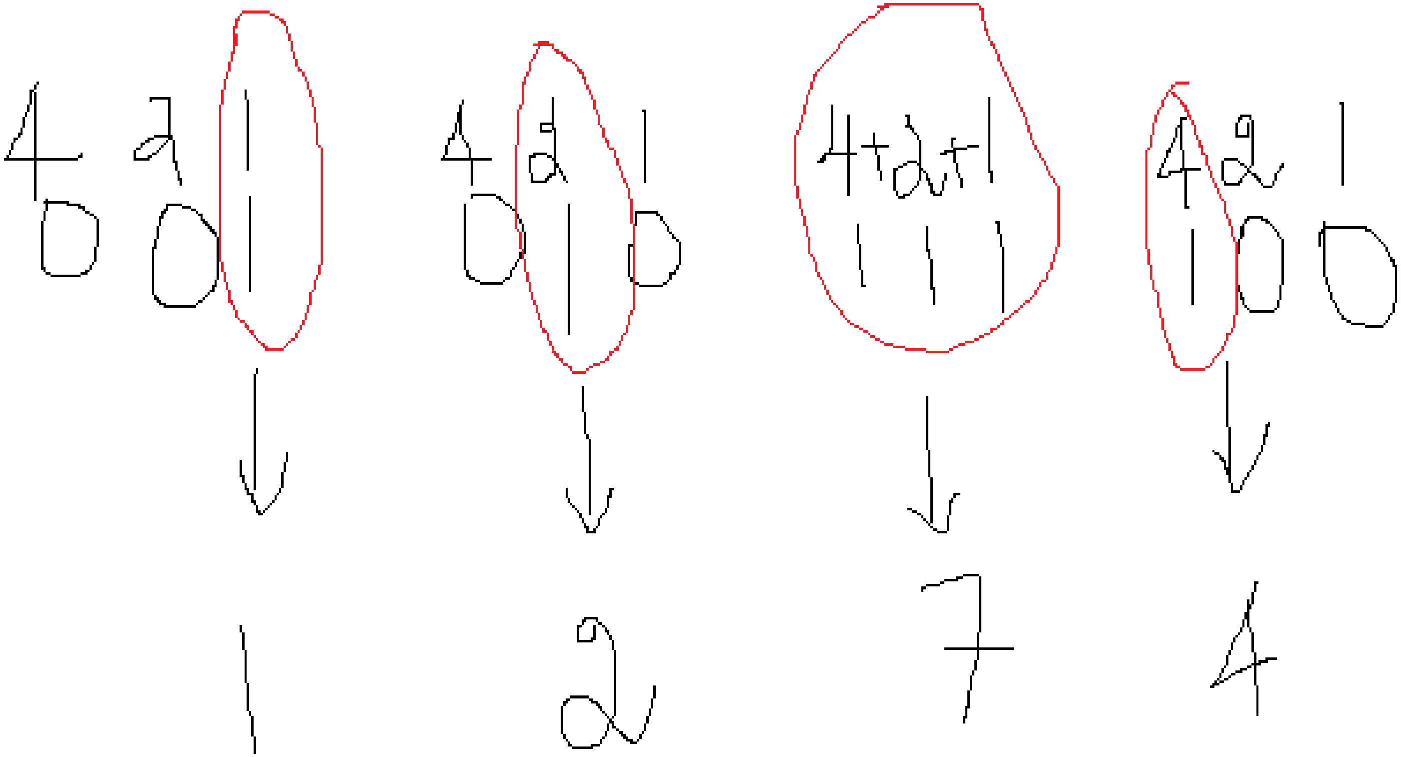
$= (0110\ 011.1011)_2$

$= (0\ 110\ 011\ .\ 101\ \ 1)_2$

$= (110\ 011\ .\ 101\ \ 100)_2$

$= (6\ 3\ .\ 5\ 4)_8$

Contd,.



OCTAL TO BINARY

This method is simple and also works as reverse of Binary to Octal Conversion. The algorithm is explained as following below.

- Take Octal number as input
- Convert each digit of octal into binary.
- That will be output as binary number.

Example: Convert octal number 540 into binary number.

According to above algorithm, equivalent binary number will be,

$= (540)_8$

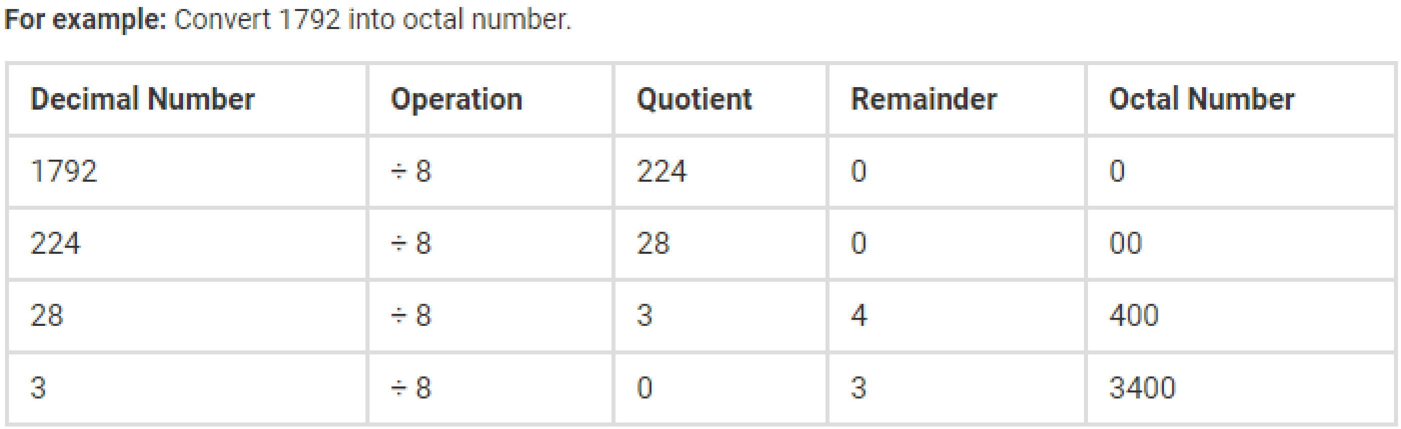
$= (101\ 100\ 000)_2$

$= (101100000)_2$

DECIMAL TO OCTAL

Follow the steps given below to learn the decimal to octal conversion:

- Write the given decimal number
- If the given decimal number is less than 8 the octal number is the same.
- If the decimal number is greater than 7 then divide the number by 8.
- Note the remainder we get after division
- Repeat step 3 and 4 with the quotient till it is less than 8
- Now, write the remainders in reverse order(bottom to top)
- The resultant is the equivalent octal number to the given decimal number.

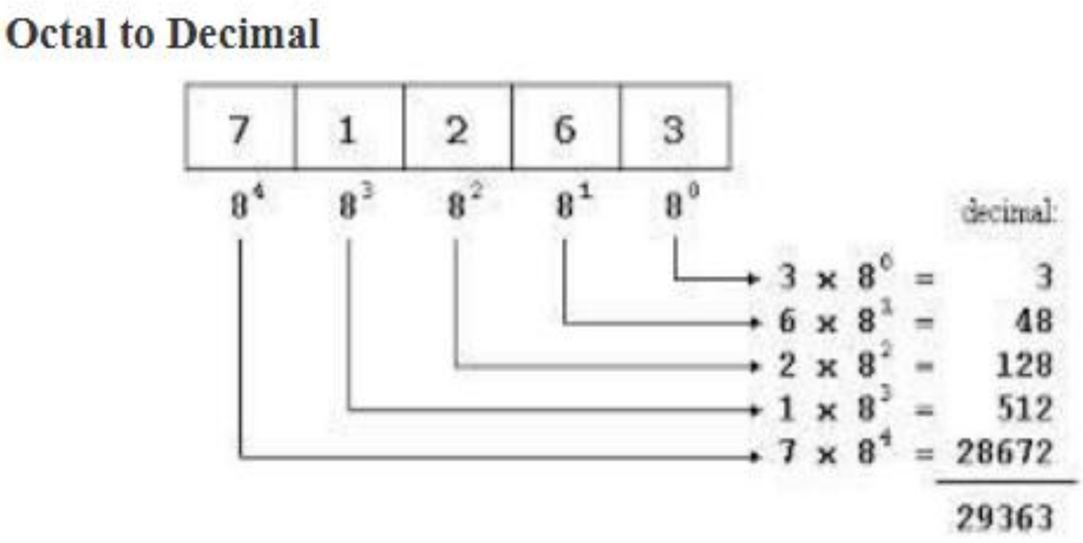


For example: Convert 1792 into octal number.

Decimal Number	Operation	Quotient	Remainder	Octal Number
1792	÷ 8	224	0	0
224	÷ 8	28	0	00
28	÷ 8	3	4	400
3	÷ 8	0	3	3400

OCTAL TO DECIMAL

- To convert an octal number (base-8) to the decimal (base-10) number system, we need to use octal place value to add the base-10 value of each digit.
- In the octal place value system, each time you move a place to the left, the value increases eight-fold.



Octal to Decimal

HEXADECIMAL TO DECIMAL

Here are the steps to convert hex to decimal:

- Get the decimal equivalent of hex from table.
- Multiply every digit with 16 power of digit location.
- Sum all the multipliers.

Example:

- 7DE = (7 * 16^2) + (13 * 16^1) + (14 * 16^0)

= (7 * 256) + (13 * 16) + (14 * 1)

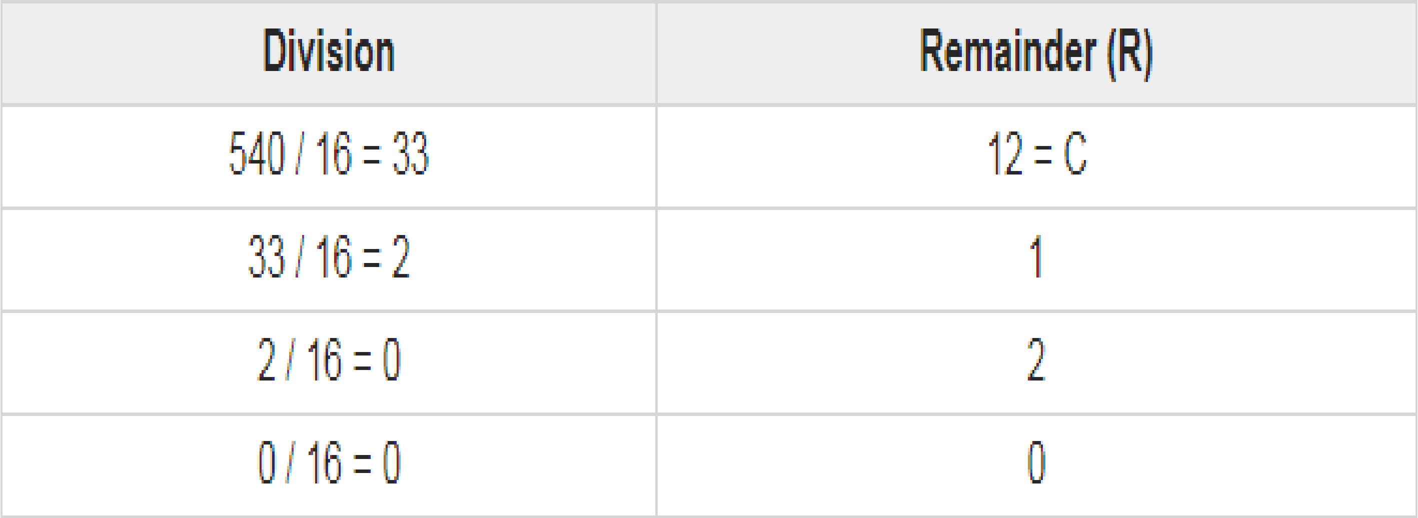
= 1792 + 208 + 14

7DE = 2014

DECIMAL TO HEXADECIMAL

- Take decimal number as dividend.
- Divide this number by 16 (16 is base of hexadecimal so divisor here).
- Store the remainder in an array (it will be: 0 to 15 because of divisor 16, replace 10, 11, 12, 13, 14, 15 by A, B, C, D, E, F respectively).
- Repeat the above two steps until the number is greater than zero.
- **Example** − Convert decimal number 540 into hexadecimal number.
- Since given number is decimal integer number, so by using above algorithm performing short division by 16 with remainder.

Contd,.



Division	Remainder (R)
540 / 16 = 33	12 = C
33 / 16 = 2	1
2 / 16 = 0	2
0 / 16 = 0	0

- Now, write remainder from bottom to up (in reverse order), this will be 021C (or only 21C) which is equivalent hexadecimal number of decimal integer 540.

BINARY TO HEXADECIMAL

- Hexadecimal number system provides convenient way of converting large binary numbers into more compact and smaller groups.
- First, we need to convert a binary into other base system (e.g., into decimal, or into octal). Then we need to convert it hexadecimal number.

Example – Convert binary number 1101010 into hexadecimal number.

- First convert this into decimal number:
- $(1101010)_2 = 1x2^6+1x2^5+0x2^4+1x2^3+0x2^2+1x2^1+0x2^0$

  $= 64+32+0+8+0+2+0 = (106)_{10}$
- Then, convert it into hexadecimal number

$(106)_{10}$ =106/16=6.625=0.625*16=10(A)

=6/16=0.35=0.35*16= 6

= $(6A)_{16}$ which is answer.

HEXADECIMAL TO BINARY

- **Step 1**: Write down the hex number. If there are any, change the hex values represented by letters to their decimal equivalents.
- **Step 2**: Each hex digit represents four binary digits and therefore is equal to a power of 2.
- **Step 3**: Determine which powers of two (8, 4, 2 or 1) sum up to your hex digits.
- **Step 4**: Write down 1 below those 8, 4, 2 and 1's that are used. Write down 0 below those that are not used.
- **Step 5**: Read the 1's and 0's from left to right to get the binary equivalent of the given hex number.

Contd,.

- Example 1:

$(2C1)_{16}$ = ?

2 C 1

2 12 1 (using 8421 code)

0010 1100 0001 -------→ binary number

Example 2:

$(9DB2)_{16}$ = ?

9 D B 2

9 13 11 2 (using 8421 code)

1001 1101 1011 0010 -------→ binary number

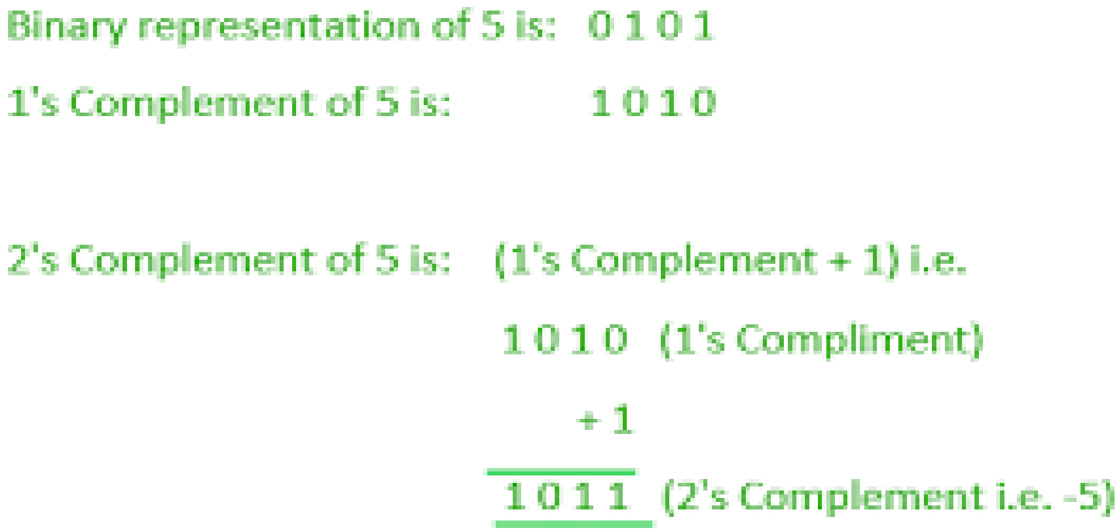
COMPLEMENT OF NUMBERS

- Complements are used in digital computers to simplify the subtraction operation and for logical manipulation.
- There are TWO types of complements for each base-r system: the radix complement and the diminished radix complement.
- The first is referred to as the r's complement and the second as the (r - 1)'s complement.
- The two types are referred to as
- **2's complement and**
- **1's complement for binary numbers**

and the 10’s complement a complement for decimal numbers.

1’s and 2’s COMPLEMENT

- The 1’s complement of a binary number is the number that results when we change all1’s to zeros and the zeros to ones.
- The 2’s complement is the binary number that results when we add 1 to the 1’s complement. It is used to represent negative numbers.



```
Binary representation of 5 is:  0 1 0 1
1's Complement of 5 is:         1 0 1 0

2's Complement of 5 is:   (1's Complement + 1) i.e.
                           1 0 1 0  (1's Compliment)
                              + 1
                           1 0 1 1  (2's Complement i.e. -5)
```

ARITHMETIC OPERATIONS

- The basic arithmetic operations for real numbers are addition, subtraction, multiplication, and division.
- The basic arithmetic properties are the commutative, associative, and distributive properties.

Commutative Property

- The commutative property describes equations in which the order of the numbers involved does not affect the result. Addition and multiplication are commutative operations:

$$2+3=3+2=5$$

$$5\cdot2=2\cdot5=10$$

Subtraction and division, however, are not commutative.

Associative Property

- The associative property describes equations in which the grouping of the numbers involved does not affect the result.

ARITHMETIC OPERATIONS

- As with the commutative property, addition and multiplication are associative operations:

$$(2+3)+6 = 2+(3+6)=11$$

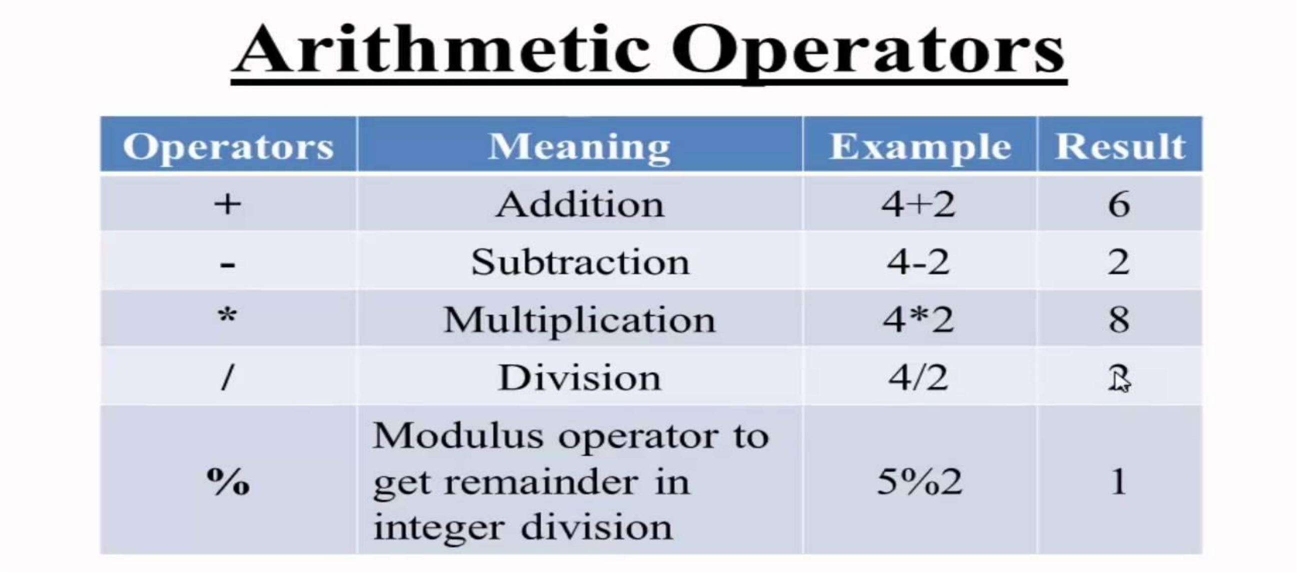
$$(4\cdot 1)\cdot 2 = 4\cdot(1\cdot 2)=8$$

Subtraction and division are not associative.

Distributive Property

- The distributive property can be used when the sum of two quantities is then multiplied by a third quantity.

$$(2+4)\cdot 3 =2\cdot 3+4\cdot 3= 18$$



Arithmetic Operators

Operators	Meaning	Example	Result
+	Addition	4+2	6
-	Subtraction	4-2	2
*	Multiplication	4*2	8
/	Division	4/2	2
%	Modulus operator to get remainder in integer division	5%2	1

RULES FOR ARTHIMETIC OPERATIONS

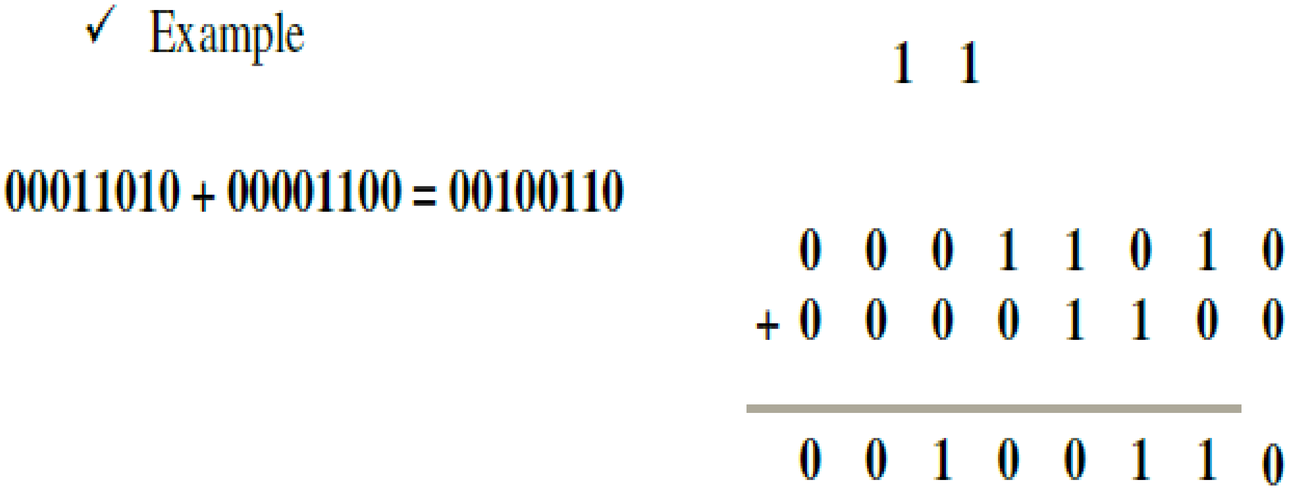
1. Rules of Binary Addition

0 + 0 = 0

0 + 1 = 1

1 + 0 = 1

1 + 1 = 0, and carry 1 to the next more significant bit



- ✓ Example

00011010 + 00001100 = 00100110

$$\begin{array}{ccccccccc}
 & & & 1 & 1 & & & & \\
 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\
+ & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
\hline
 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0
\end{array}$$

RULES FOR ARTHIMETIC OPERATIONS

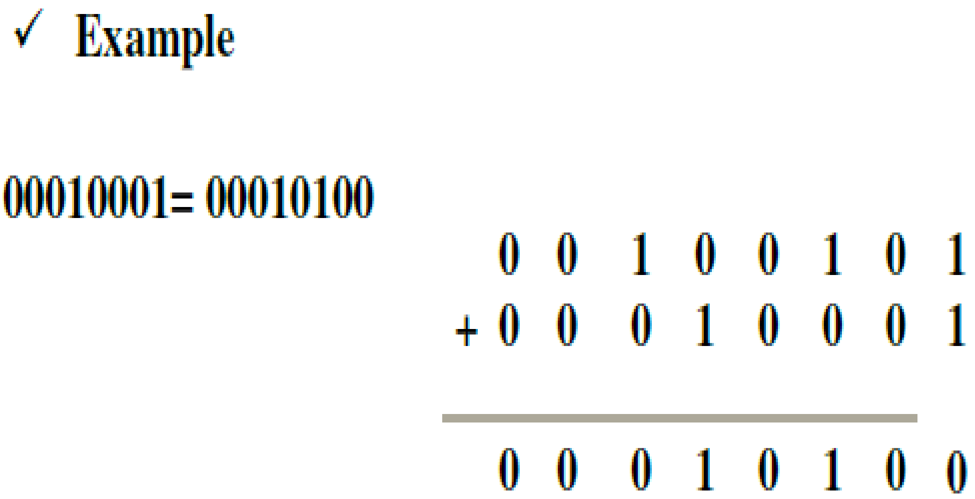
2. **Rules of Binary Subtraction**

0 - 0 = 0

0 - 1 = 1, and borrow 1 from the next more significant bit

1 - 0 = 1

1 - 1 = 0

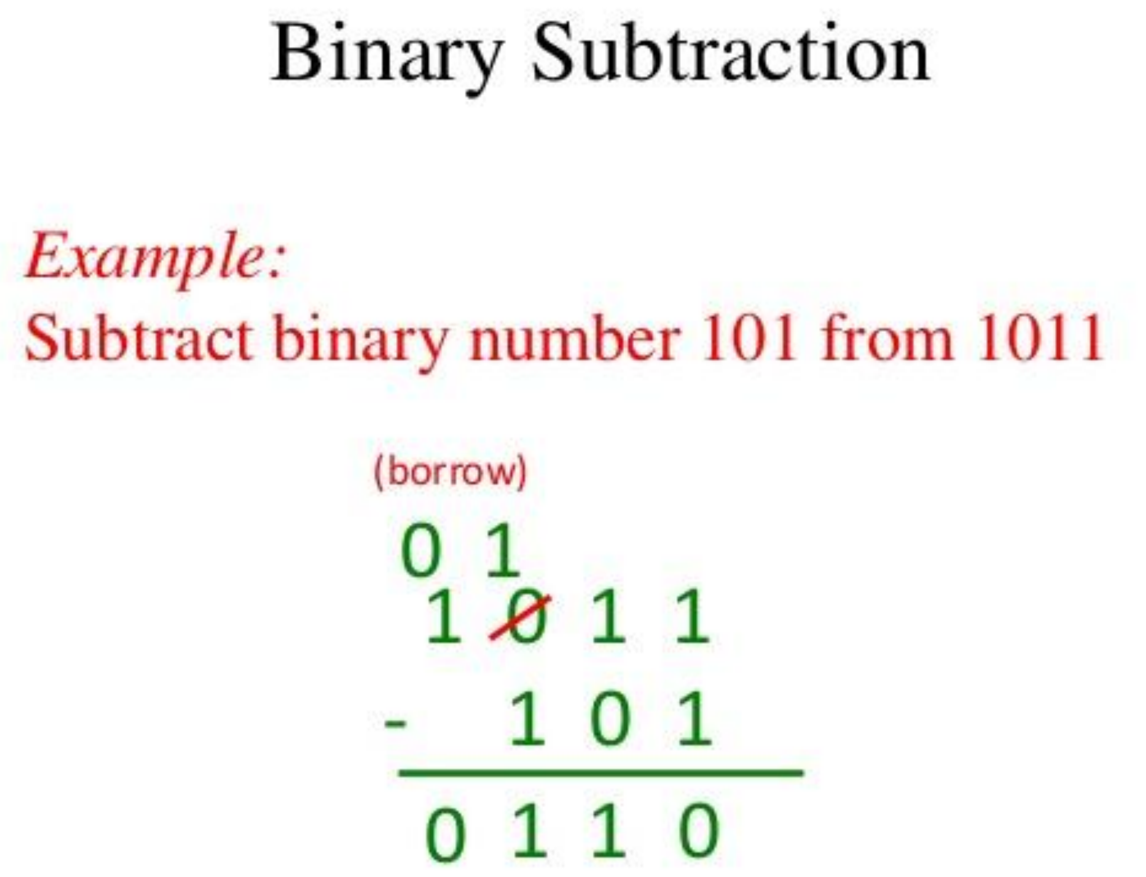


✓ Example

00100101 - 00010001= 00010100

```
   0 0 1 0 0 1 0 1
 + 0 0 0 1 0 0 0 1
 _________________
   0 0 0 1 0 1 0 0
```

EXAMPLE



Binary Subtraction

Example:

Subtract binary number 101 from 1011

(borrow)

$$\begin{array}{ccccc} & 0 & 1 & & \\ & 1 & \cancel{0} & 1 & 1 \\ - & & 1 & 0 & 1 \\ \hline & 0 & 1 & 1 & 0 \end{array}$$

RULES FOR ARTHIMETIC OPERATIONS

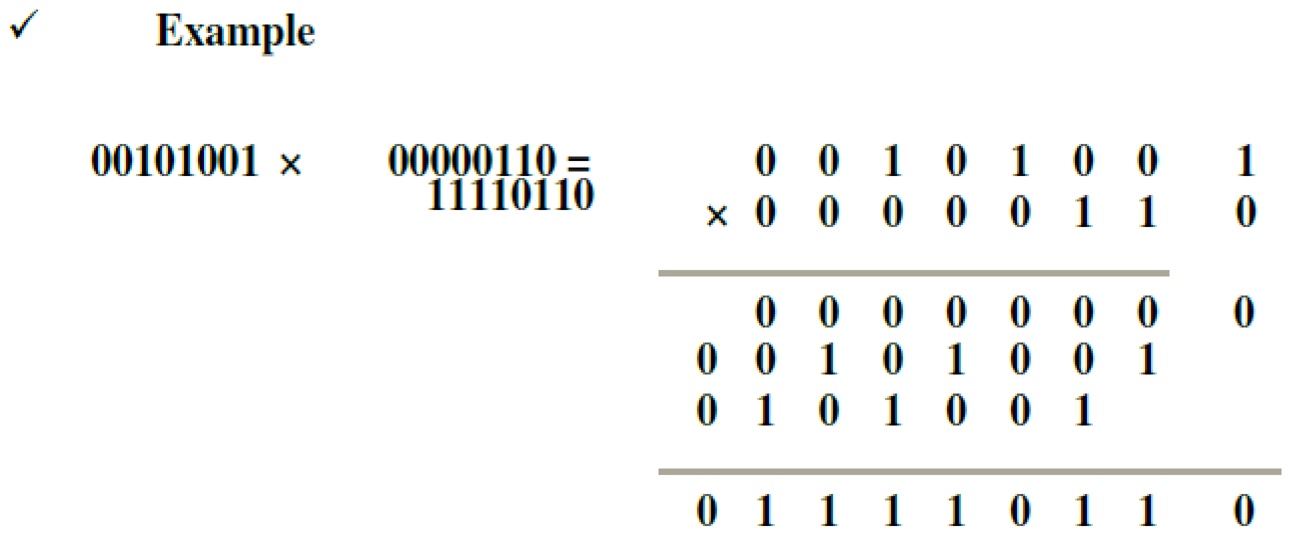
3) Rules of Binary Multiplication

0 x 0 = 0

0 x 1 = 0

1 x 0 = 0

1 x 1 = 1, and no carry or borrow bits



✓ **Example**

00101001 × 00000110 = 11110110

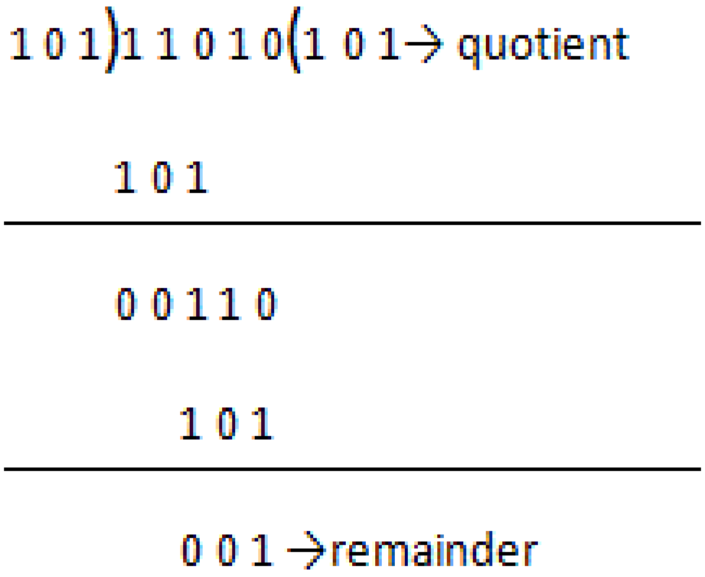
```
     0  0  1  0  1  0  0     1
  ×  0  0  0  0  0  1  1     0
  -----------------------
     0  0  0  0  0  0  0     0
  0  0  1  0  1  0  0  1
  0  1  0  1  0  0  1
  ----------------------------
  0  1  1  1  1  0  1  1     0
```

RULES FOR ARTHIMETIC OPERATIONS

4) **Rules of Binary Division**

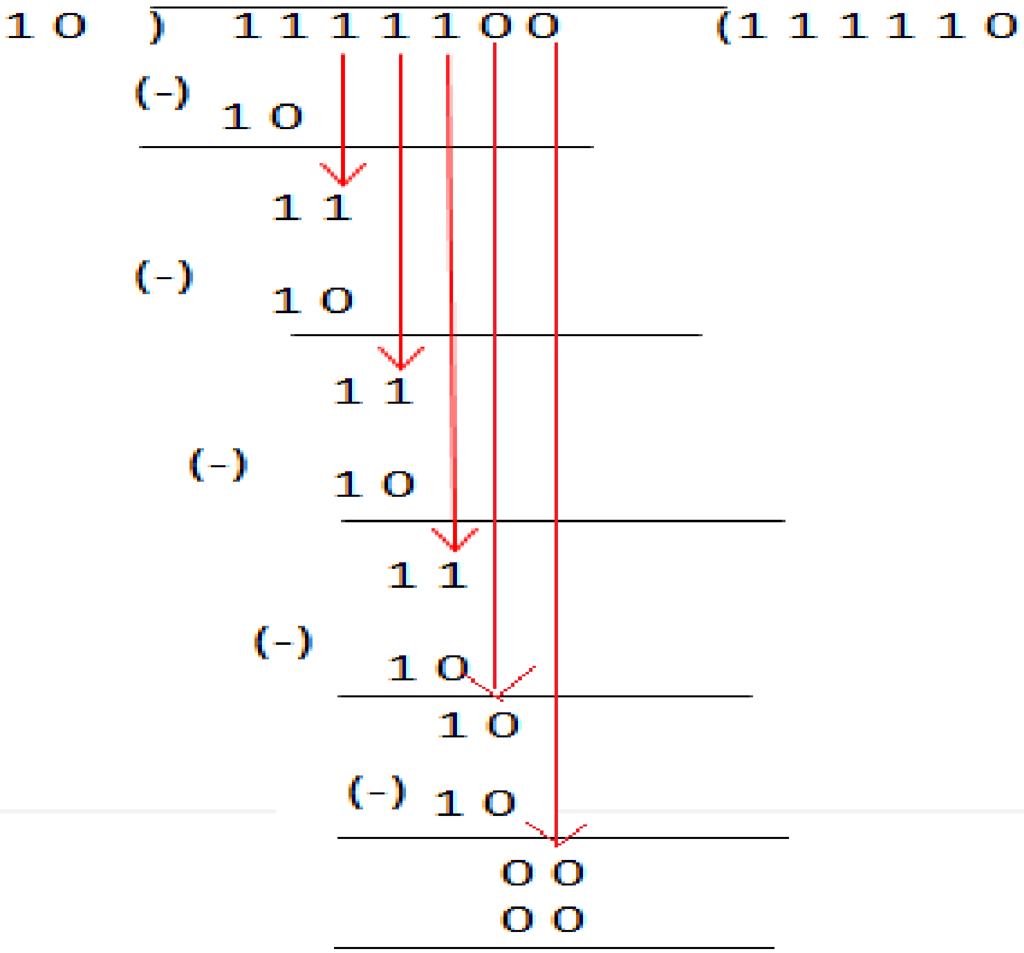
- The process of binary **division** does not have any **specific** rules to follow. Though this process is quite similar to the decimal division.

Example:



```
101)11010(101→ quotient
    101
--------------------
    00110
      101
--------------------
      001→remainder
```

BINARY DIVISION



```
      _______________
10 )  1 1 1 1 1 0 0    (1 1 1 1 1 0
(-)   1 0
      ---
        1 1
(-)     1 0
        ---
          1 1
(-)       1 0
          ---
            1 1
(-)         1 0
            ---
              1 0
(-)           1 0
              ---
                0 0
                0 0
                ---
```

BINARY CODES

- The digital data is represented, stored and transmitted as group of binary bits. This group is also called as **binary code**.
- Binary codes are codes which are represented in binary system with modification from the original ones.
- The types of binary codes are:

1) Weighted codes.
2) Non-Weighted codes.
3) Binary Coded Decimal Code
4) Alphanumeric Codes
5) Error Detecting Codes
6) Error Correcting Codes

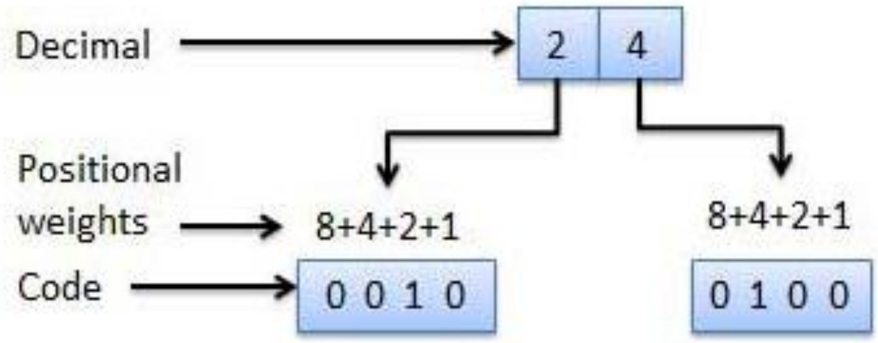
ADVANTAGES OF BINARY CODES

Following is the list of advantages that binary code offers.

- Binary codes are suitable for the computer applications.
- Binary codes are suitable for the digital communications.
- Binary codes make the analysis and designing of digital circuits if we use the binary codes.
- Since only 0 & 1 are being used, implementation becomes easy.

WEIGHTED BINARY CODES

- Weighted binary codes are those binary codes which obey the positional weight principle.
- Each position of the number represents a specific weight.
- Several systems of the codes are used to express the decimal digits 0 through 9.
- In these codes each decimal digit is represented by a group of four bits.



NON-WEIGHTED CODES

- In this type of binary codes, the positional weights are not assigned.
- The examples of non-weighted codes are

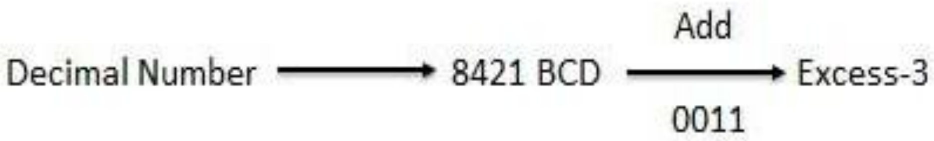
  Excess-3 code and Gray code.

Excess-3 code

- The Excess-3 code is also called as XS-3 code. It is non-weighted code used to express decimal numbers.
- The Excess-3 code words are derived from the 8421 BCD code words adding $(0011)_2$ or (3)10 to each code word in 8421.

EXCESS-3 CODE

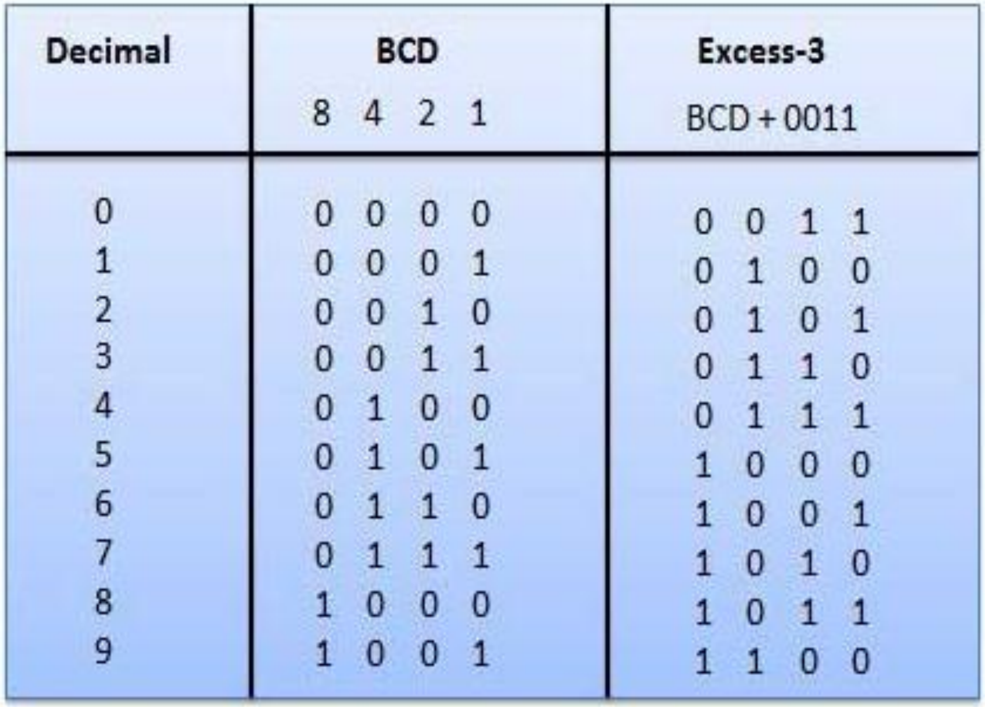
- The excess-3 codes are obtained as follows –



Decimal Number → 8421 BCD → Excess-3

Add 0011

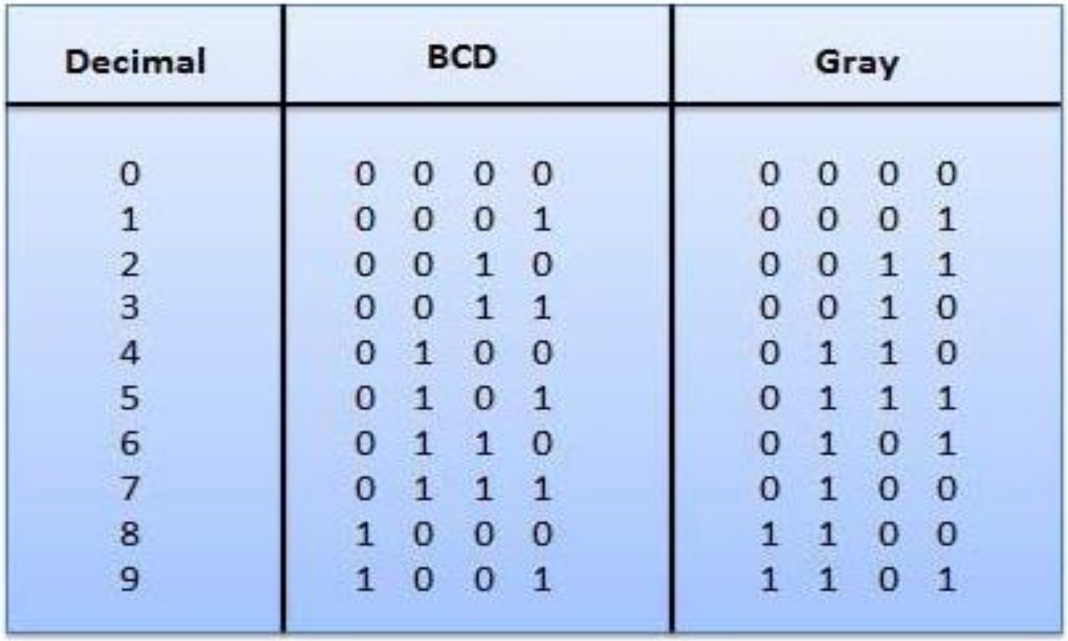
Example



Decimal	BCD 8 4 2 1	Excess-3 BCD + 0011
0	0 0 0 0	0 0 1 1
1	0 0 0 1	0 1 0 0
2	0 0 1 0	0 1 0 1
3	0 0 1 1	0 1 1 0
4	0 1 0 0	0 1 1 1
5	0 1 0 1	1 0 0 0
6	0 1 1 0	1 0 0 1
7	0 1 1 1	1 0 1 0
8	1 0 0 0	1 0 1 1
9	1 0 0 1	1 1 0 0

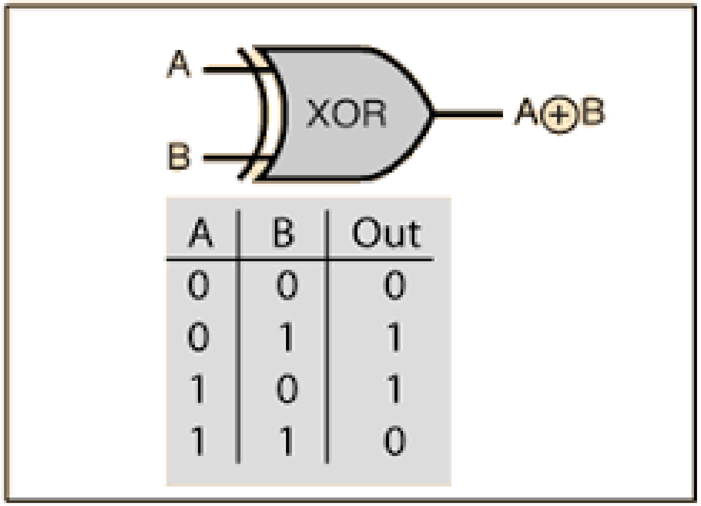
GRAY CODE

- It is the non-weighted code and it is not arithmetic codes. That means there are no specific weights assigned to the bit position.
- As only one bit changes at a time so the gray code is called as a unit distance code. The gray code is also called as a cyclic code.
- Gray code cannot be used for arithmetic operation.

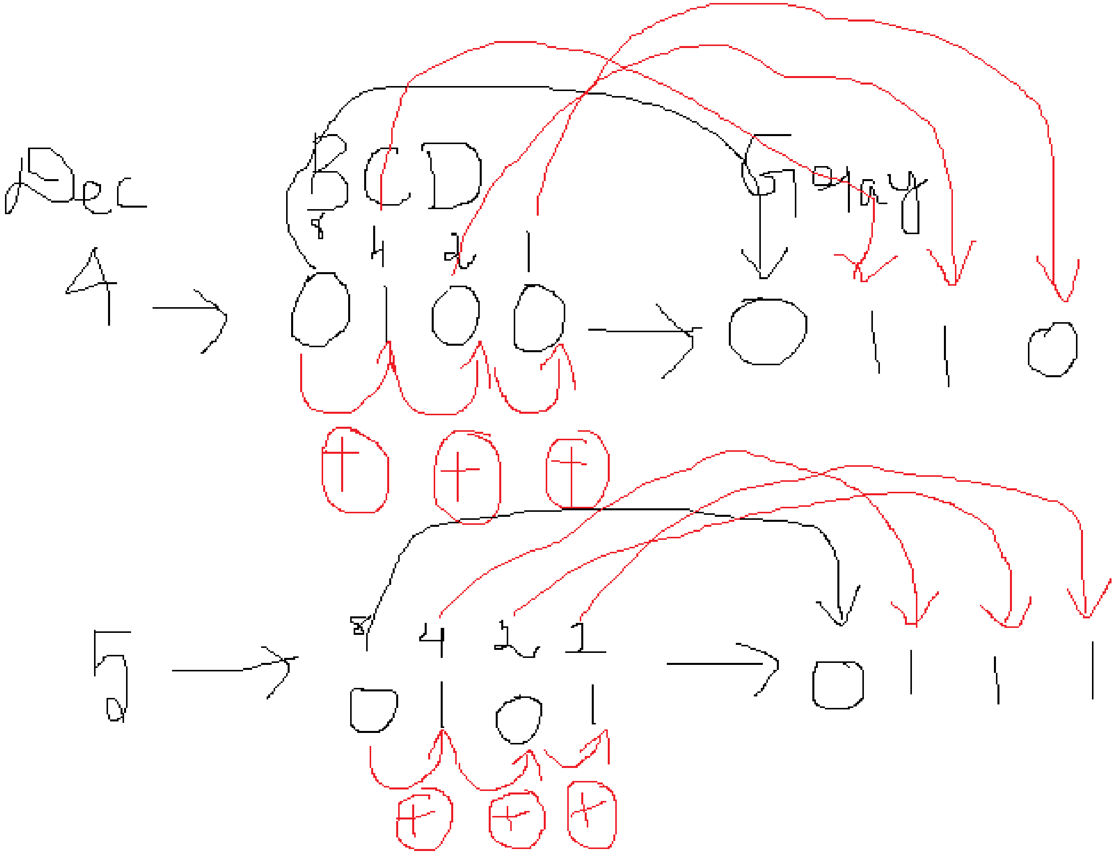


Decimal	BCD	Gray
0	0 0 0 0	0 0 0 0
1	0 0 0 1	0 0 0 1
2	0 0 1 0	0 0 1 1
3	0 0 1 1	0 0 1 0
4	0 1 0 0	0 1 1 0
5	0 1 0 1	0 1 1 1
6	0 1 1 0	0 1 0 1
7	0 1 1 1	0 1 0 0
8	1 0 0 0	1 1 0 0
9	1 0 0 1	1 1 0 1

EXOR OPERATION

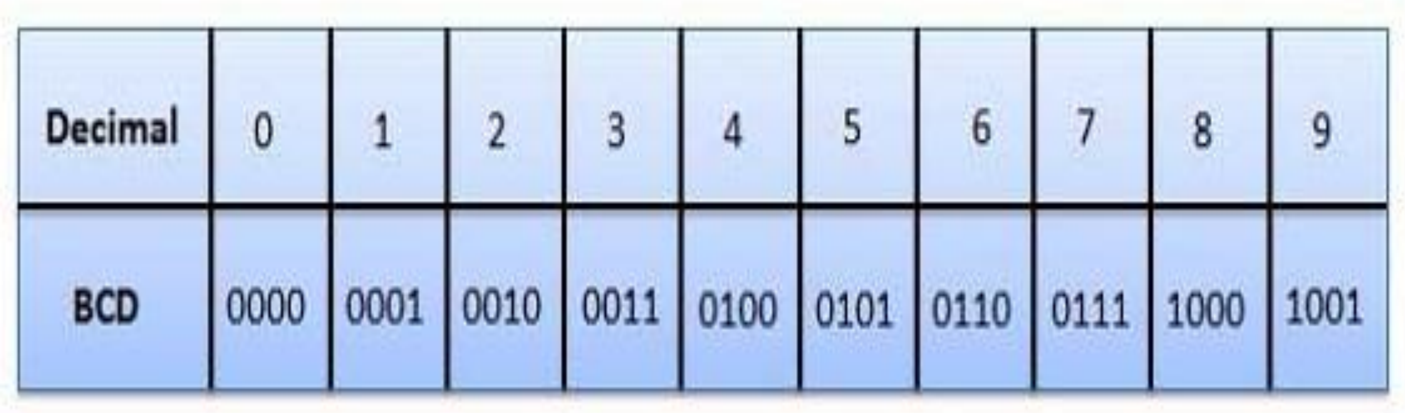


A	B	Out
0	0	0
0	1	1
1	0	1
1	1	0



BINARY CODED DECIMAL (BCD) CODE

- In this code each decimal digit is represented by a 4-bit binary number.
- BCD is a way to express each of the decimal digits with a binary code.
- In the BCD, with four bits we can represent sixteen numbers (0000 to 1111).



Decimal	0	1	2	3	4	5	6	7	8	9
BCD	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001

BCD CODES

Advantages of BCD Codes

- It is very similar to decimal system.
- We need to remember binary equivalent of decimal numbers 0 to 9 only.

Disadvantages of BCD Codes

- The addition and subtraction of BCD have different rules.
- The BCD arithmetic is little more complicated.
- BCD needs more number of bits than binary to represent the decimal number. So BCD is less efficient than binary.

ALPHANUMERIC CODES

- The alphanumeric codes are the codes that represent numbers and alphabetic characters.
- Mostly such codes also represent other characters such as symbol and various instructions necessary for conveying information.
- An alphanumeric code should at least represent 10 digits.
- The following three alphanumeric codes are very commonly used for the data representation.

1.American Standard Code for Information Interchange (ASCII).

2.Extended Binary Coded Decimal Interchange Code (EBCDIC).

ERROR CODES

- There are binary code techniques available to detect and correct data during data transmission.

Error detecting codes

- When data is transmitted from one point to another, there are chances that data may get corrupted. To detect these data errors, we use special codes, which are error detection codes.

Error-correcting codes

- It not only detect errors, but also correct them. This is used normally in Satellite
- communication, where turn-around delay is very high as is the probability of data getting corrupt.

ERROR CODES

Hamming Codes

- Hamming code adds a minimum number of bits to the data transmitted in a noisy channel, to be able to correct every possible one-bit error.
- Two types of parity are:

Even parity: Checks if there is an even number of ones; if so, parity bit is zero. When the number of one's is odd then parity bit is set to 1.

Odd Parity: Checks if there is an odd number of ones; if so, parity bit is zero. When the number of one's is even then parity bit is set to 1.

CODE CONVERSION

- There are many methods or techniques which can be used to convert code from one format to another.

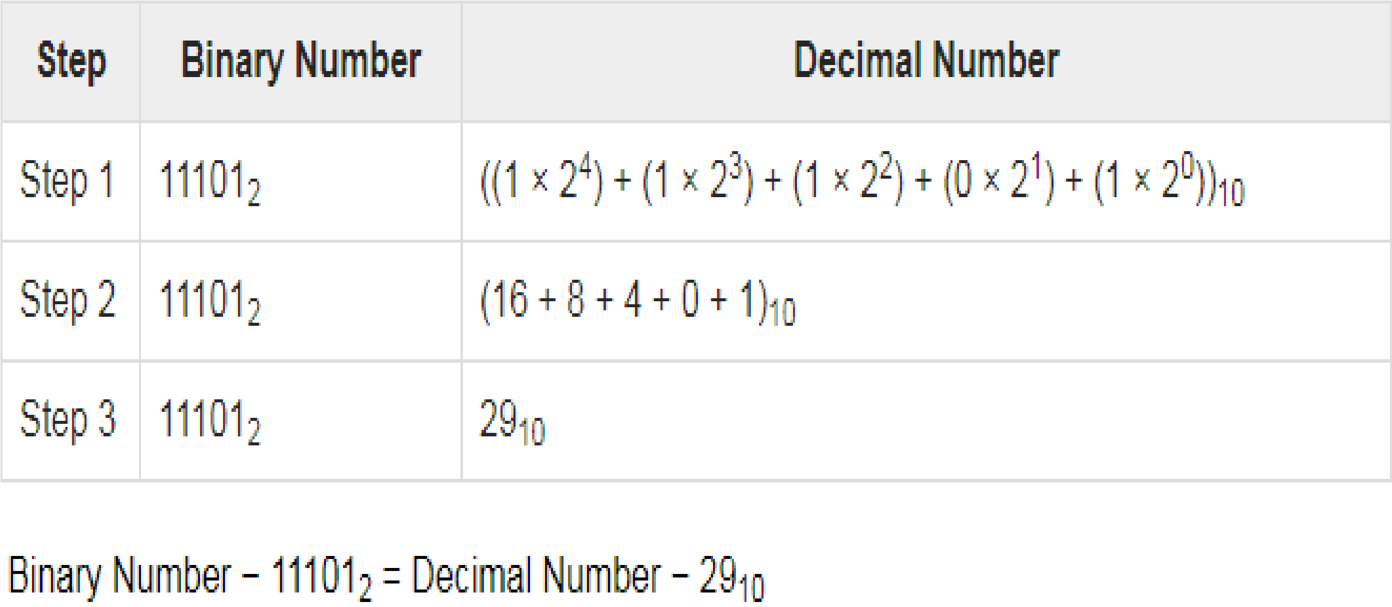
- Binary to BCD Conversion
- BCD to Binary Conversion
- BCD to Excess-3
- Excess-3 to BCD

BINARY TO BCD CONVERSION

- **Step 1** -- Convert the binary number to decimal.
- **Step 2** -- Convert decimal number to BCD.

Example − convert $(11101)_2$ to BCD.

Step 1 − Convert the given number to Decimal

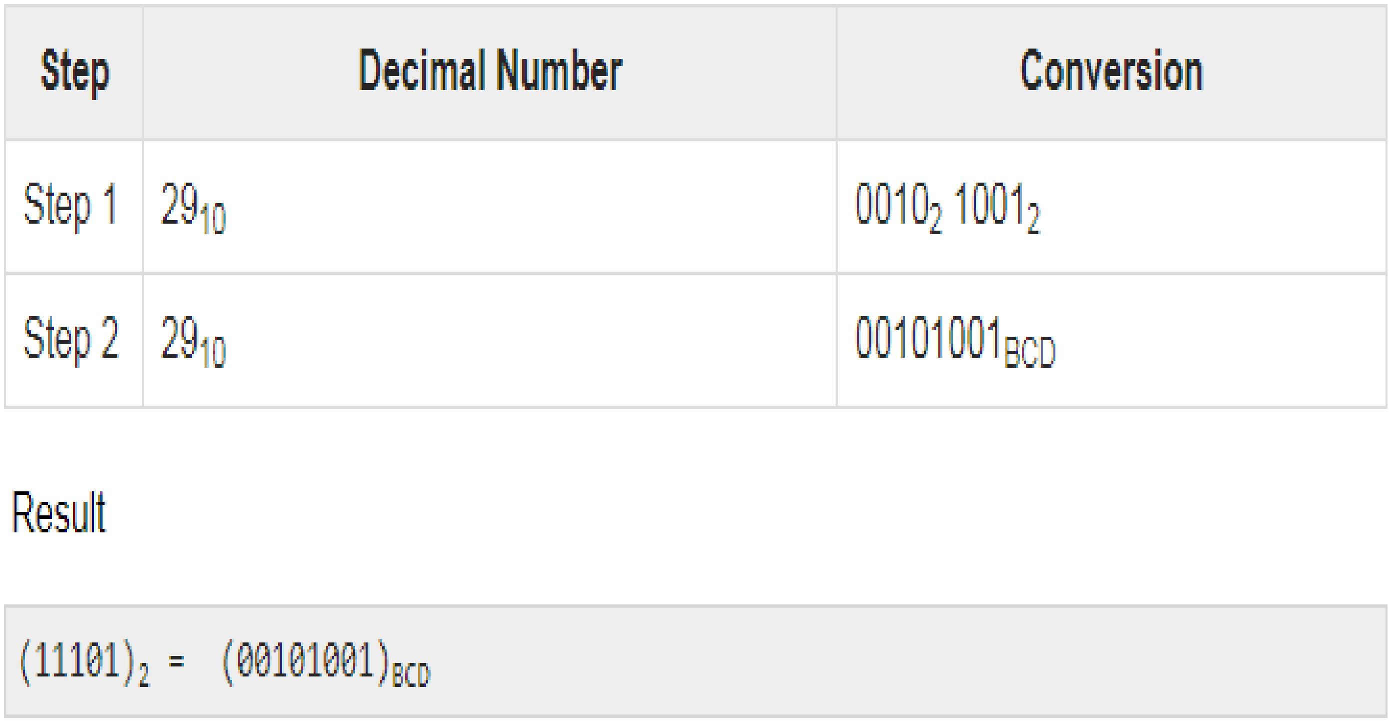


Step	Binary Number	Decimal Number
Step 1	11101_2	$((1 \times 2^4) + (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0))_{10}$
Step 2	11101_2	$(16 + 8 + 4 + 0 + 1)_{10}$
Step 3	11101_2	29_{10}

Binary Number – 11101_2 = Decimal Number – 29_{10}

BINARY TO BCD CONVERSION

- Step 2 − Convert to BCD



Step	Decimal Number	Conversion
Step 1	29_{10}	0010_2 1001_2
Step 2	29_{10}	00101001_{BCD}

Result

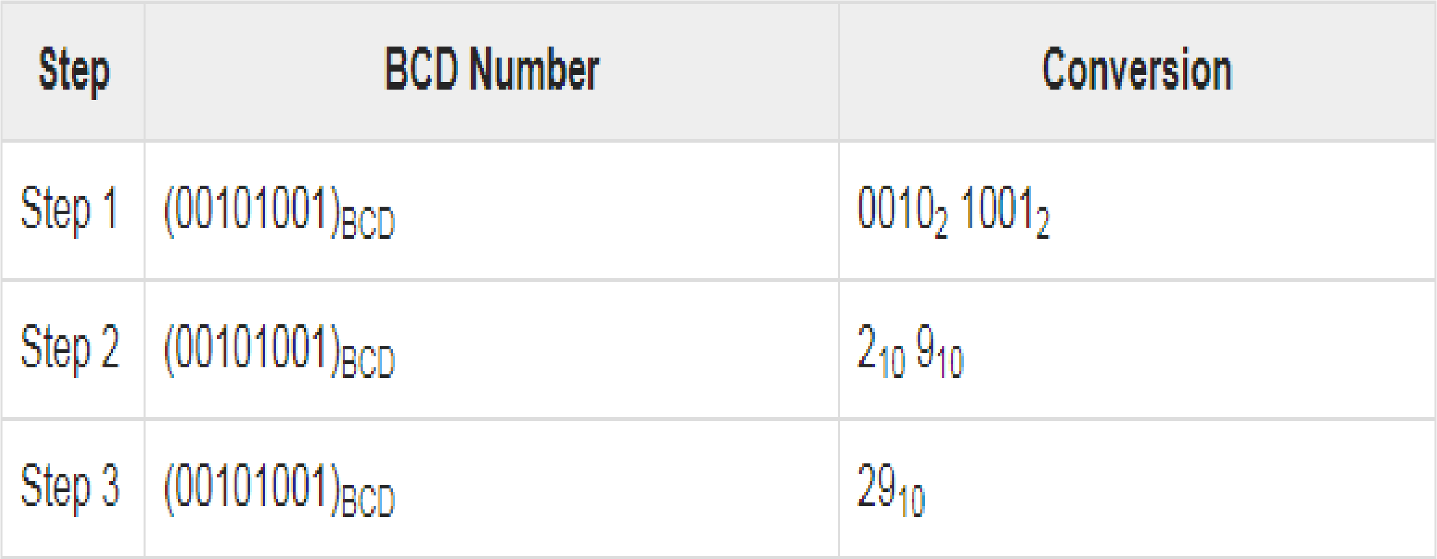
$(11101)_2$ = $(00101001)_{BCD}$

BCD TO BINARY CONVERSION

- **Step 1** -- Convert the BCD number to decimal.
- **Step 2** -- Convert decimal to binary.

Example – convert $(00101001)_{BCD}$ to Binary.

Step 1 - Convert to BCD



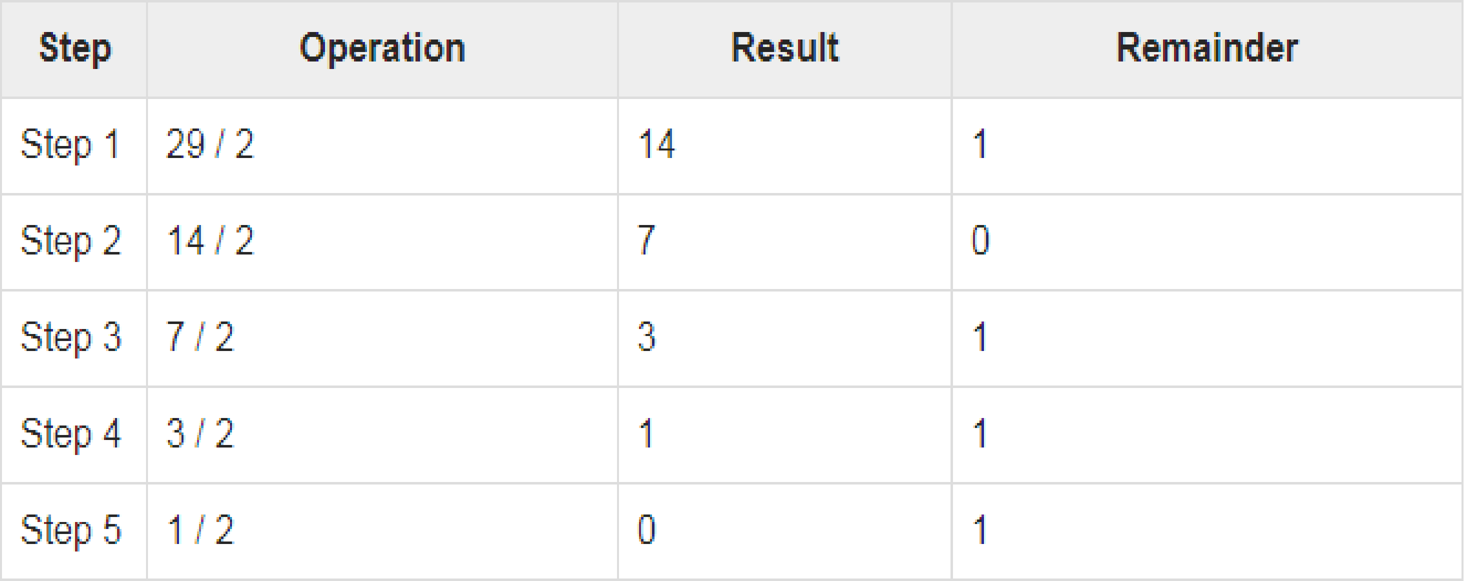
Step	BCD Number	Conversion
Step 1	$(00101001)_{BCD}$	0010_2 1001_2
Step 2	$(00101001)_{BCD}$	2_{10} 9_{10}
Step 3	$(00101001)_{BCD}$	29_{10}

BCD TO BINARY CONVERSION

- Step 2 - Convert to Binary

Use long division method for decimal to binary conversion.

Decimal Number − 29_{10}

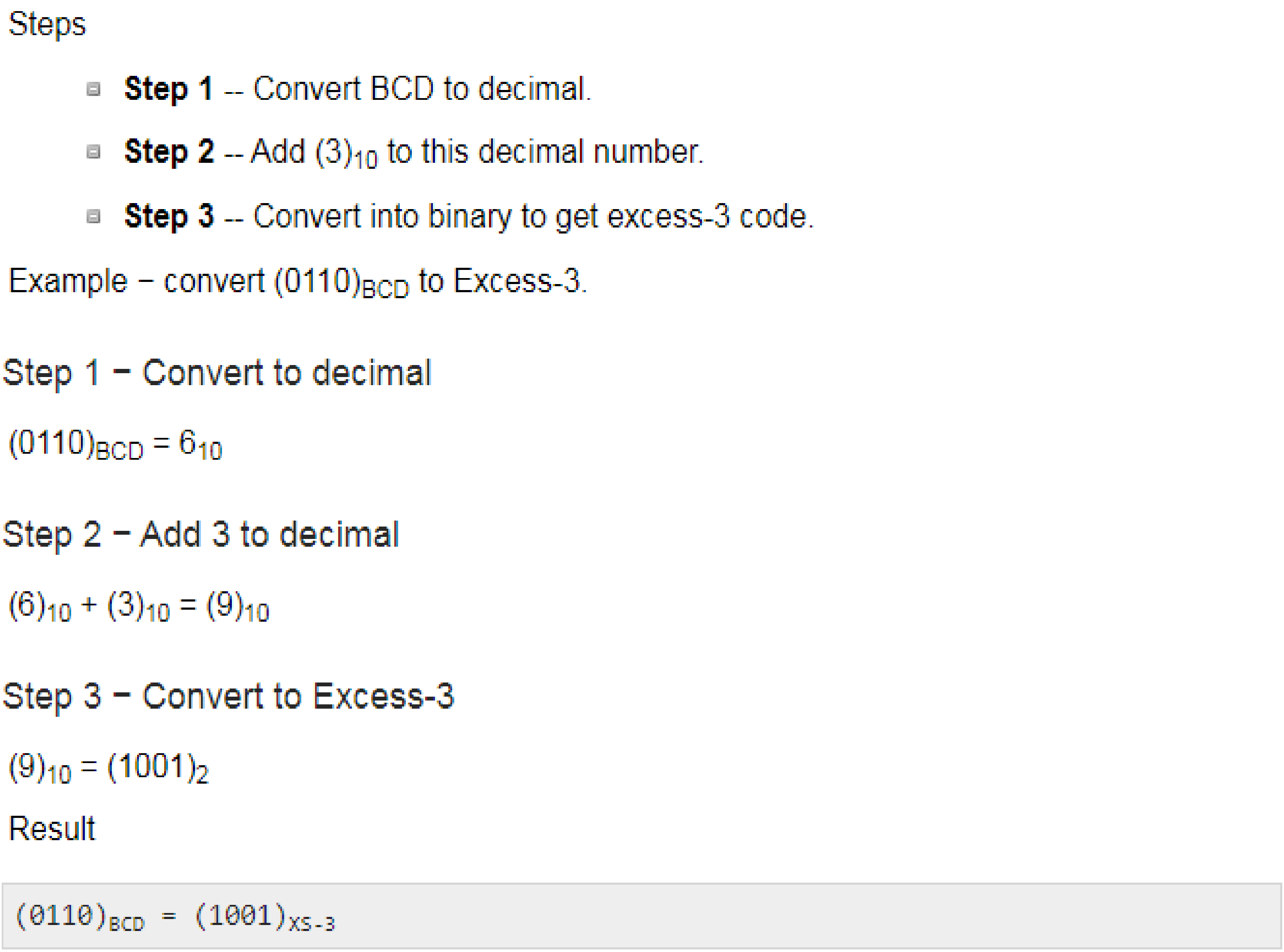


Step	Operation	Result	Remainder
Step 1	29 / 2	14	1
Step 2	14 / 2	7	0
Step 3	7 / 2	3	1
Step 4	3 / 2	1	1
Step 5	1 / 2	0	1

Result

- $(00101001)_{BCD} = (11101)_2$

BCD TO EXCESS-3



Steps

- **Step 1** -- Convert BCD to decimal.
- **Step 2** -- Add $(3)_{10}$ to this decimal number.
- **Step 3** -- Convert into binary to get excess-3 code.

Example – convert $(0110)_{BCD}$ to Excess-3.

Step 1 − Convert to decimal

$(0110)_{BCD} = 6_{10}$

Step 2 − Add 3 to decimal

$(6)_{10} + (3)_{10} = (9)_{10}$

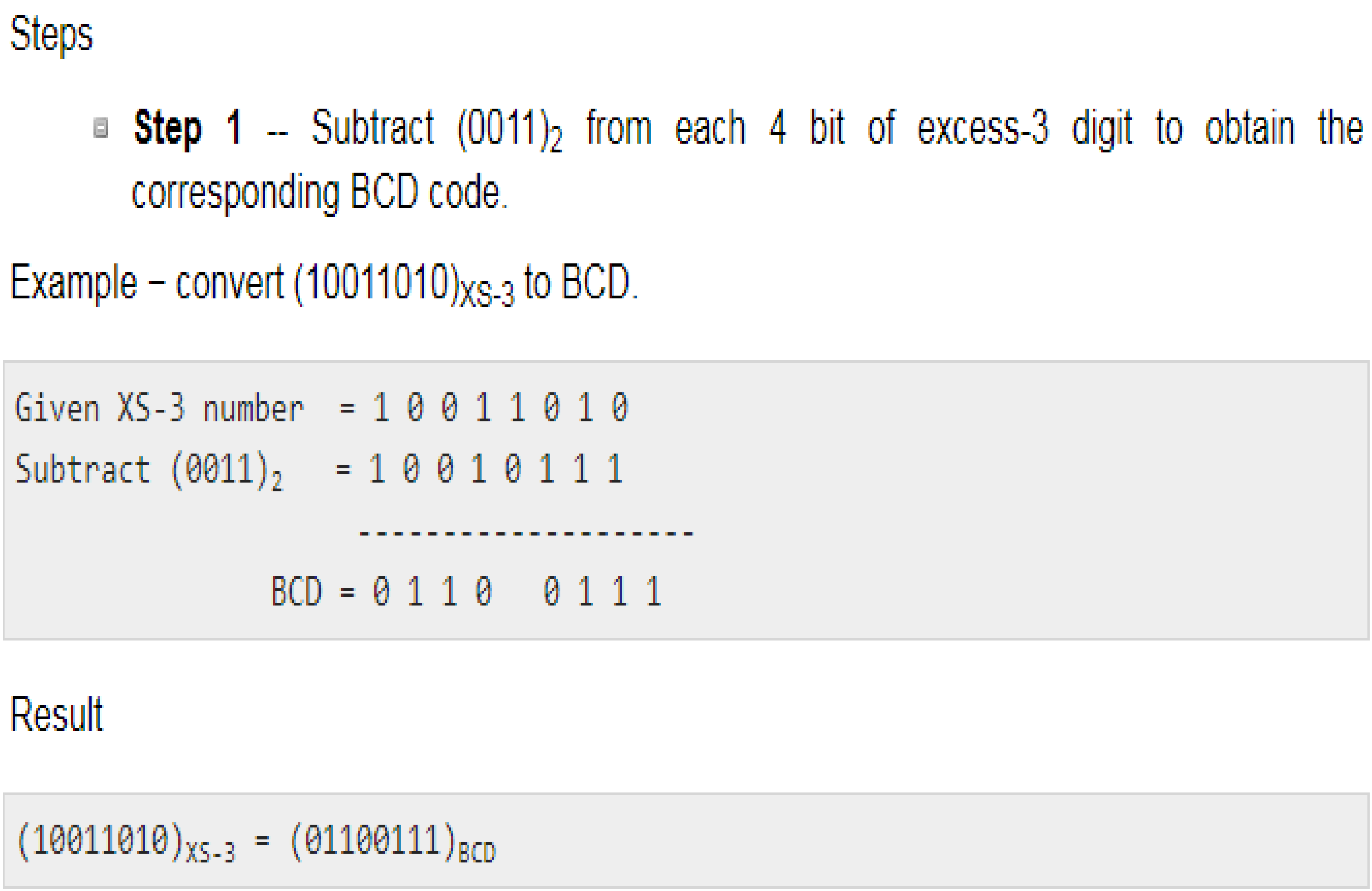
Step 3 − Convert to Excess-3

$(9)_{10} = (1001)_2$

Result

```
(0110)BCD = (1001)XS-3
```

EXCESS-3 TO BCD



Steps

- **Step 1** -- Subtract $(0011)_2$ from each 4 bit of excess-3 digit to obtain the corresponding BCD code.

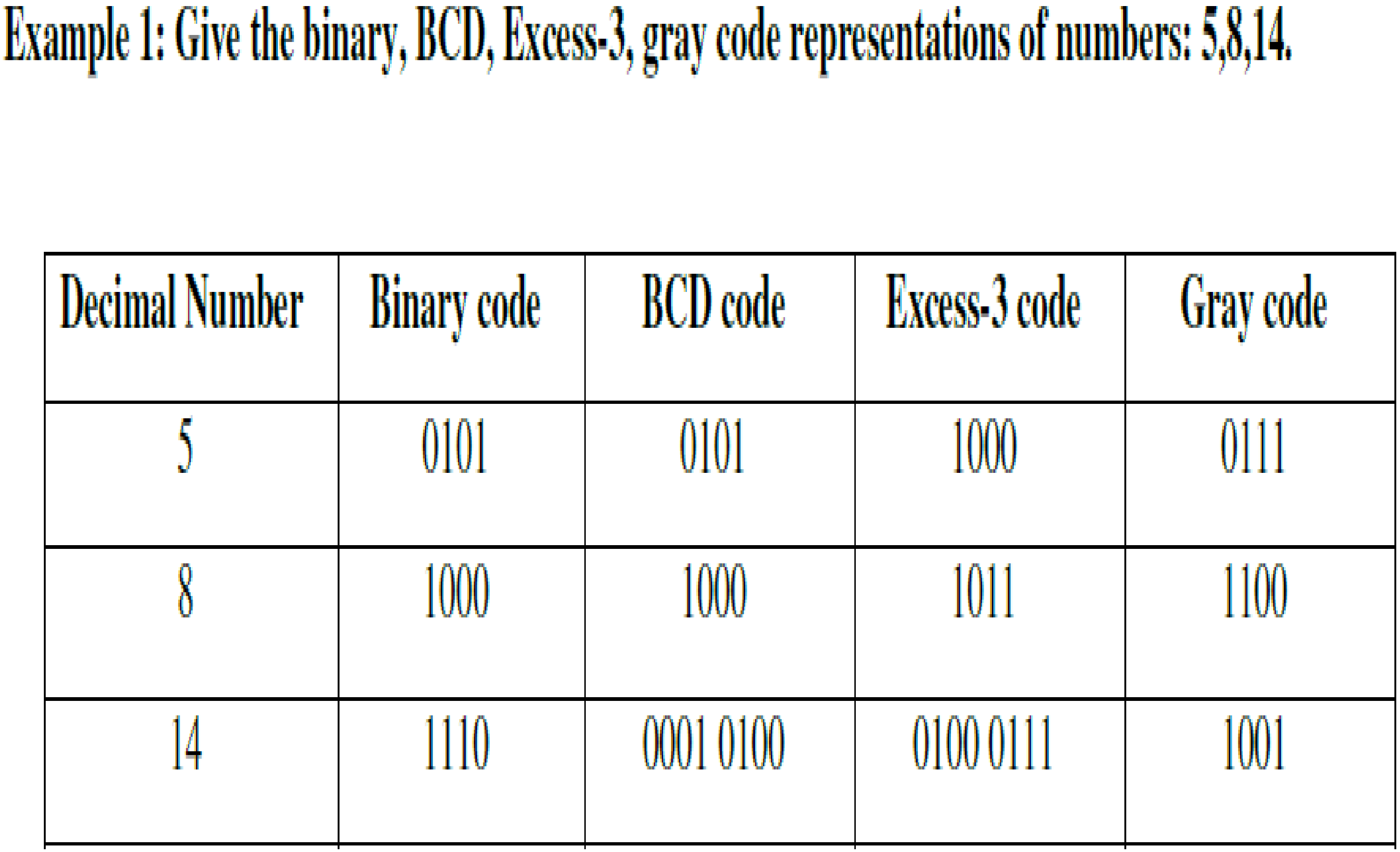
Example – convert $(10011010)_{XS-3}$ to BCD.

```
Given XS-3 number  = 1 0 0 1 1 0 1 0
Subtract (0011)₂   = 1 0 0 1 0 1 1 1
                    --------------------
             BCD = 0 1 1 0   0 1 1 1
```

Result

$(10011010)_{XS-3} = (01100111)_{BCD}$

BINARY CODE CONVERSION

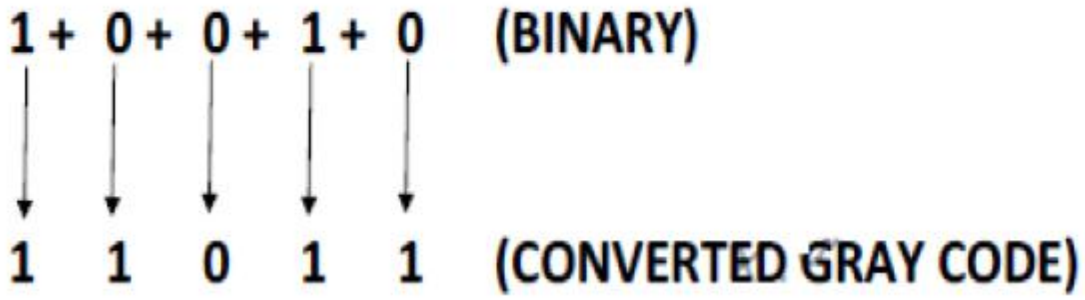


Example 1: Give the binary, BCD, Excess-3, gray code representations of numbers: 5,8,14.

Decimal Number	Binary code	BCD code	Excess-3 code	Gray code
5	0101	0101	1000	0111
8	1000	1000	1011	1100
14	1110	0001 0100	0100 0111	1001

BINARY CODE CONVERSION

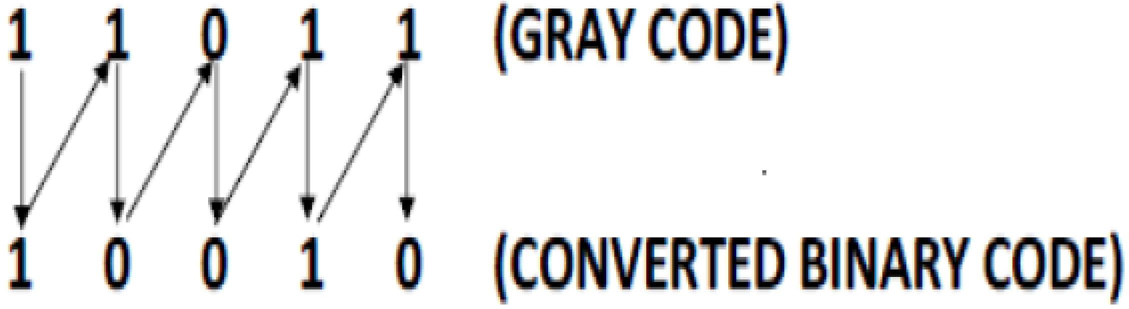
Example 2: Binary To Gray Code Conversion



1 + 0 + 0 + 1 + 0 (BINARY)

1 1 0 1 1 (CONVERTED GRAY CODE)

Example 3:Gray code to Binary code



1 1 0 1 1 (GRAY CODE)

1 0 0 1 0 (CONVERTED BINARY CODE)

BOOLEAN ALGEBRA

- Boolean algebra is an algebraic structure defined by a set of elements, B, together with two binary operators, + and., provider that the following postulates are satisfied.

Principle of Duality

- It states that every algebraic expression is deducible from the postulates of Boolean algebra, and it remains valid if the operators & identity elements are interchanged.
- 1. Interchanging the OR and AND operations of the expression.
- 2. Interchanging the 0 and 1 elements of the expression.
- 3. Not changing the form of the variables.

BOOLEAN THEOREM

- The theorems of Boolean algebra can be used to simplify many a complex Boolean expression and also to transform the given expression into a more useful and meaningful equivalent expression.
- The theorems are presented as pairs, with the two theorems in a given pair being the dual of each other.
- These theorems can be very easily verified by the method of perfect induction.

T1: Commutative Law

(a) $A + B = B + A$

(b) $A\,B = B\,A$

T2: Associative Law

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

(b) $(A\,B)\,C = A\,(B\,C)$

BOOLEAN THEOREM

T3: Distributive Law

(a) A (B + C) = A B + A C

(b) A + (B C) = (A + B) (A + C)

T4: Identity Law

(a) A + A = A

(b) A A = A

T5: Negation Law

$(\overline{A}) = \overline{A}$ and $(\overline{\overline{A}}) = A$

T6: Redundancy

(a) A + A B = A

(b) A (A + B) = A

Contd,.

T7: Operations with ‘0’ & ‘1’

(a) $0 + A = A$

(b) $1\,A = A$

(c) $1 + A = 1$

(d) $0\,A = 0$

T8 : Complement laws

(a) $\bar{A} + A = 1$

(b) $\bar{A}.A = 0$

T9 : (a) $A + \bar{A}B = A + B$

(b) $\mathrm{A}.(\bar{A} + B) = A.B$

Contd,.

T10: De Morgan's Theorem

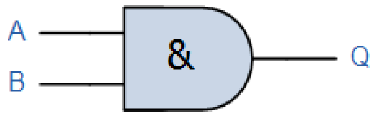
It States that —The complement of the sum of the variables is equal to the product of the complement of each variable This theorem may be expressed by the following Boolean expression.

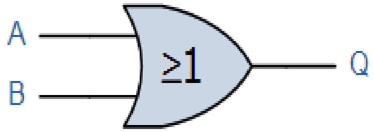
$$(\overline{A + B}) = \bar{A}.\bar{B}$$

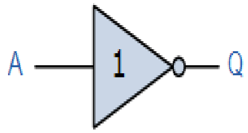
- It states that the —Complement of the product of variables is equal to the sum of complements of each individual variable. Boolean expression for this theorem is

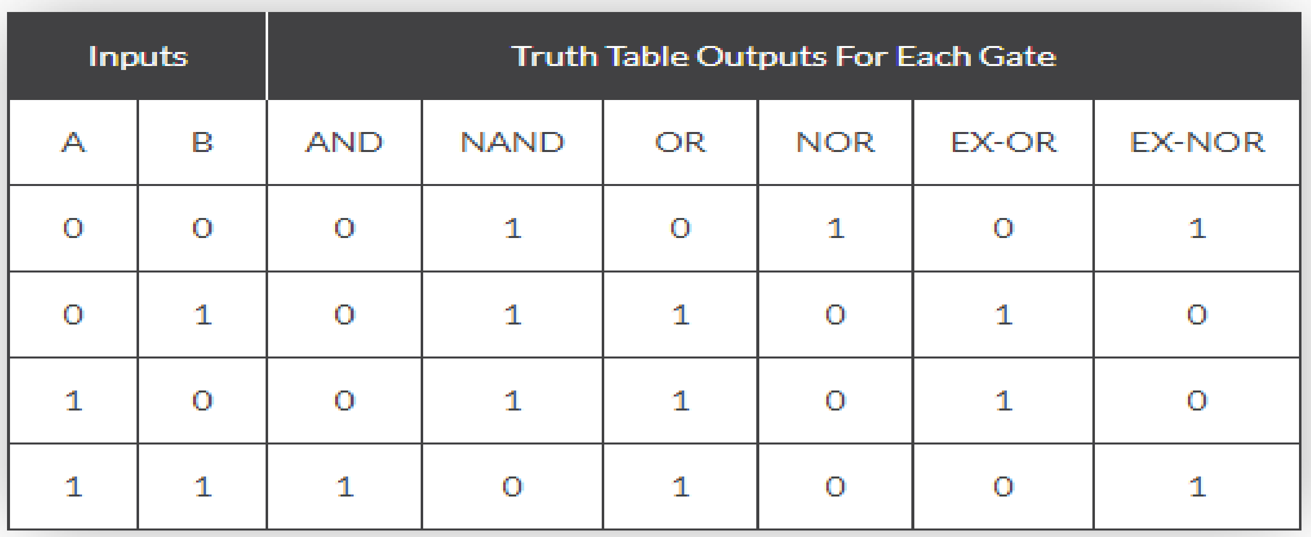
$$(\overline{AB}) = \bar{A} + \bar{B}$$

GATES

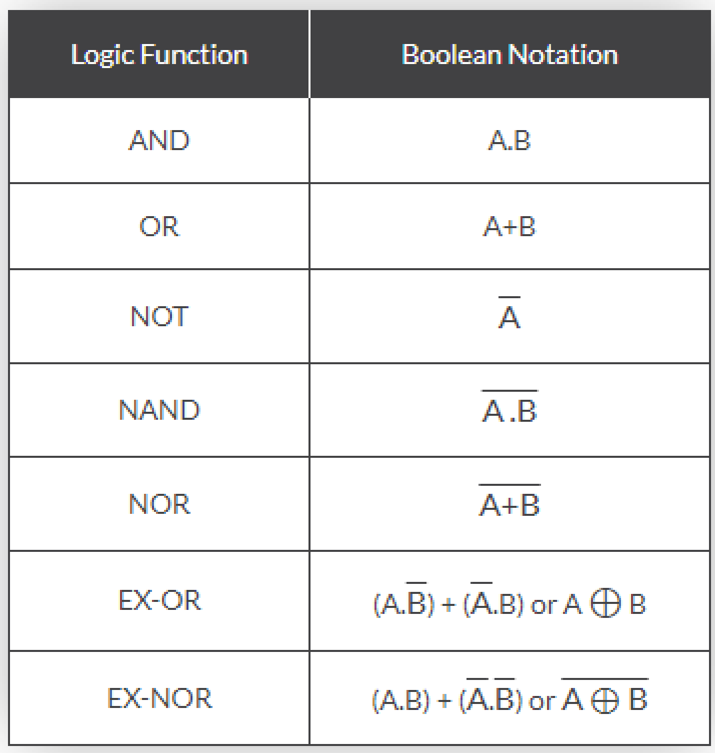
Symbol	Truth Table		
 2-input AND Gate	A	B	Q
	0	0	0
	0	1	0
	1	0	0
	1	1	1
Boolean Expression Q = A.B	Read as A AND B gives Q		

Symbol	Truth Table		
 2-input OR Gate	A	B	Q
	0	0	0
	0	1	1
	1	0	1
	1	1	1
Boolean Expression Q = A+B	Read as A OR B gives Q		

Symbol	Truth Table	
 Inverter or NOT Gate	A	Q
	0	1
	1	0
Boolean Expression Q = NOT A or $\overline{A}$	Read as inversion of A gives Q	



Inputs		Truth Table Outputs For Each Gate					
A	B	AND	NAND	OR	NOR	EX-OR	EX-NOR
0	0	0	1	0	1	0	1
0	1	0	1	1	0	1	0
1	0	0	1	1	0	1	0
1	1	1	0	1	0	0	1



Logic Function	Boolean Notation
AND	$A.B$
OR	$A+B$
NOT	$\overline{A}$
NAND	$\overline{A.B}$
NOR	$\overline{A+B}$
EX-OR	$(A.\overline{B}) + (\overline{A}.B)$ or $A \oplus B$
EX-NOR	$(A.B) + (\overline{A}.\overline{B})$ or $\overline{A \oplus B}$

VERIFICATION OF DEMORGAN’S LAW

De Morgan's First Theorem states:

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

De Morgan's Second Theorem states:

- The complement of sum of variables is equal to the product of the complements of the dividable variables

- **FIRST LAW**

$\overline{A.B} = \overline{|A} + \overline{B}$

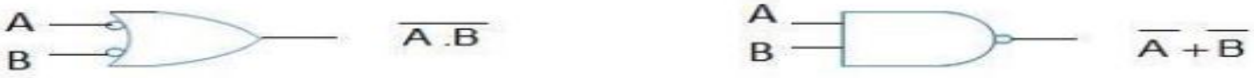


Figure: De Morgan’s First Law

- **SECOND LAW**

$\overline{A + B} = \overline{A}.\overline{B}$

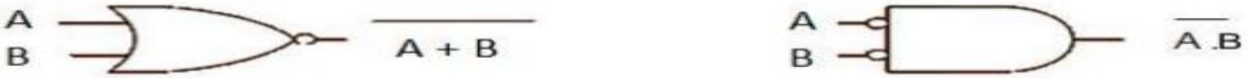


Figure: De Morgan’s Second Law

PROBLEMS

Example 1: Using theorems, find A+A’B.

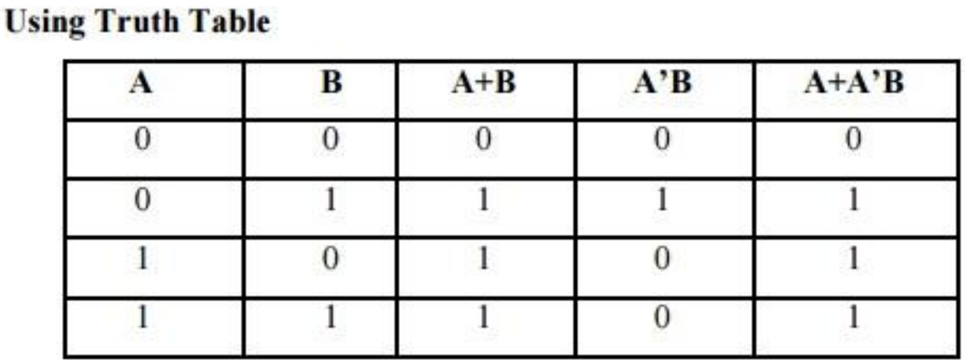
$$A + A'B = A1 + A'B$$

$$= A(1 + B) + A'B$$

$$= A + AB + A'B$$

$$= A + B(A + A')$$

$$= A + B$$



Using Truth Table

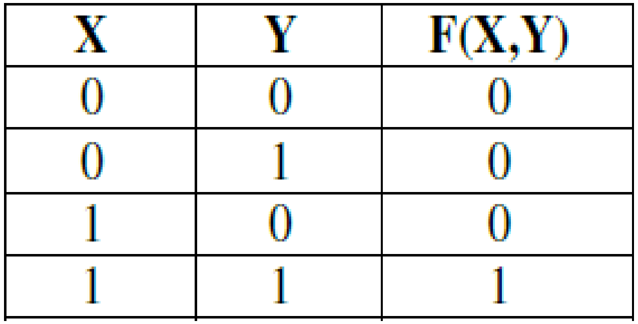
A	B	A+B	A’B	A+A’B
0	0	0	0	0
0	1	1	1	1
1	0	1	0	1
1	1	1	0	1

ORDER OF PRECEDENCE

- NOT operations have the highest precedence, followed by AND operations, followed by OR operations.
- Brackets can be used as with other forms of algebra.
- e.g. X.Y + Z and X.(Y + Z) are not the same function.
- Truth tables are a means of representing the results of a logic function using a table.

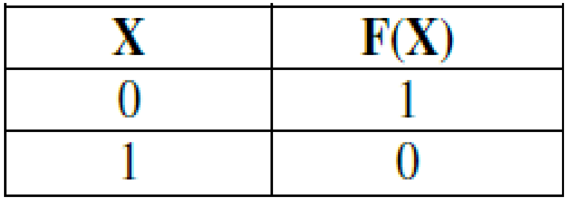
Contd,.

AND



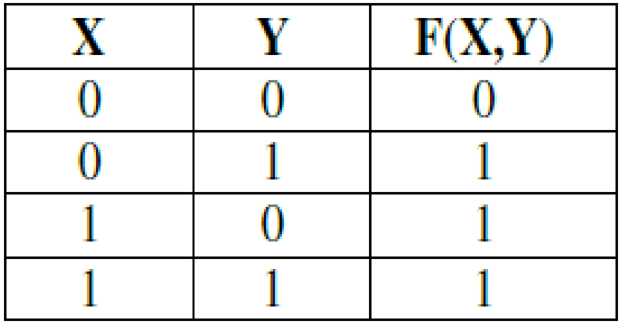
X	Y	F(X,Y)
0	0	0
0	1	0
1	0	0
1	1	1

NOT



X	F(X)
0	1
1	0

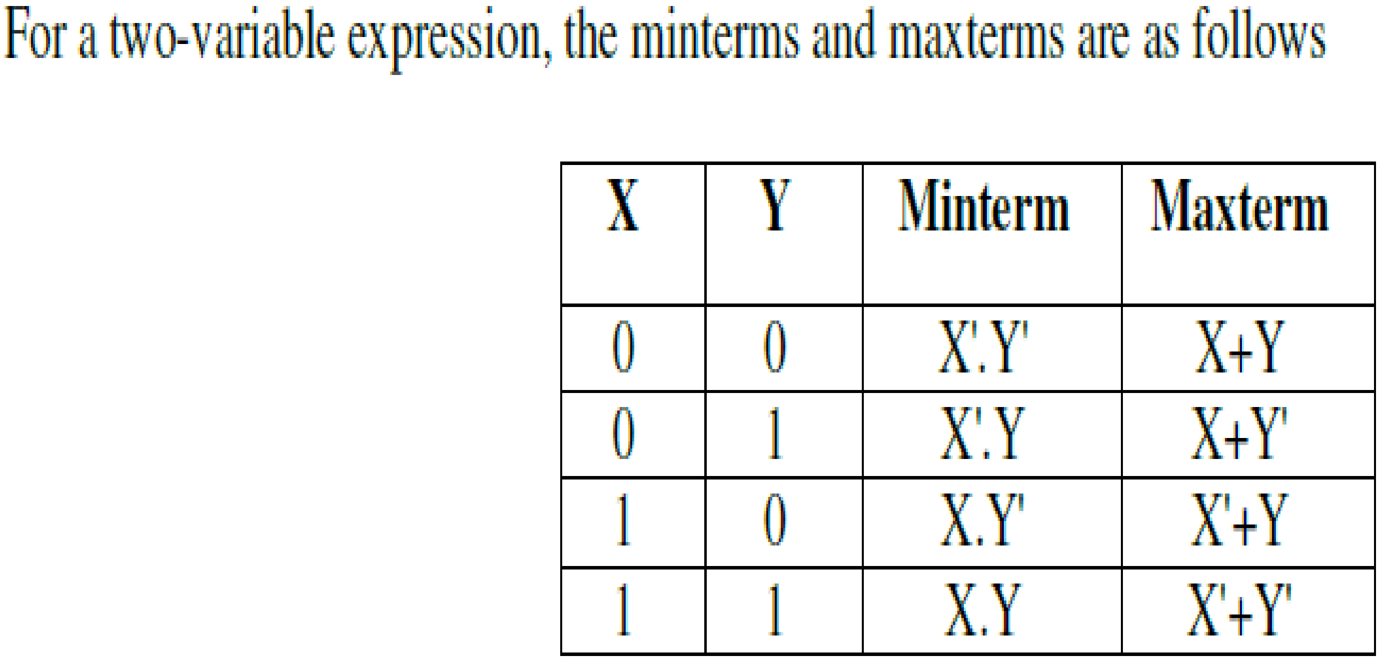
OR



X	Y	F(X,Y)
0	0	0
0	1	1
1	0	1
1	1	1

MINTERM AND MAXTERM

- A minterm is the product of N distinct literals where each literal occurs exactly once.
- A maxterm is the sum of N distinct literals where each literal occurs exactly once.

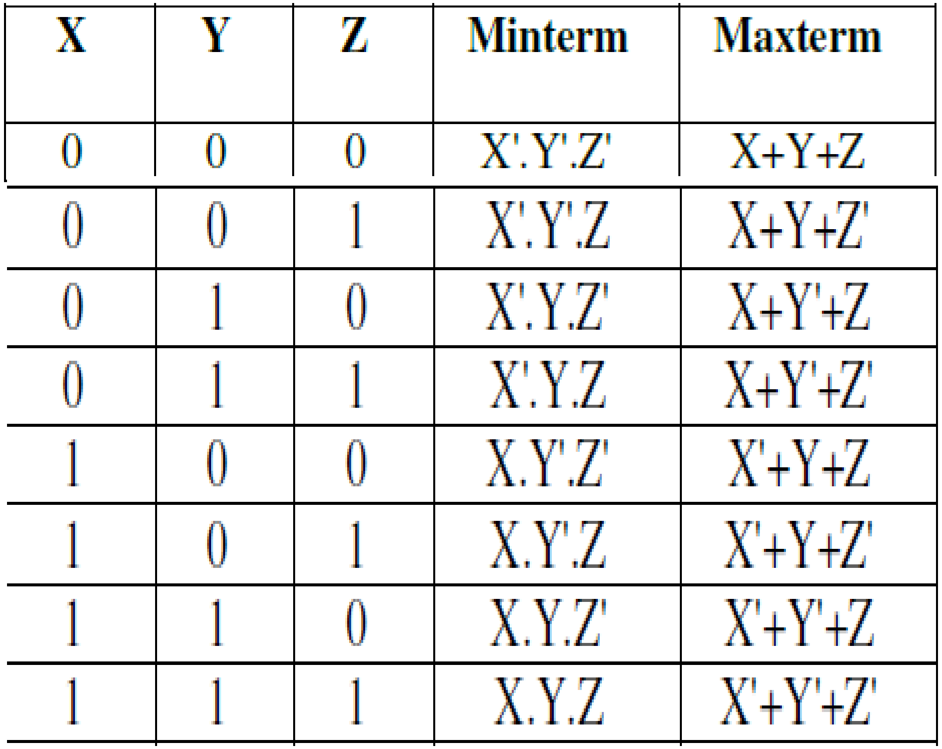


For a two-variable expression, the minterms and maxterms are as follows

X	Y	Minterm	Maxterm
0	0	X'.Y'	X+Y
0	1	X'.Y	X+Y'
1	0	X.Y'	X'+Y
1	1	X.Y	X'+Y'

Contd,.

For a three-variable expression, the minterms and maxterms are as follows



X	Y	Z	Minterm	Maxterm
0	0	0	X'.Y'.Z'	X+Y+Z
0	0	1	X'.Y'.Z	X+Y+Z'
0	1	0	X'.Y.Z'	X+Y'+Z
0	1	1	X'.Y.Z	X+Y'+Z'
1	0	0	X.Y'.Z'	X'+Y+Z
1	0	1	X.Y'.Z	X'+Y+Z'
1	1	0	X.Y.Z'	X'+Y'+Z
1	1	1	X.Y.Z	X'+Y'+Z'

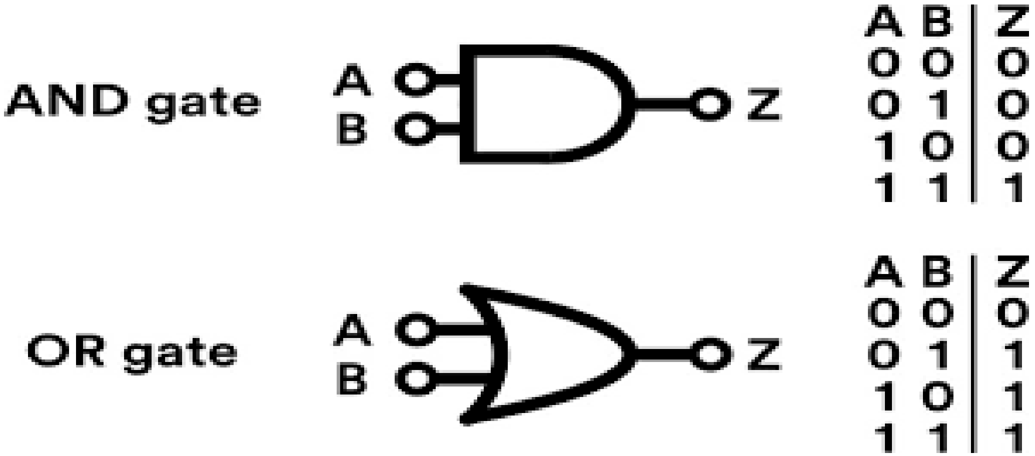
BOOLEAN FUNCTION

- A Boolean expression is an expression which consists of variables, constants and logical operators which results in true or false.
- A Boolean function is an algebraic form of Boolean expression.
- The different ways of representing a Boolean function is shown below.

1. Sum-of-Products (SOP) Form
2. Product-of-sums (POS) form
3.Canonical forms

SOP

- The Sum of Product (SOP) expression comes from the fact that two or more products (AND) are summed (OR) together.
- The outputs from two or more AND gates are connected to the input of an OR gate so that they are effectively OR'ed together to create the final AND-OR logical output.

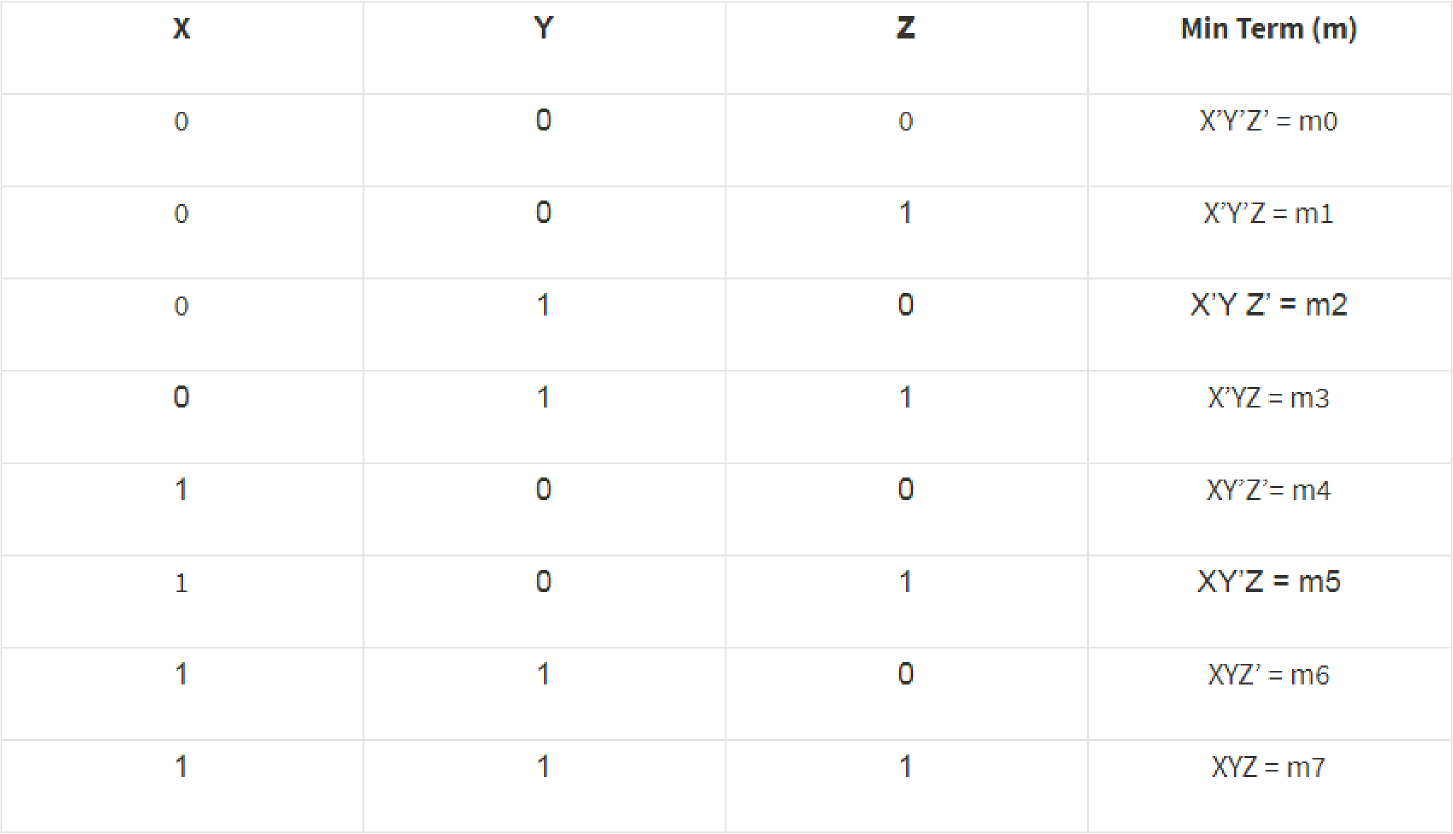


A	B	Z
0	0	0
0	1	0
1	0	0
1	1	1

A	B	Z
0	0	0
0	1	1
1	0	1
1	1	1

SOP

- The short form of the sum of the product is SOP, and it is one kind of Boolean algebra expression.
- The min term can be defined as, when the minimum combinations of inputs are high then the output will be high.



X	Y	Z	Min Term (m)
0	0	0	X’Y’Z’ = m0
0	0	1	X’Y’Z = m1
0	1	0	X’Y Z’ = m2
0	1	1	X’YZ = m3
1	0	0	XY’Z’= m4
1	0	1	XY’Z = m5
1	1	0	XYZ’ = m6
1	1	1	XYZ = m7

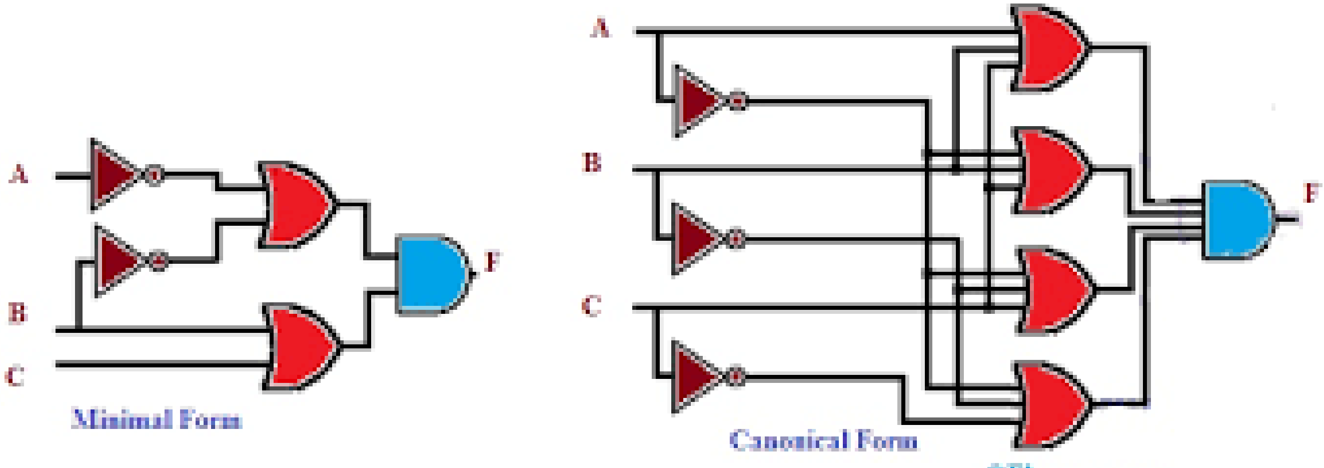
SOP

The sum of products is available in three different forms which include the following.

1). Canonical Sum of Products

2). Non-Canonical Sum of Products

3). Minimal Sum of Products



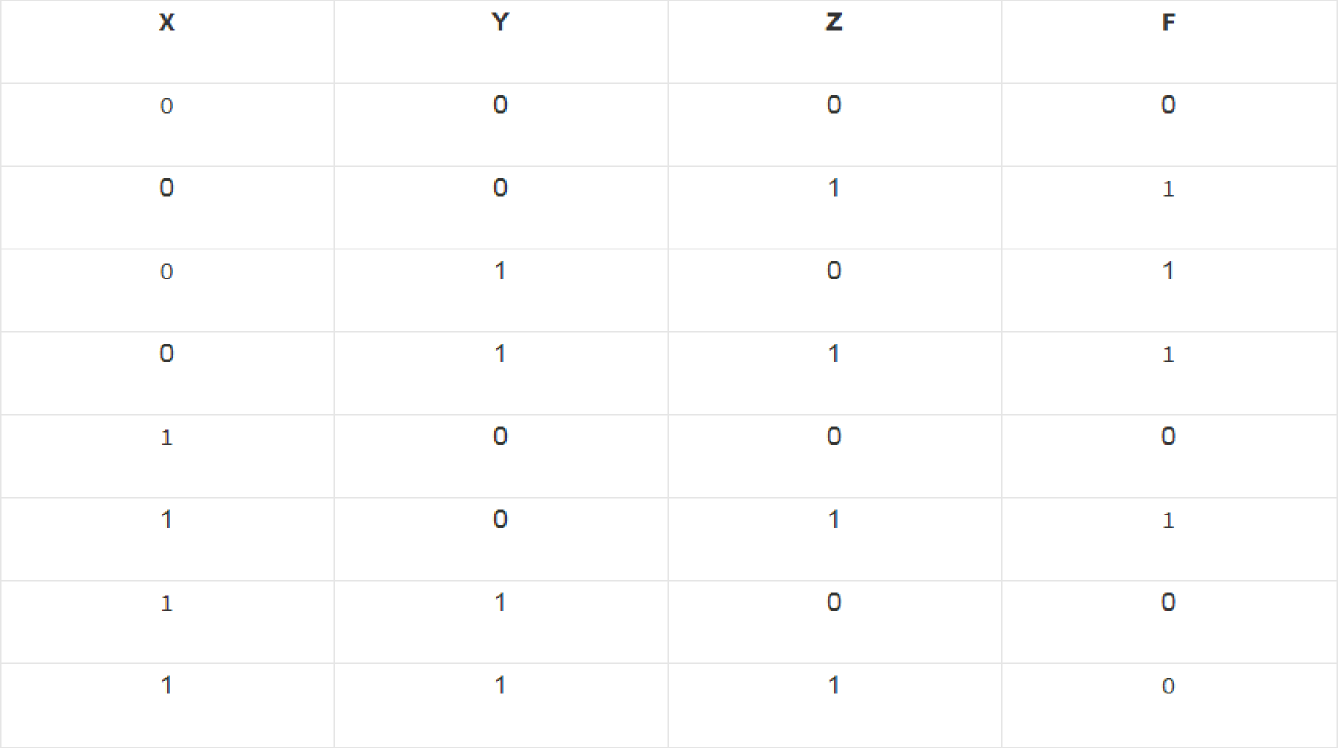
Minimal Form

Canonical Form

©Elprocus.com

CANONICAL SUM OF PRODUCTS

- This is a normal form of SOP.
- The expression of the canonical SOP is denoted with sign summation (∑), and the minterms in the bracket are taken when the output is true.
- The truth table of the canonical sum of the product is shown below.



X	Y	Z	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

Contd,.

- For the above table, the **canonical SOP form** can be written as

  $$F = \sum (m1, m2, m3, m5)$$

  By expanding the above summation we can get the following function.

  $$F = m1 + m2 + m3 + m5$$

  By substituting the minterms in the above equation we can get the below expression

  $$F = X'Y'Z + X'YZ' + X'YZ + XY'Z$$

- The product term of the canonical form includes both complemented and non-complimented inputs

NON-CANONICAL SUM OF PRODUCTS

- In the non-canonical sum of product form, the product terms are simplified. For example, let’s take the above canonical expression
- **$F = X'Y'Z + X'YZ' + X'YZ + XY'Z$**

  $F = X'Y'Z + X'Y\,(Z'+Z) + XY'Z$

  Here **$Z'+Z = 1$** (Standard function)

  $F = X'Y'Z + X'Y\,(1) + XY'Z$

  $F = X'Y'Z + X'Y + XY'Z$

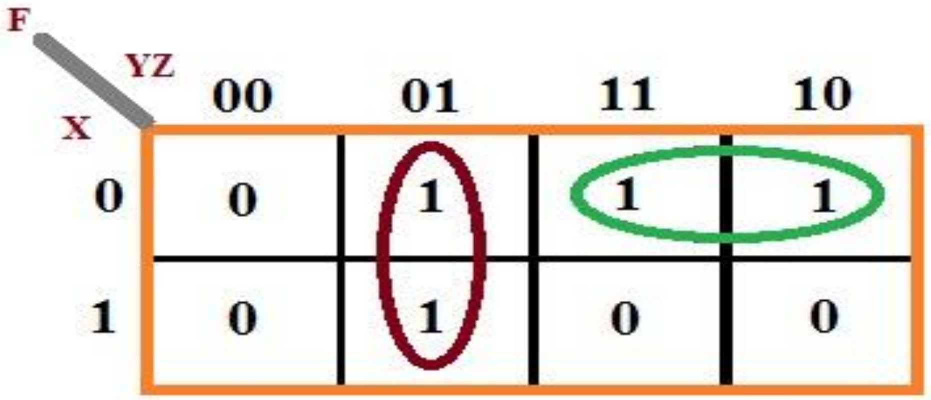
  This is still in the form of SOP, but it is the non-canonical form

MINIMAL SUM OF PRODUCTS

- This is the most simplified expression of the sum of the product, and It is also a type of non-canonical.
- This type of can is made simplified with the Boolean algebraic theorem although it is simply done by using **K-map (Karnaugh map)**.
- This form is chosen due to the number of input lines & gates are used in this is minimum.

K-MAP

- Let’s take an example of canonical form function, and the minimal **Sum of Products K map** is



F X \ YZ	00	01	11	10
0	0	1	1	1
1	0	1	0	0

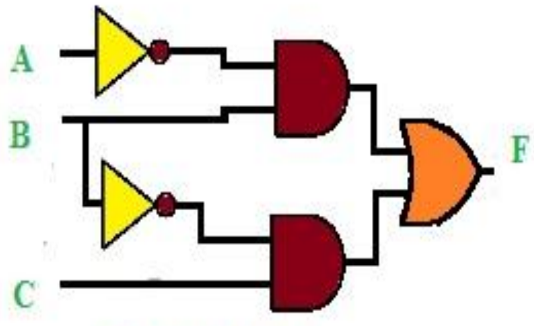
©Elprocus.com

The expression of this based on the K-map will be

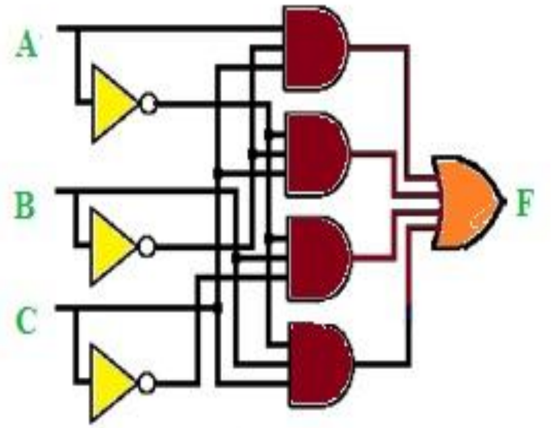
F = Y’Z + X’Y

SCHEMATIC DESIGN

- The expression of the sum of product executes two-level AND-OR design, and this design requires a collection of AND gates and one OR gate.



Minimal Form



Canonical Form

©Elprocus.com

MINIMAL TO CANONICAL SOP FORM

- Conversion from minimal or any sort of non-canonical form to canonical form is very simple.
- Example of conversion for minimal SOP form is given below.

$$\mathbf{F = \overline{A}B + \overline{B}C} \qquad \text{(Minimal SOP form)}$$

- The term $\mathbf{\overline{A}B}$ is missing input C. So we will multiply $\mathbf{\overline{A}B}$ with $\mathbf{(C+\overline{C})}$ because $\mathbf{(C+\overline{C} = 1)}$. The term $\mathbf{\overline{B}C}$ is missing input **A**. so it will be multiplied with $\mathbf{(A+\overline{A})}$

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

$$\mathbf{F = \overline{A}BC + \overline{A}B\overline{C} + A\overline{B}C + \overline{A}\overline{B}C} \text{ (Canonical SOP)}$$

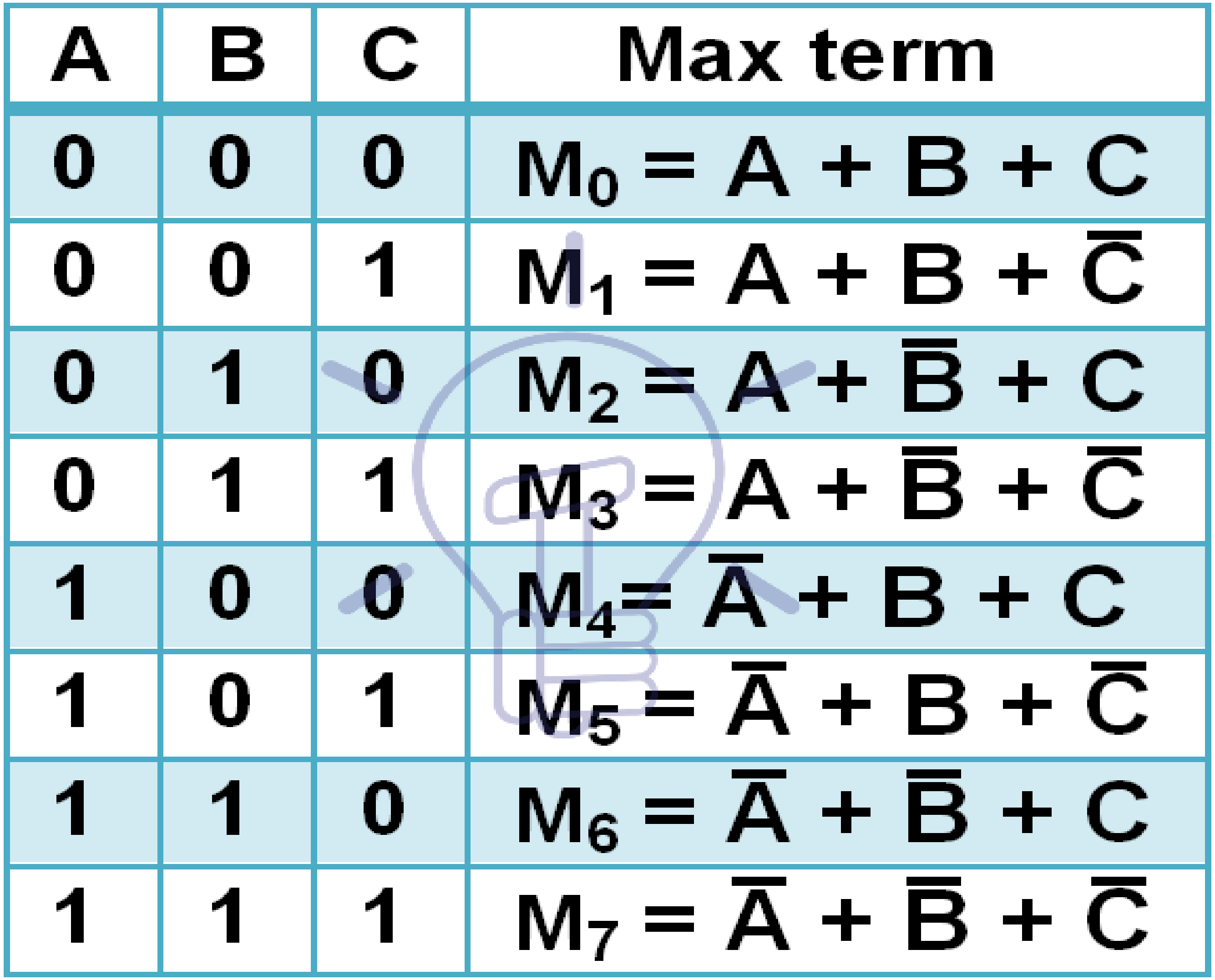
PRODUCT OF SUM(POS)

- The product of Sum form is a form in which products of different sum terms of inputs are taken.
- These are not arithmetic product and sum but they are logical Boolean AND and OR respectively.

Max Term

- Maxterm means the term or expression that is true for a maximum number of input combinations or that is false for only one combination of inputs.

MAX TERM



A	B	C	Max term
0	0	0	$M_0 = A + B + C$
0	0	1	$M_1 = A + B + \overline{C}$
0	1	0	$M_2 = A + \overline{B} + C$
0	1	1	$M_3 = A + \overline{B} + \overline{C}$
1	0	0	$M_4 = \overline{A} + B + C$
1	0	1	$M_5 = \overline{A} + B + \overline{C}$
1	1	0	$M_6 = \overline{A} + \overline{B} + C$
1	1	1	$M_7 = \overline{A} + \overline{B} + \overline{C}$

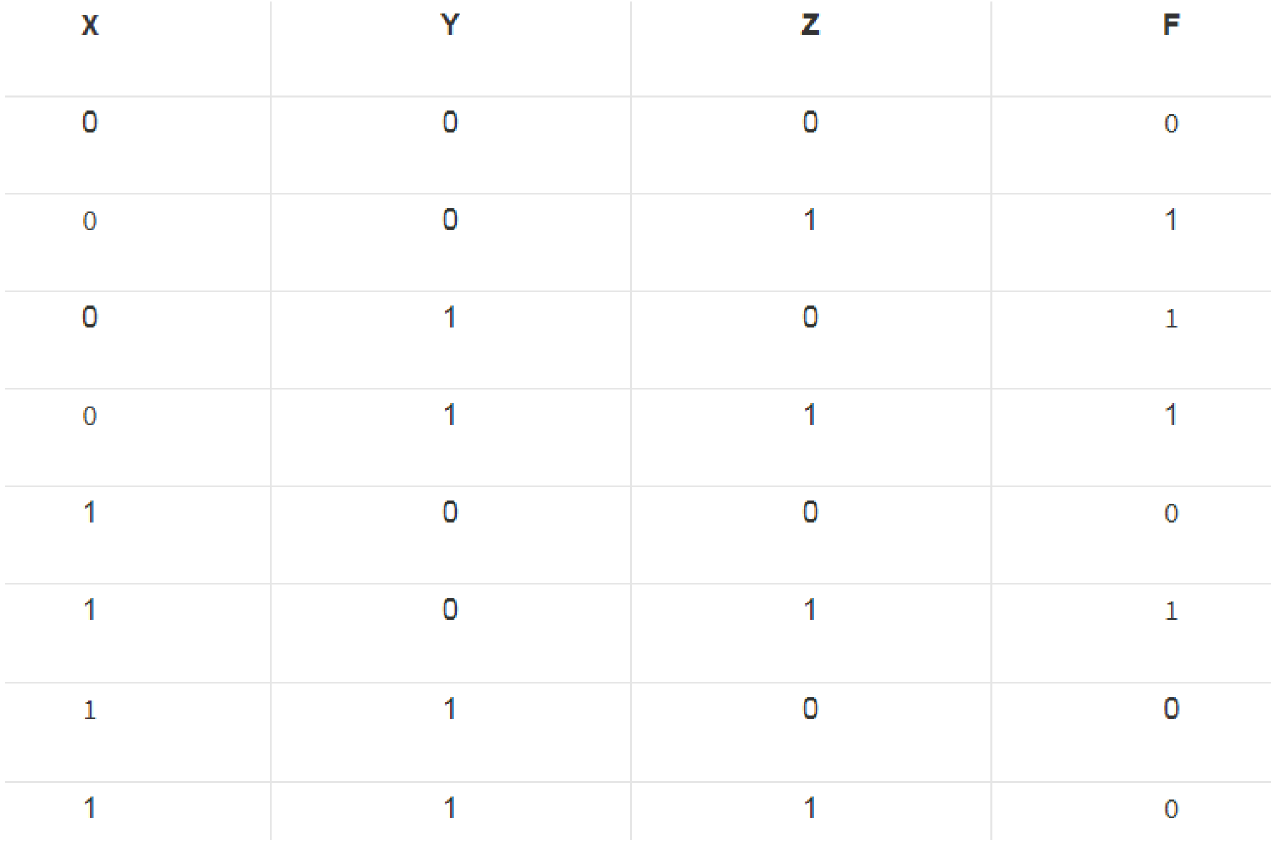
TYPES OF POS

The product of the sum is classified into three types which include the following.

- Canonical Product of Sums
- Non – Canonical Product of Sums
- Minimal Product of Sums

CANONICAL PRODUCT OF SUM

- The canonical POS is also named as a product of max term.
- The expression this is denoted by ∏ and the max terms in the bracket are taken when the output is false.



X	Y	Z	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

Contd,.

- For the above table, the canonical POS can be written as

$$\mathbf{F = \prod (M0, M4, M6, M7)}$$

By expanding the above equation we can get the following function.

$$\mathbf{F = M0, M4, M6, M7}$$

By substituting the max terms in the above equation we can get the below expression

$$\mathbf{F = (X+Y+Z)\ (X'+Y+Z)(X'+Y'+Z)(X'+Y'+Z')}$$

The product term of the canonical form includes both complemented and non-complimented inputs

NON – CANONICAL PRODUCT OF SUM

- The expression of the **product of sum (POS)** is not in normal form is named as non-canonical form. For example, let's take the above expression

$$F = (X+Y+Z)\ (X'+Y+Z)(X'+Y'+Z)(X'+Y'+Z')$$

$$F = (Y+Z)\ (X'+Y+Z)\ (X'+Y'+Z')$$

Similar although reversed terms remove from two Max terms & forms only term to show it here is an instance.

$$= (X+Y+Z)\ (X'+Y+Z)$$

$$= XX'+XY+XZ+X'Y+YY+YZ+X'Z+YZ+ZZ$$

$$= 0+XY+XZ+X'Y+YY+YZ+X'Z+YZ+Z$$

$$= X\ (Y+Z) + X'\ (Y+Z) + Y(1+Z) +Z$$

$$= (Y+Z)\ (X+X') + Y\ (1) +Z$$

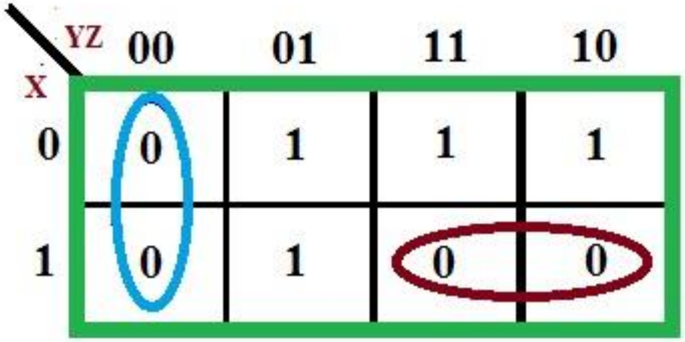
$$= (Y+Z)\ (0) +Y+Z$$

$$= Y+Z$$

The above expression is in the form of non-canonical.

MINIMAL PRODUCT OF SUM

- This type of can is made simplified with the Boolean algebraic theorems although it is simply done by using K-map (Karnaugh map).
- Let’s take an example of canonical form function, and the **Product of sums K map** is



X \ YZ	00	01	11	10
0	0	1	1	1
1	0	1	0	0

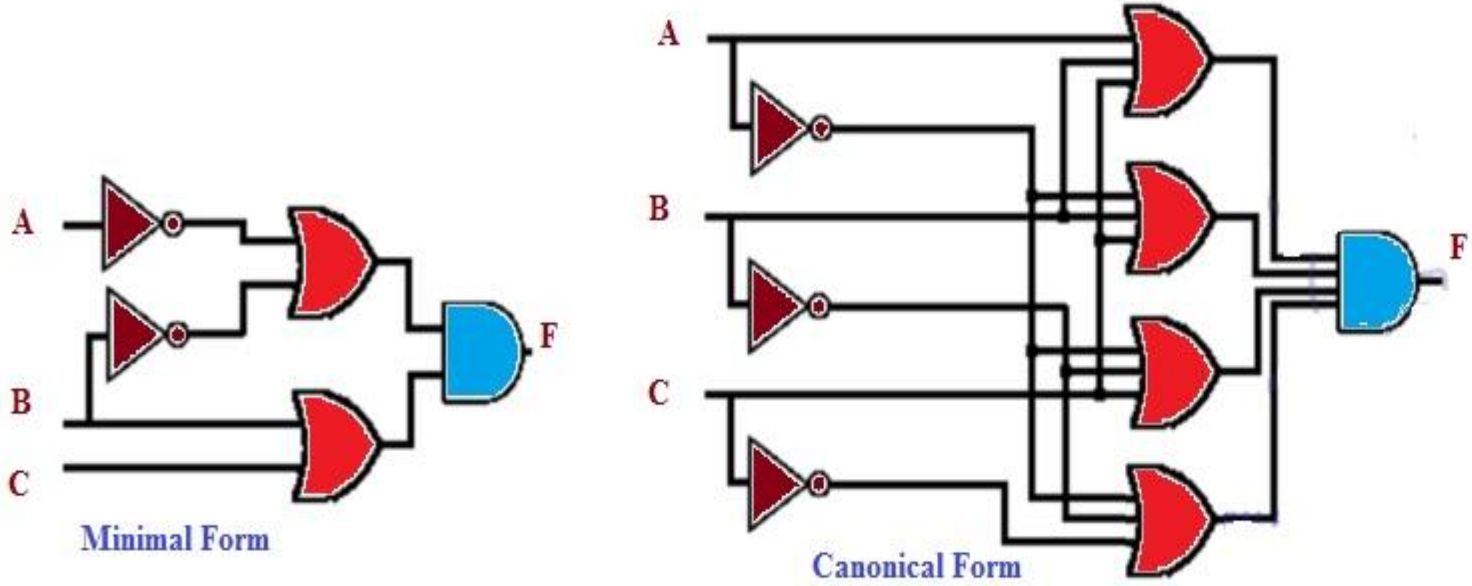
©Elprocus.com

- POS K-mapThe expression of this based on the K-map will be

F = (Y+Z) (X’+Y’)

SCHEMATIC DESIGN

- The expression of the product of the sum executes two levels OR- AND design and this design requires a collection of OR gates and one AND gate.



©Elprocus.com

UNIT-2 COMBINATIONAL CIRCUITS

Combinational Logic

- Logic circuits for digital systems may be combinational or sequential.
- A combinational circuit consists of input variables, logic gates, and output variables.

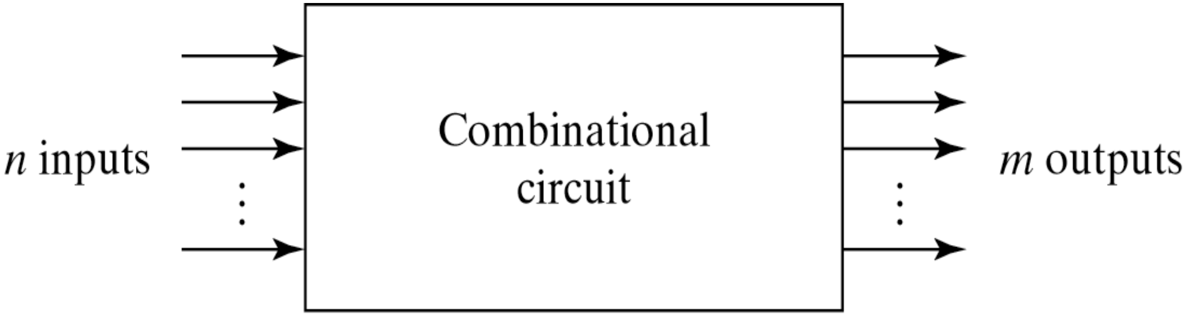


Fig. 4-1 Block Diagram of Combinational Circuit

4-2. Analysis procedure

- To obtain the output Boolean functions from a logic diagram, proceed as follows:

1. Label all gate outputs that are a function of input variables with arbitrary symbols. Determine the Boolean functions for each gate output.

2. Label the gates that are a function of input variables and previously labeled gates with other arbitrary symbols. Find the Boolean functions for these gates.

4-2. Analysis procedure

3. Repeat the process outlined in step 2 until the outputs of the circuit are obtained.

4. By repeated substitution of previously defined functions, obtain the output Boolean functions in terms of input variables.

Example

$F_2 = AB + AC + BC$; $T_1 = A + B + C$; $T_2 = ABC$; $T_3 = F_2'T_1$;

$F_1 = T_3 + T_2$

$F_1 = T_3 + T_2 = F_2'T_1 + ABC = A'BC' + A'B'C + AB'C' + ABC$

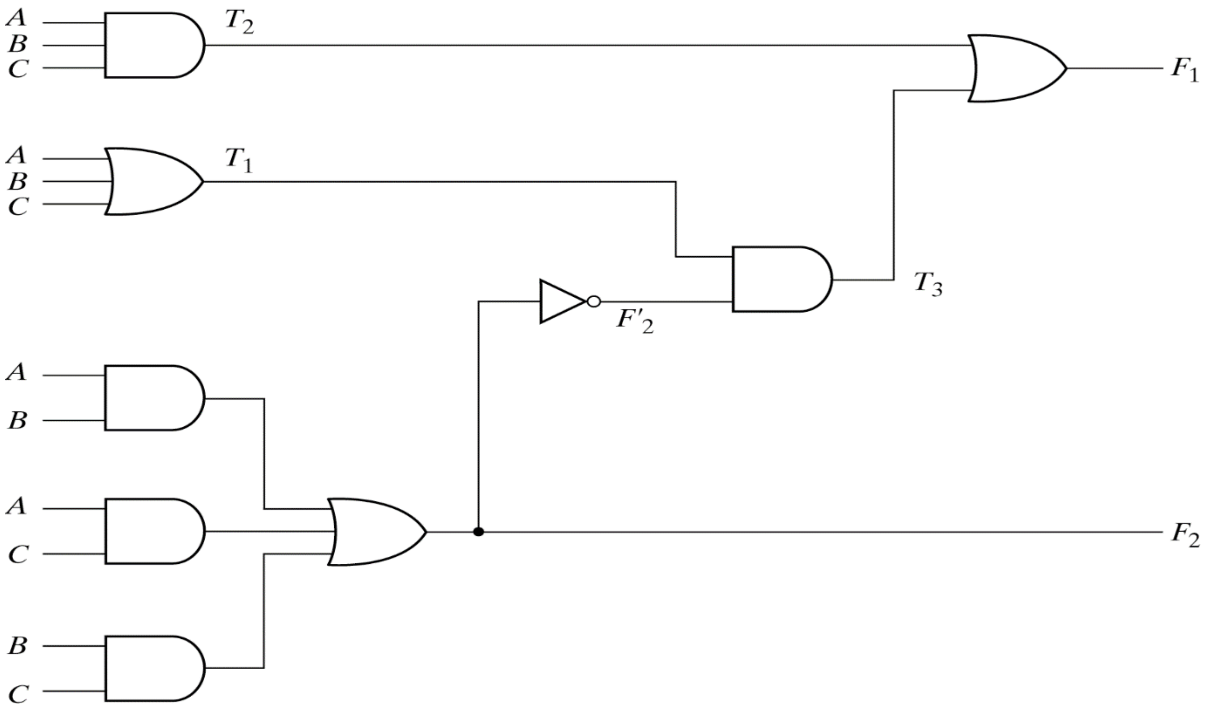


Fig. 4-2 Logic Diagram for Analysis Example

Derive truth table from logic diagram

- We can derive the truth table in Table 4-1 by using the circuit of Fig.4-2.

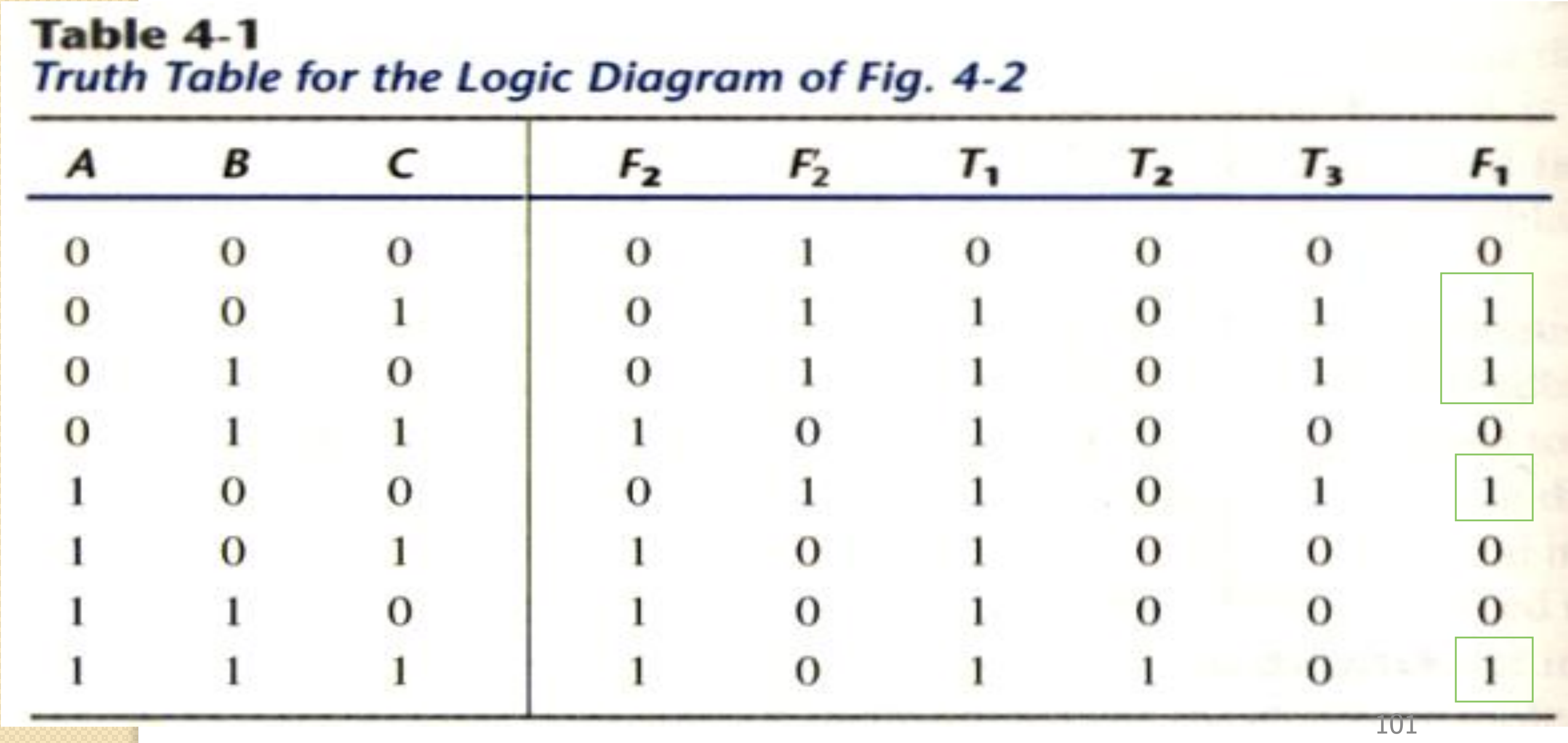


Table 4-1
Truth Table for the Logic Diagram of Fig. 4-2

A	B	C	F_2	F_2'	T_1	T_2	T_3	F_1
0	0	0	0	1	0	0	0	0
0	0	1	0	1	1	0	1	1
0	1	0	0	1	1	0	1	1
0	1	1	1	0	1	0	0	0
1	0	0	0	1	1	0	1	1
1	0	1	1	0	1	0	0	0
1	1	0	1	0	1	0	0	0
1	1	1	1	0	1	1	0	1

4-3. Design procedure

1. Table4-2 is a Code-Conversion example, first, we can list the relation of the BCD and Excess-3 codes in the truth table.

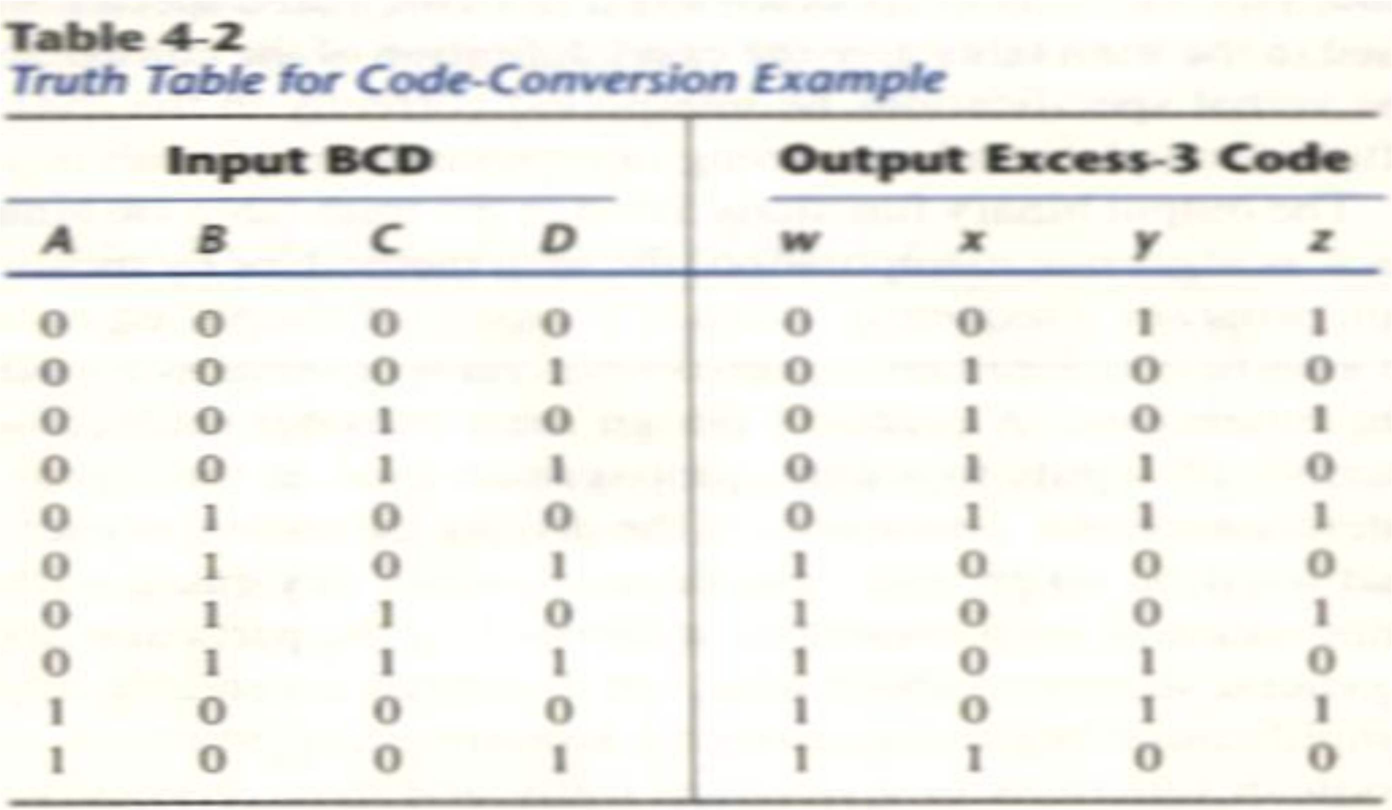
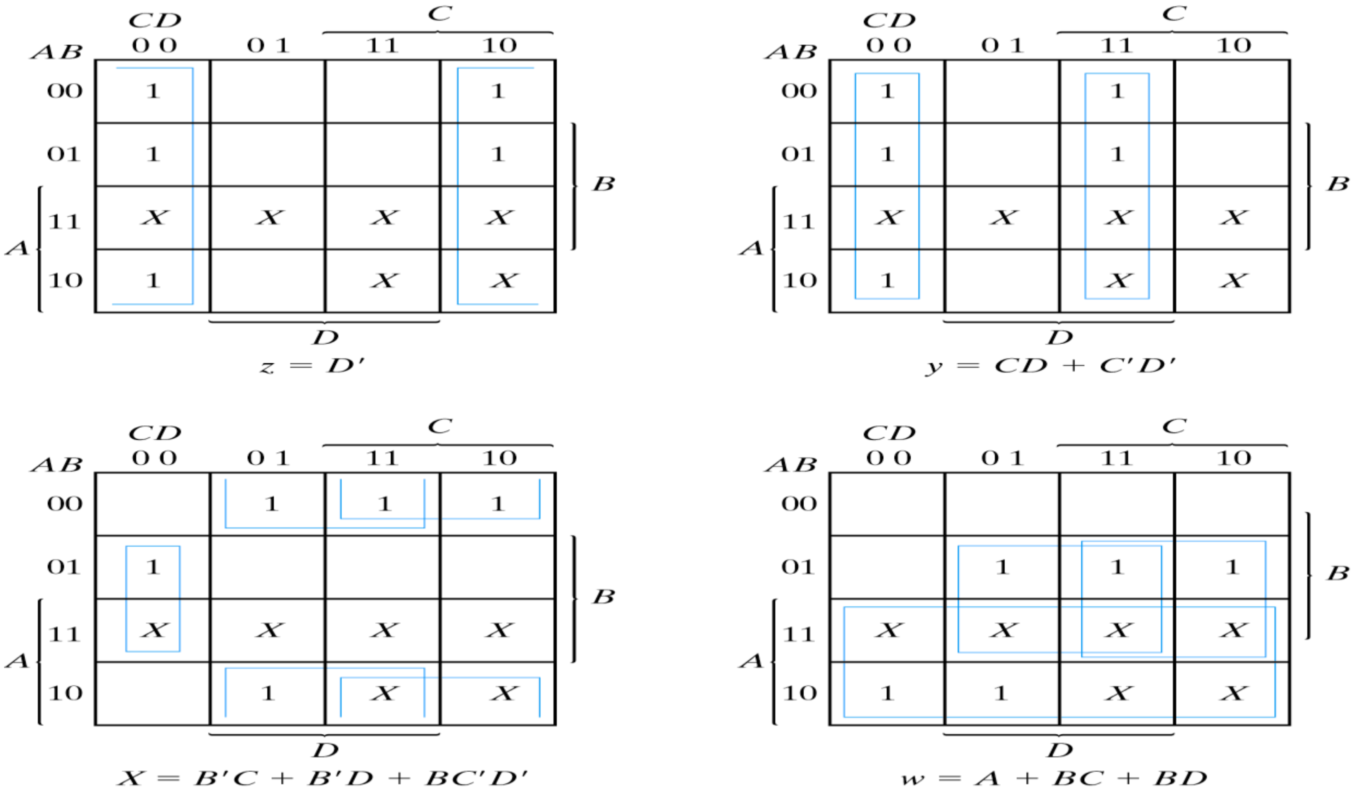


Table 4-2
Truth Table for Code-Conversion Example

Input BCD				Output Excess-3 Code			
A	*B*	*C*	*D*	*w*	*x*	*y*	*z*
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

Karnaugh map

2. For each symbol of the Excess-3 code, we use 1's to draw the map for simplifying Boolean function.



AB \ CD	00	01	11	10
00	1			1
01	1			1
11	X	X	X	X
10	1		X	X

$z = D'$

AB \ CD	00	01	11	10
00	1		1	
01	1		1	
11	X	X	X	X
10	1		X	X

$y = CD + C'D'$

AB \ CD	00	01	11	10
00		1	1	1
01	1			
11	X	X	X	X
10		1	X	X

$X = B'C + B'D + BC'D'$

AB \ CD	00	01	11	10
00				
01		1	1	1
11	X	X	X	X
10	1	1	X	X

$w = A + BC + BD$

Fig. 4-3 Maps for BCD to Excess-3 Code Converter

Circuit implementation

z = D';   y = CD + C'D' = CD + (C + D)'

x = B'C + B'D + BC'D' = B'(C + D) + B(C + D)'

w = A + BC + BD = A + B(C + D)

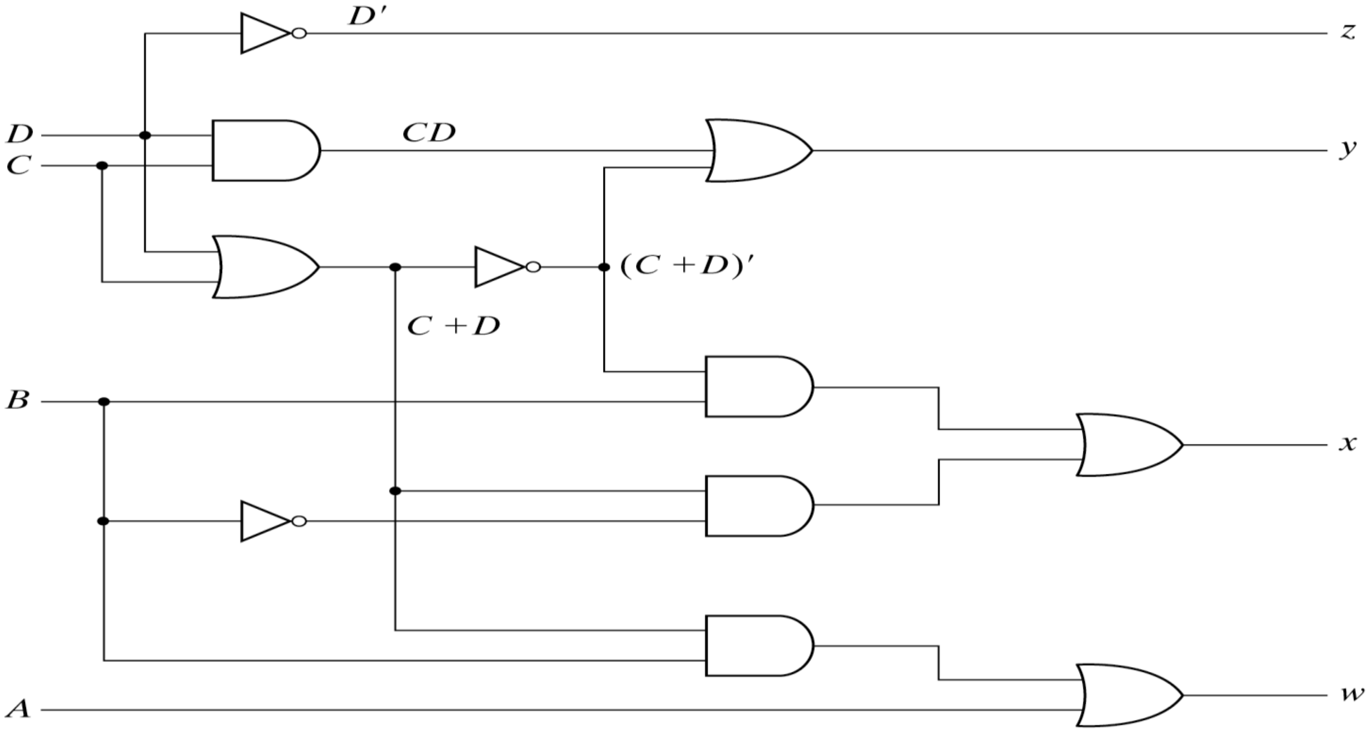


Fig. 4-4 Logic Diagram for BCD to Excess-3 Code Converter

4-4. Binary Adder-Subtractor

- A combinational circuit that performs the addition of two bits is called a half adder.
- The truth table for the half adder is listed below:

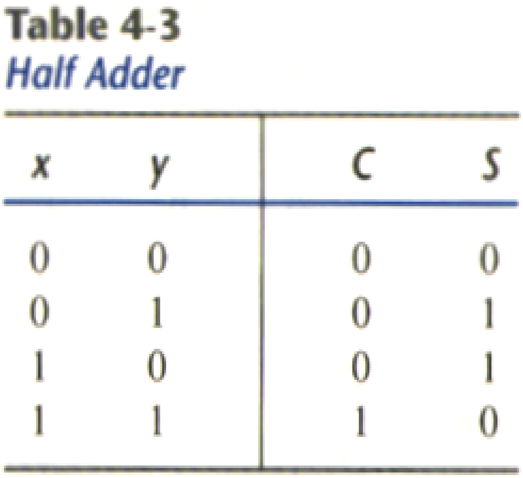


Table 4-3
Half Adder

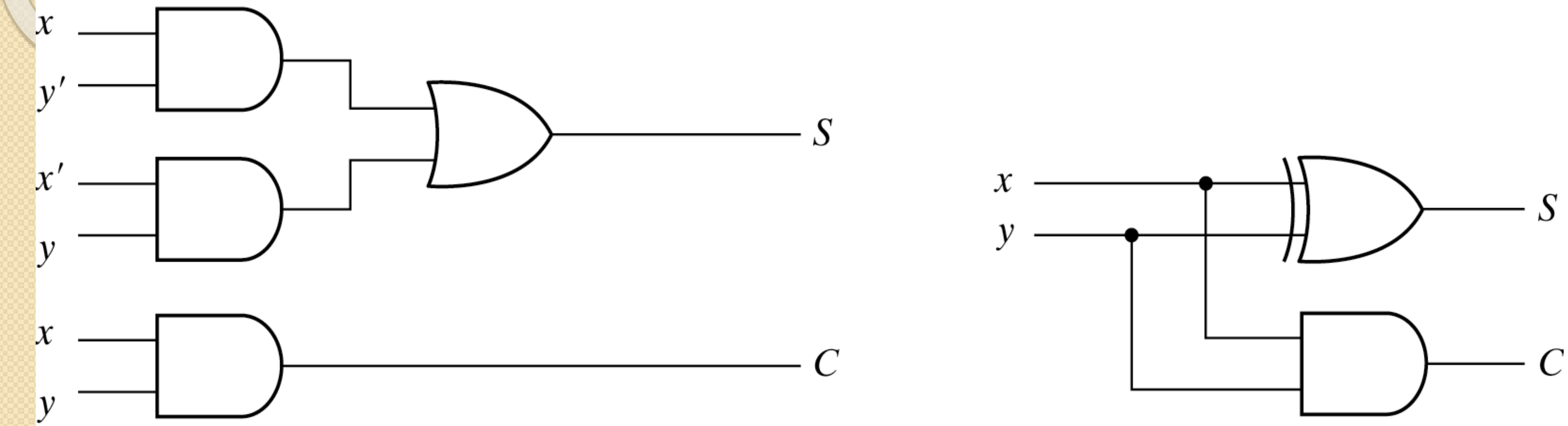
x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

S: Sum
C: Carry

$S = x'y + xy'$

$C = xy$

Implementation of Half-Adder



(a) $S = xy' + x'y$
$C = xy$

(b) $S = x \oplus y$
$C = xy$

Fig. 4-5 Implementation of Half-Adder

Full-Adder

- One that performs the addition of three bits(two significant bits and a previous carry) is a full adder.

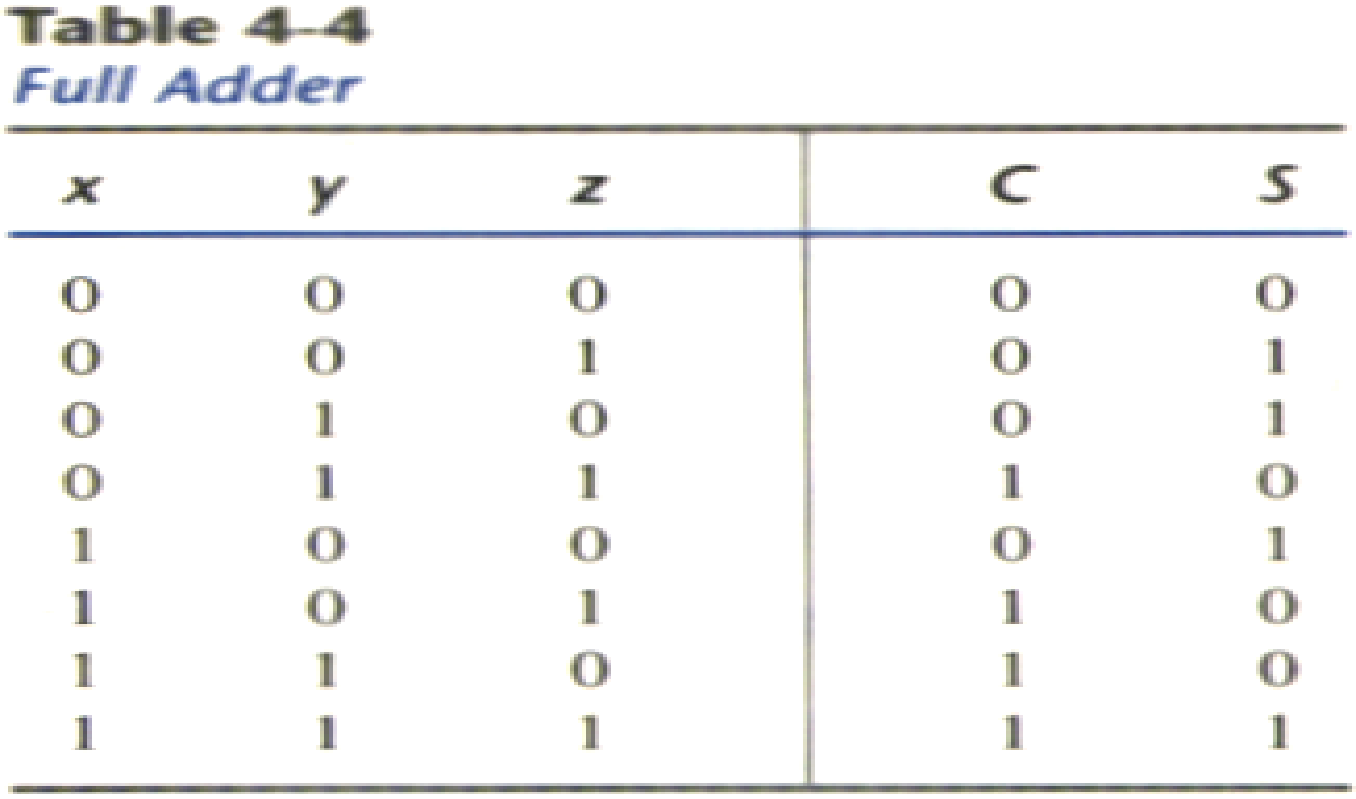
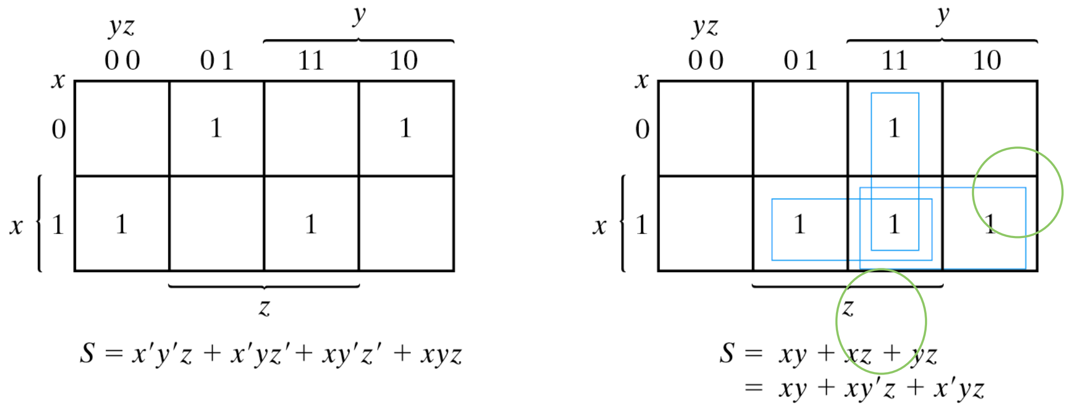


Table 4-4
Full Adder

x	*y*	*z*	*C*	*S*
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Simplified Expressions



$S = x'y'z + x'yz' + xy'z' + xyz$

$S = xy + xz + yz$
$= xy + xy'z + x'yz$

Fig. 4-6 Maps for Full Adder

C

S = x’y’z + x’yz’ + xy’z’ + xyz

C = xy + xz + yz

Full adder implemented in SOP

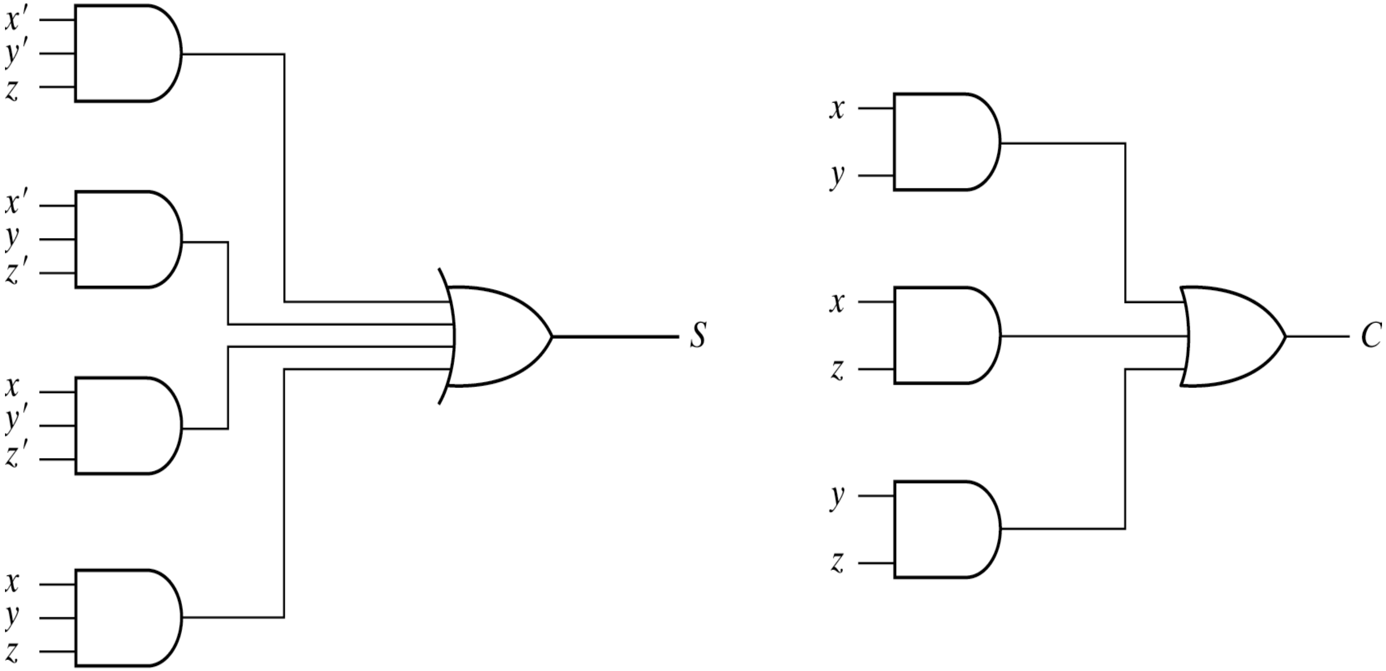


Fig. 4-7 Implementation of Full Adder in Sum of Products

Another implementation

- Full-adder can also implemented with two half adders and one OR gate (Carry Look-Ahead adder).

$S = z \oplus (x \oplus y)$

$\quad = z'(xy' + x'y) + z(xy' + x'y)'$

$\quad = xy'z' + x'yz' + xyz + x'y'z$

$C = z(xy' + x'y) + xy = xy'z + x'yz + xy$

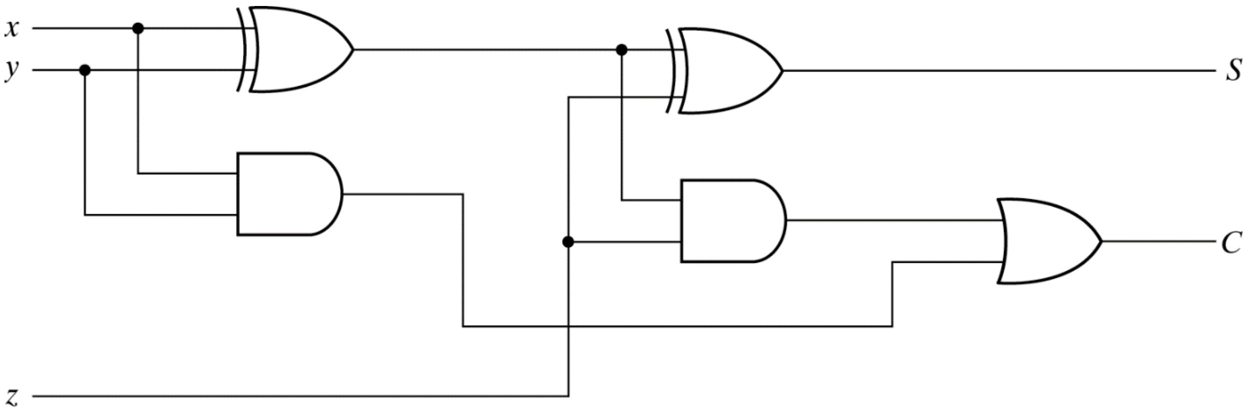
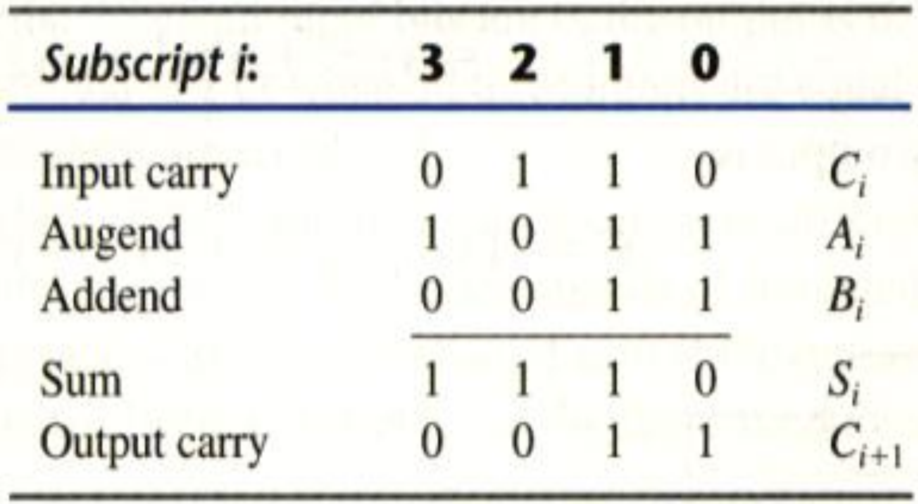


Fig. 4-8 Implementation of Full Adder with Two Half Adders and an OR Gate

Binary adder

- This is also called Ripple Carry Adder ,because of the construction with full adders are connected in cascade.



Subscript i:	**3**	**2**	**1**	**0**	
Input carry	0	1	1	0	C_i
Augend	1	0	1	1	A_i
Addend	0	0	1	1	B_i
Sum	1	1	1	0	S_i
Output carry	0	0	1	1	C_{i+1}

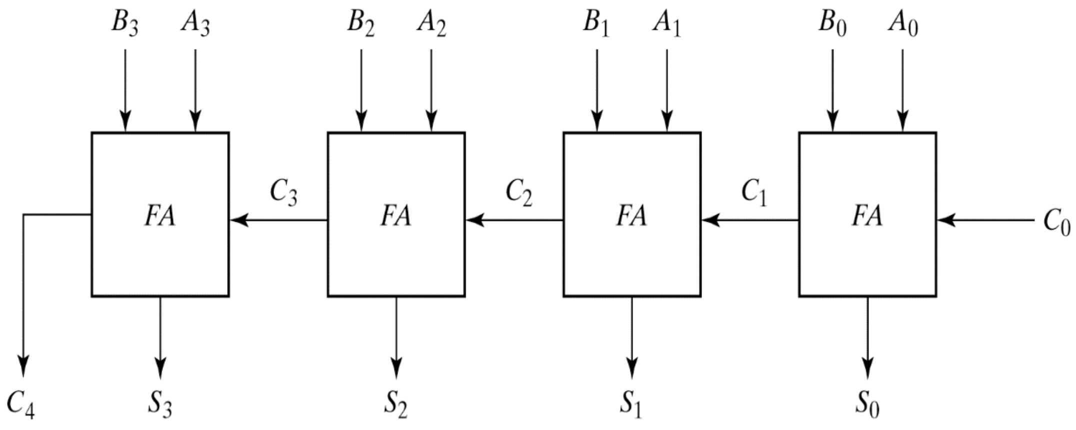


Fig. 4-9 4-Bit Adder

Carry Propagation

- Fig.4-9 causes a unstable factor on carry bit, and produces a longest propagation delay.
- The signal from C_i to the output carry C_{i+1}, propagates through an AND and OR gates, so, for an n-bit RCA, there are 2n gate levels for the carry to propagate from input to output.

Carry Propagation

- Because the propagation delay will affect the output signals on different time, so the signals are given enough time to get the precise and stable outputs.
- The most widely used technique employs the principle of carry look-ahead to improve the speed of the algorithm.

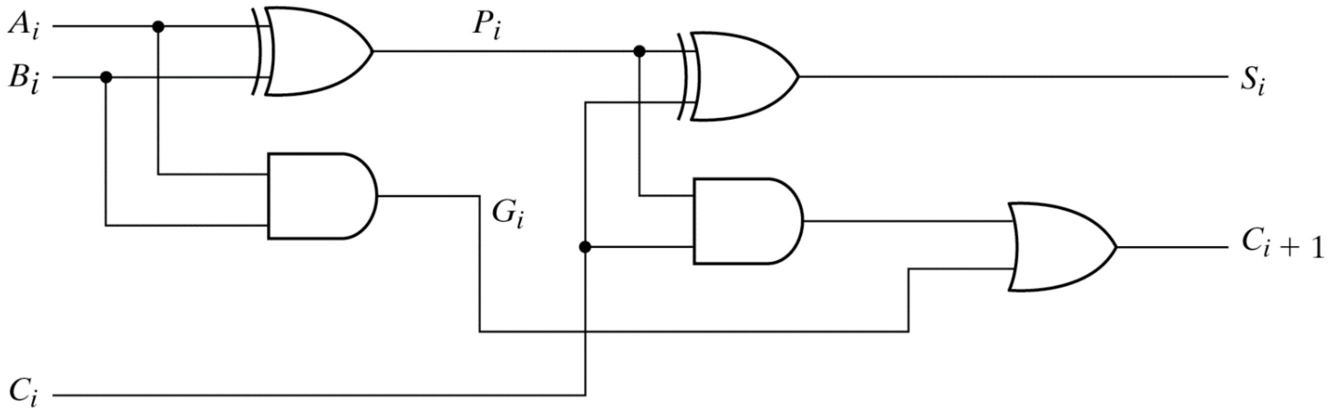


Fig. 4-10 Full Adder with P and G Shown

Boolean functions

$P_i = A_i \oplus B_i$ steady state value

$G_i = A_iB_i$ steady state value

Output sum and carry

$S_i = P_i \oplus C_i$

$C_{i+1} = G_i + P_iC_i$

G_i : carry generate P_i : carry propagate

C_0 = input carry

$C_1 = G_0 + P_0C_0$

$C_2 = G_1 + P_1C_1 = G_1 + P_1G_0 + P_1P_0C_0$

$C_3 = G_2 + P_2C_2 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0$

- C_3 does not have to wait for C_2 and C_1 to propagate.

Logic diagram of carry look-ahead generator

- C_3 is propagated at the same time as C_2 and C_1.

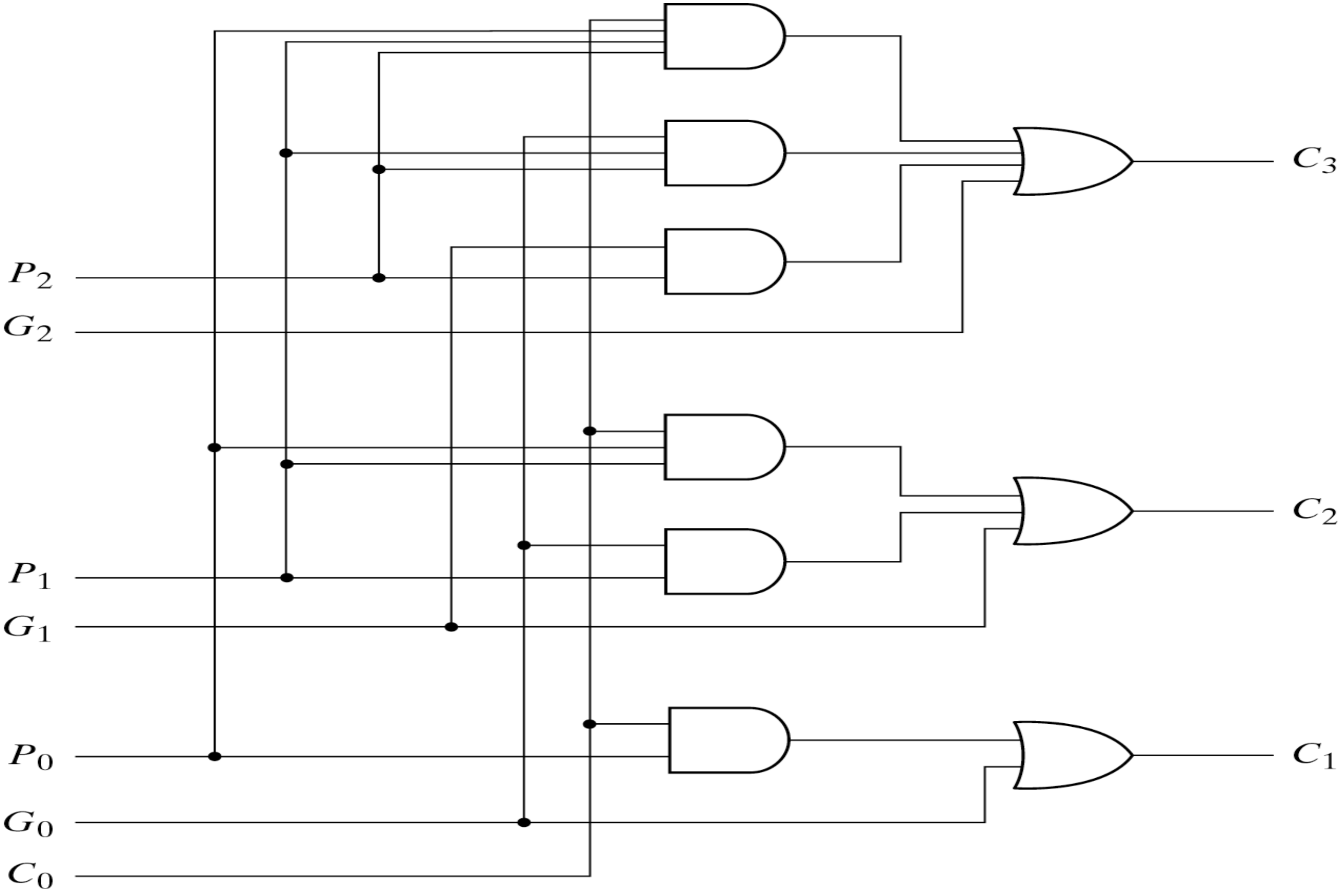


Fig. 4-11 Logic Diagram of Carry Lookahead Generator

4-bit adder with carry lookahead

- Delay time of n-bit CLAA = XOR + (AND + OR) + XOR

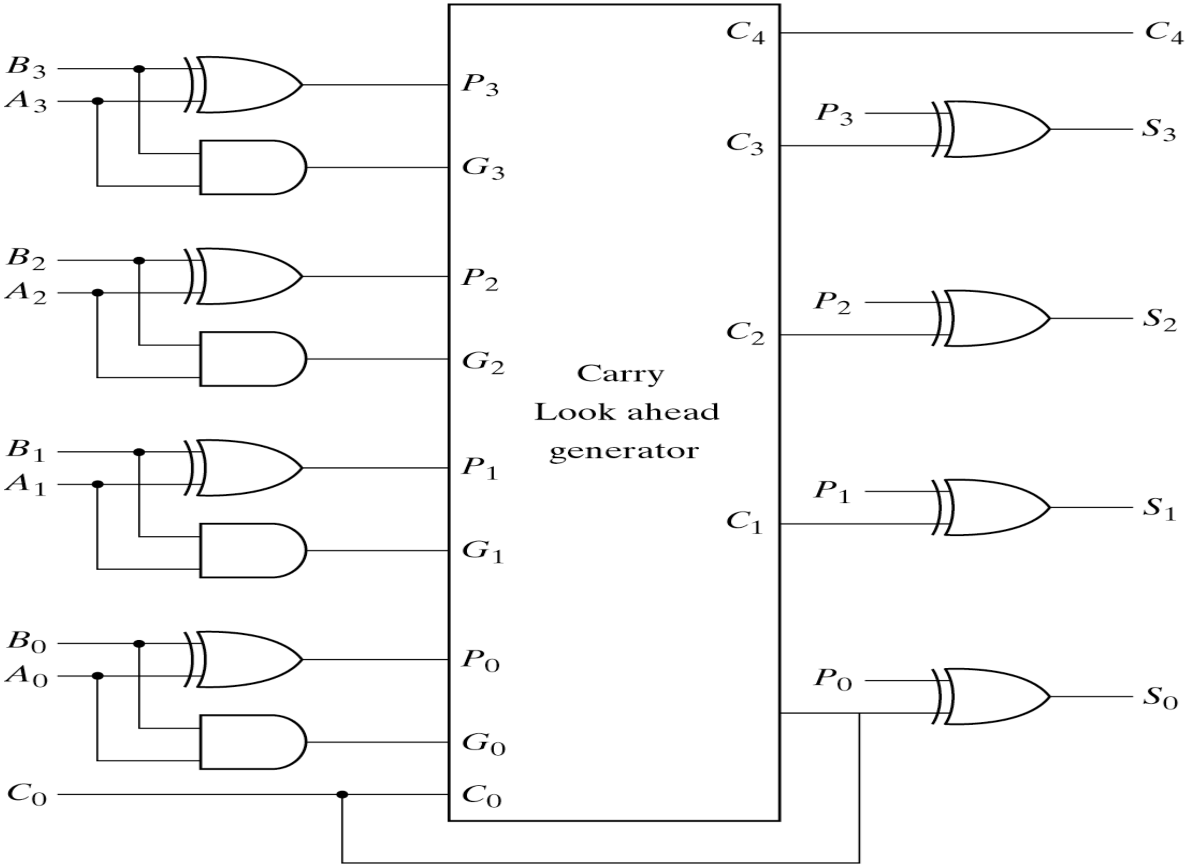


Fig. 4-12 4-Bit Adder with Carry Lookahead

Binary subtractor

M = 1→subtractor ; M = 0→adder

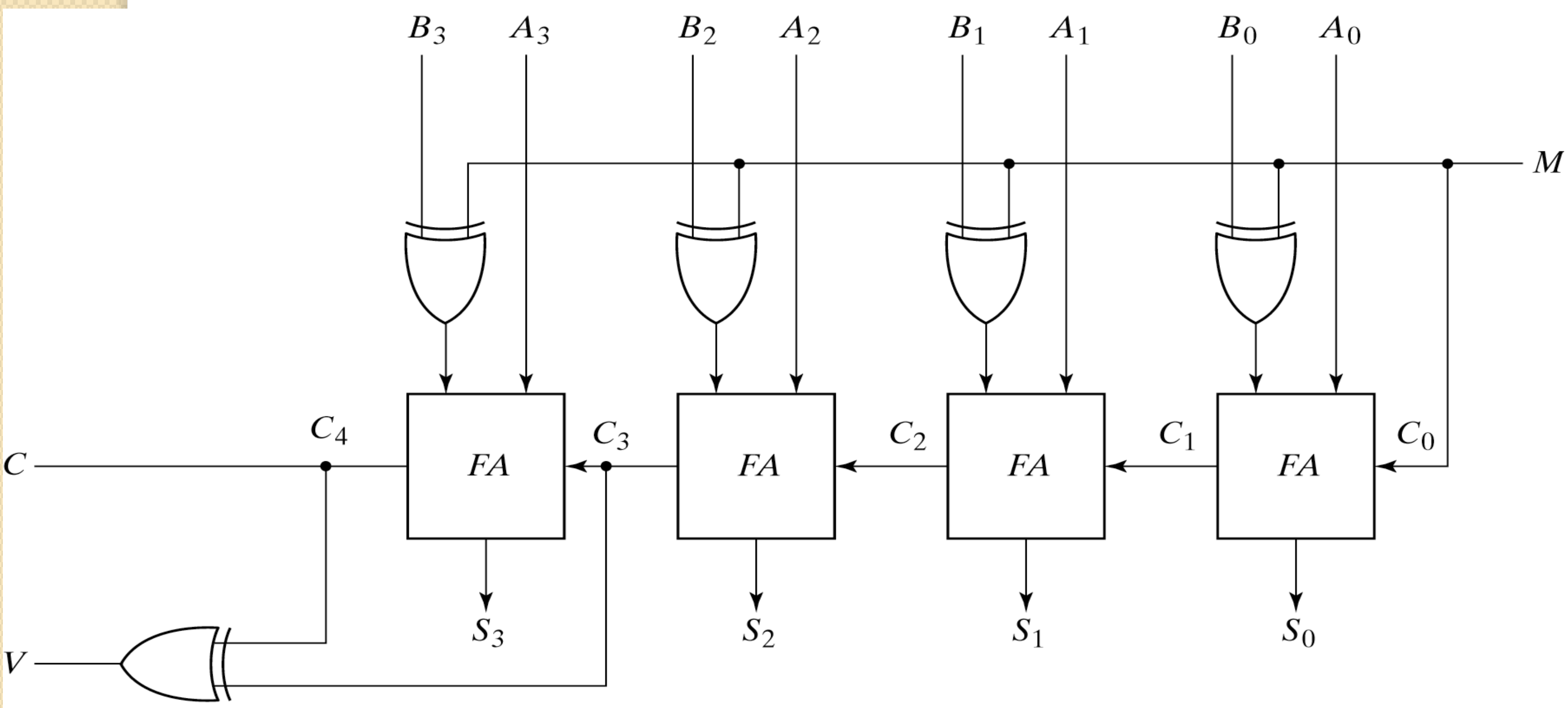


Fig. 4-13 4-Bit Adder Subtractor

Overflow

- It is worth noting Fig.4-13 that binary numbers in the signed-complement system are added and subtracted by the same basic addition and subtraction rules as unsigned numbers.
- Overflow is a problem in digital computers because the number of bits that hold the number is finite and a result that contains n+1 bits cannot be accommodated.

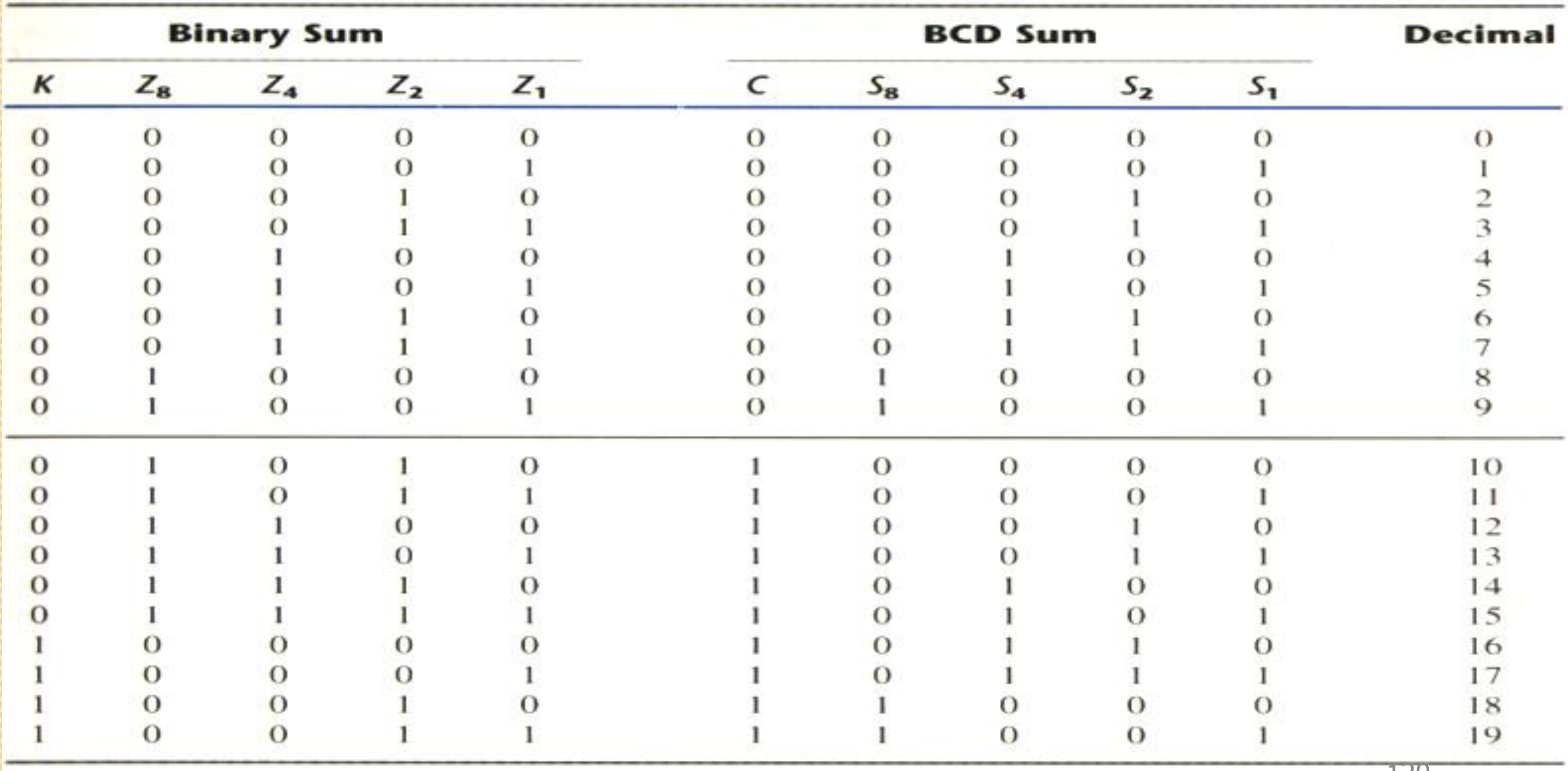
Overflow on signed and unsigned

- When two unsigned numbers are added, an overflow is detected from the end carry out of the MSB position.
- When two signed numbers are added, the sign bit is treated as part of the number and the end carry does not indicate an overflow.
- An overflow cann't occur after an addition if one number is positive and the other is negative.
- An overflow may occur if the two numbers added are both positive or both negative.

4-5 Decimal adder

BCD adder can’t exceed 9 on each input digit. K is the carry.

Table 4-5
Derivation of BCD Adder



Binary Sum					BCD Sum					Decimal
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

Rules of BCD adder

- When the binary sum is greater than 1001, we obtain a non-valid BCD representation.
- The addition of binary 6(0110) to the binary sum converts it to the correct BCD representation and also produces an output carry as required.
- To distinguish them from binary 1000 and 1001, which also have a 1 in position Z_8, we specify further that either Z_4 or Z_2 must have a 1.

$$C = K + Z_8Z_4 + Z_8Z_2$$

Implementation of BCD adder

- A decimal parallel adder that adds n decimal digits needs n BCD adder stages.
- The output carry from one stage must be connected to the input carry of the next higher-order stage.

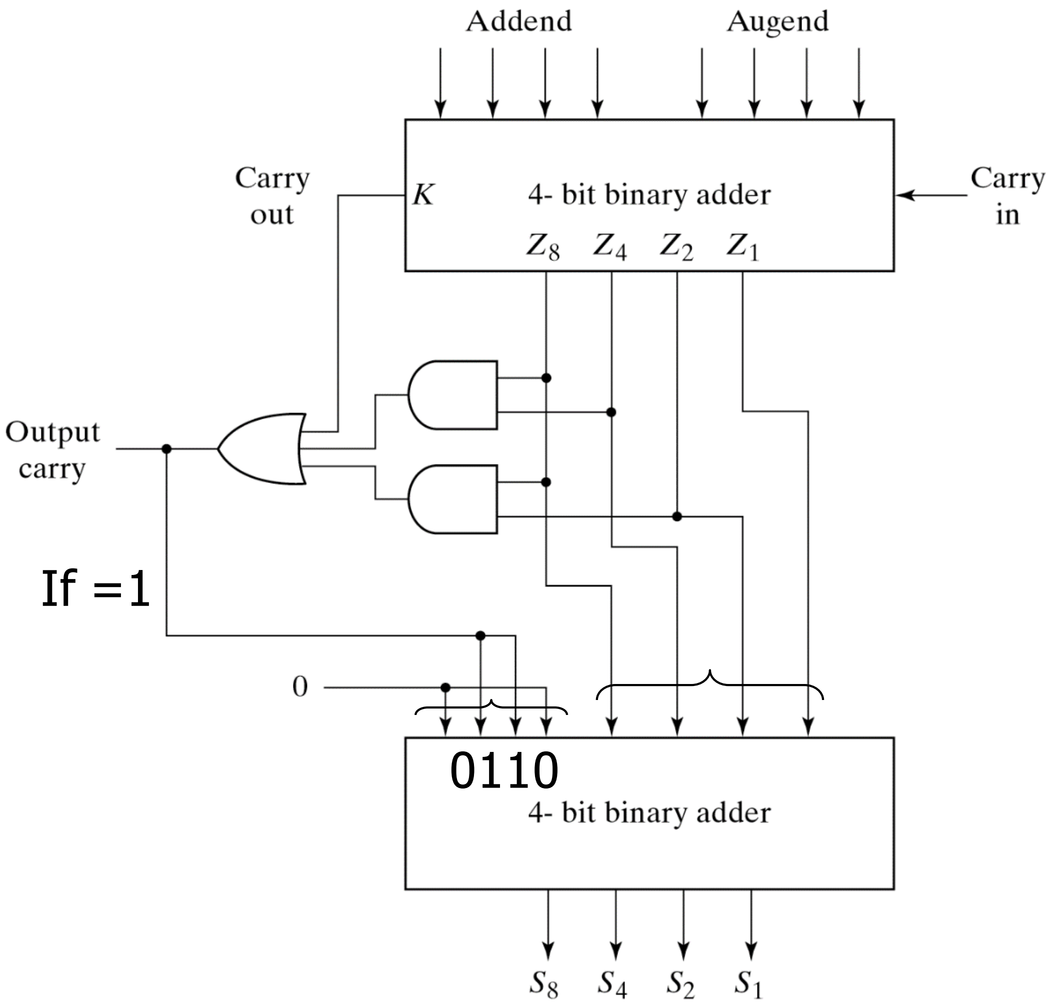


Fig. 4-14 Block Diagram of a BCD Adder

4-6. Binary multiplier

- Usually there are more bits in the partial products and it is necessary to use full adders to produce the sum of the partial products.

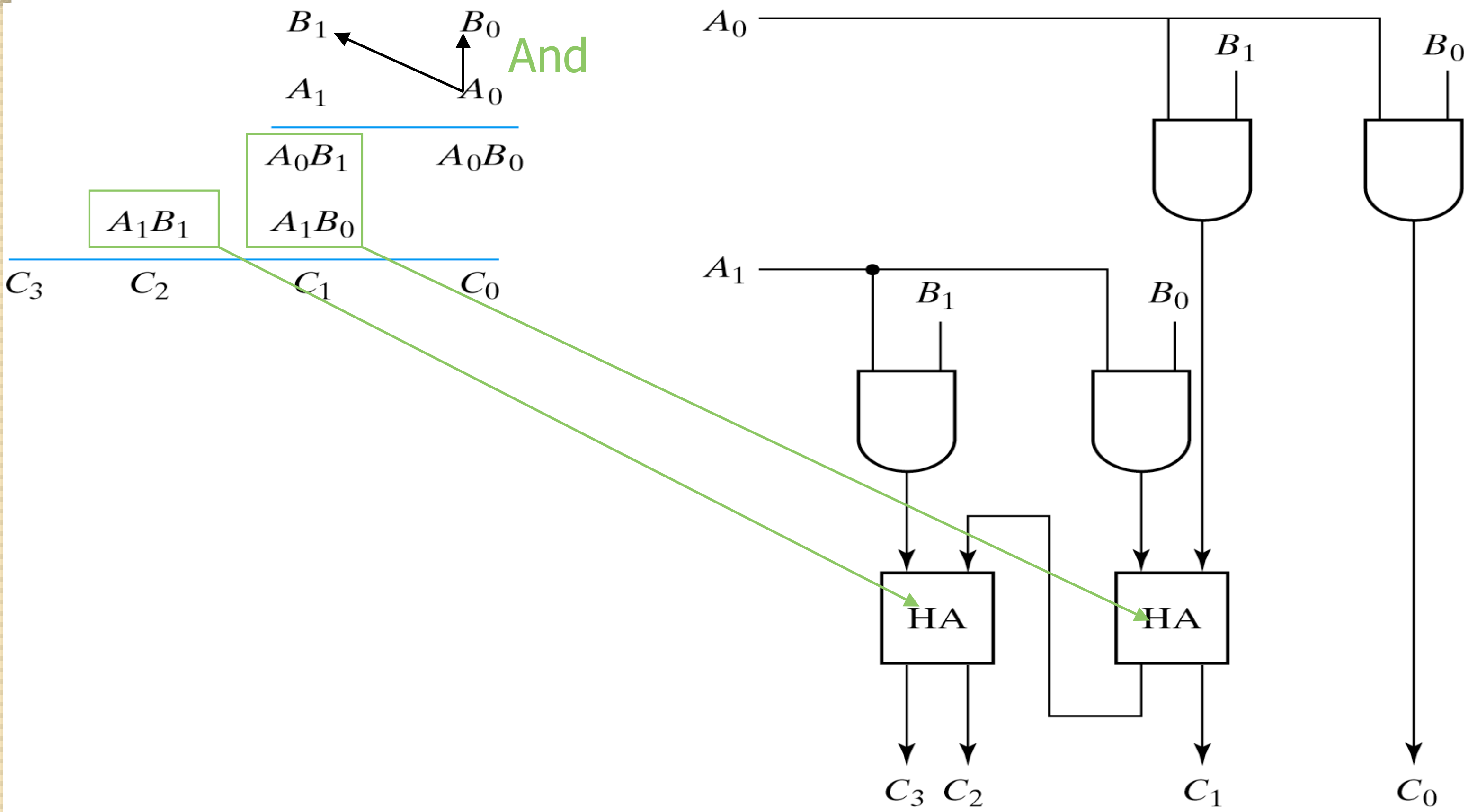


Fig. 4-15 2-Bit by 2-Bit Binary Multiplier

4-bit by 3-bit binary multiplier

- For J multiplier bits and K multiplicand bits we need (J X K) AND gates and (J − 1) K-bit adders to produce a product of J+K bits.
- K=4 and J=3, we need 12 AND gates and two 4-bit adders.

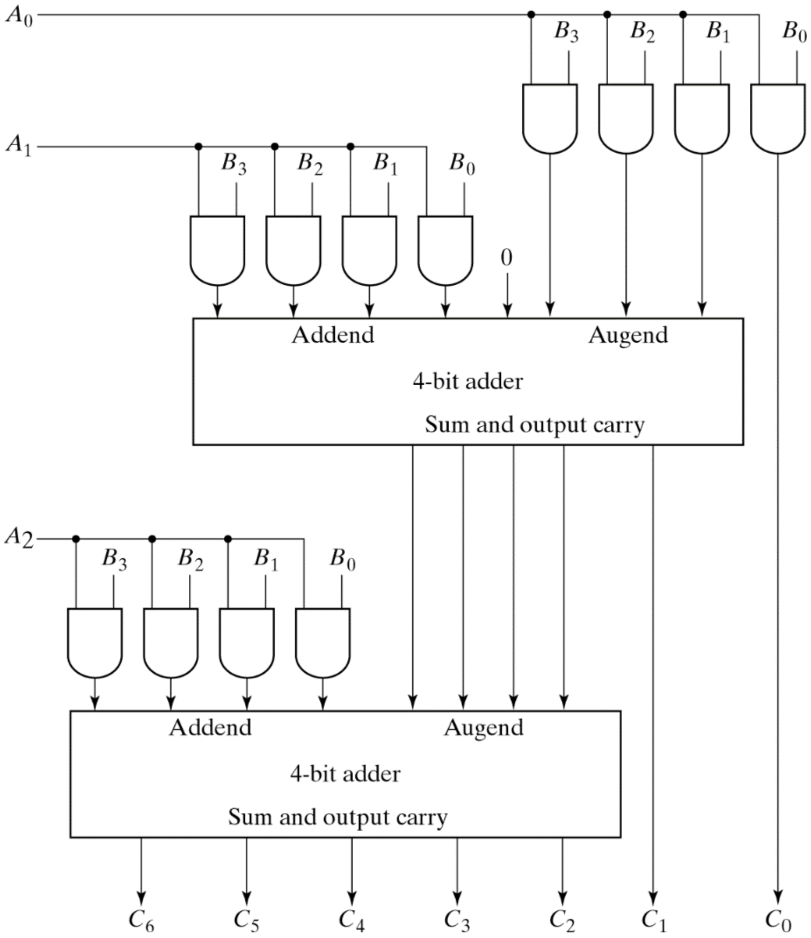


Fig. 4-16 4-Bit by 3-Bit Binary Multiplier

4-7. Magnitude comparator

- The equality relation of each pair of bits can be expressed logically with an exclusive-NOR function as:

$A = A_3A_2A_1A_0$; $B = B_3B_2B_1B_0$

$x_i = A_iB_i + A_i'B_i'$ for $i = 0, 1, 2, 3$

$(A = B) = x_3x_2x_1x_0$

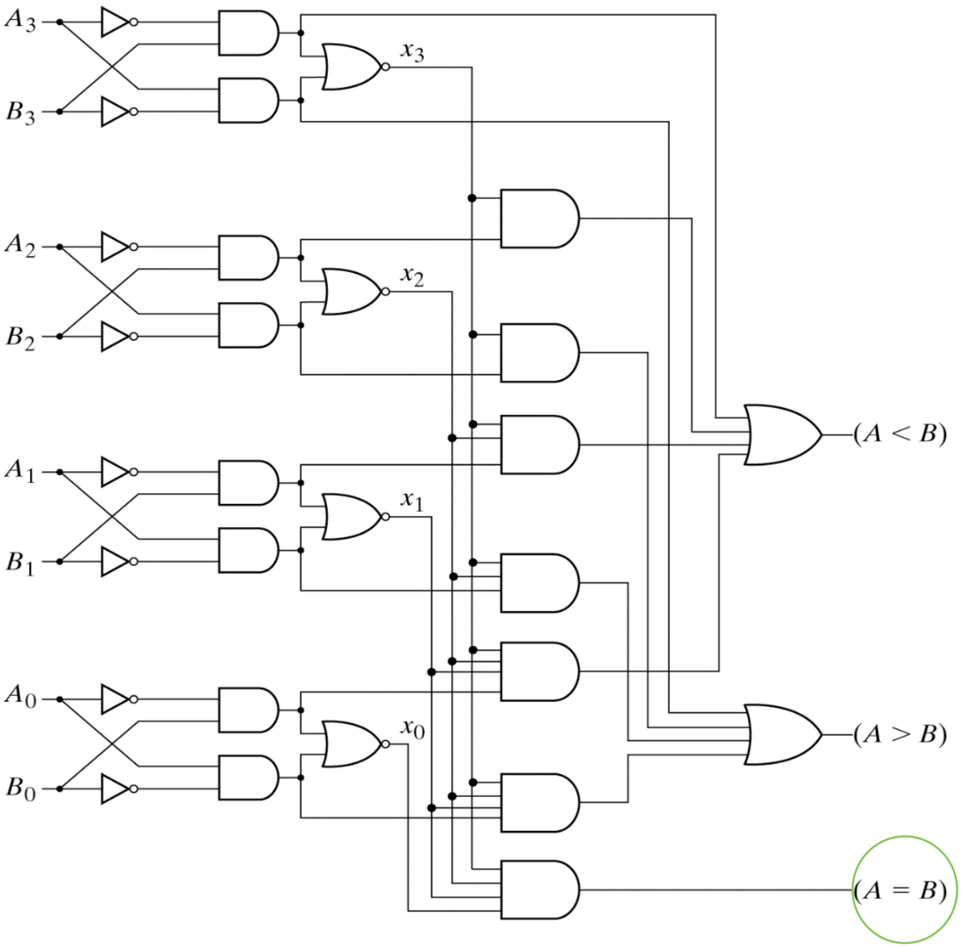


Fig. 4-17 4-Bit Magnitude Comparator

Magnitude comparator

- We inspect the relative magnitudes of pairs of MSB. If equal, we compare the next lower significant pair of digits until a pair of unequal digits is reached.
- If the corresponding digit of A is 1 and that of B is 0, we conclude that A>B.

$(A>B)=$
$A_3B'_3+x_3A_2B'_2+x_3x_2A_1B'_1+x_3x_2x_1A_0B'_0$
$(A<B)=$
$A'_3B_3+x_3A'_2B_2+x_3x_2A'_1B_1+x_3x_2x_1A'_0B_0$

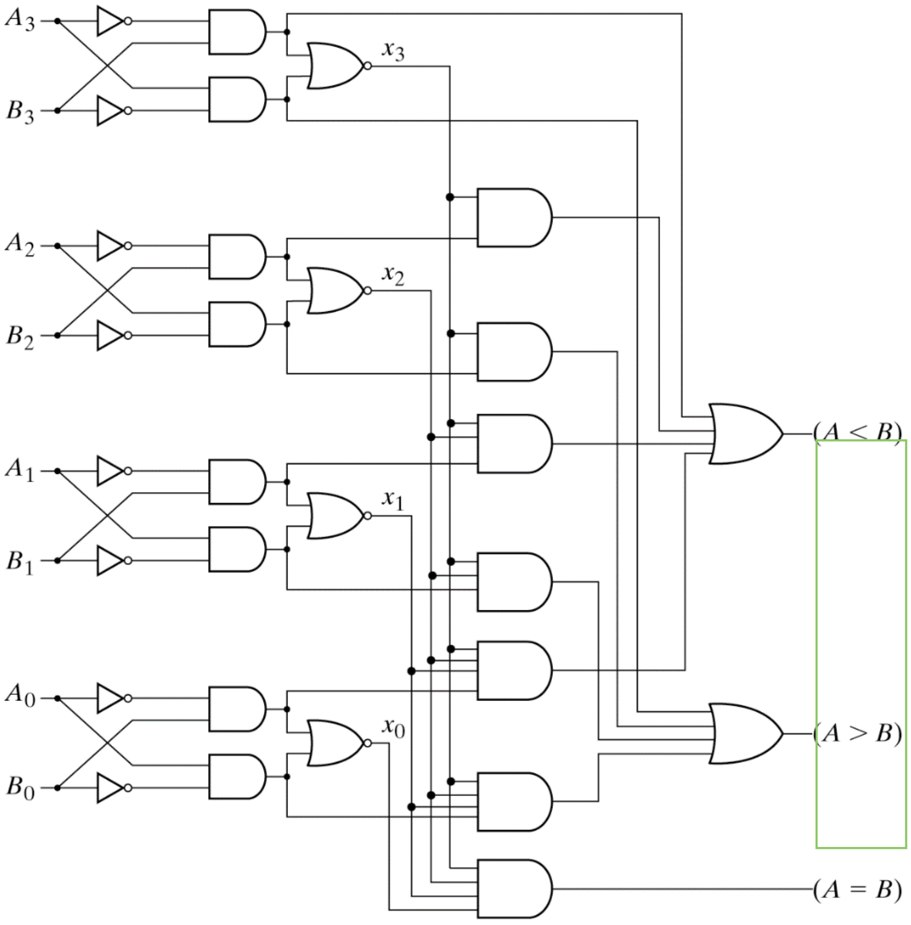


Fig. 4-17 4-Bit Magnitude Comparator

4-8. Decoders

- The decoder is called n-to-m-line decoder, where $m \leq 2^n$.
- the decoder is also used in conjunction with other code converters such as a BCD-to-seven_segment decoder.
- 3-to-8 line decoder: For each possible input combination, there are seven outputs that are equal to 0 and only one that is equal to 1.

Implementation and truth table

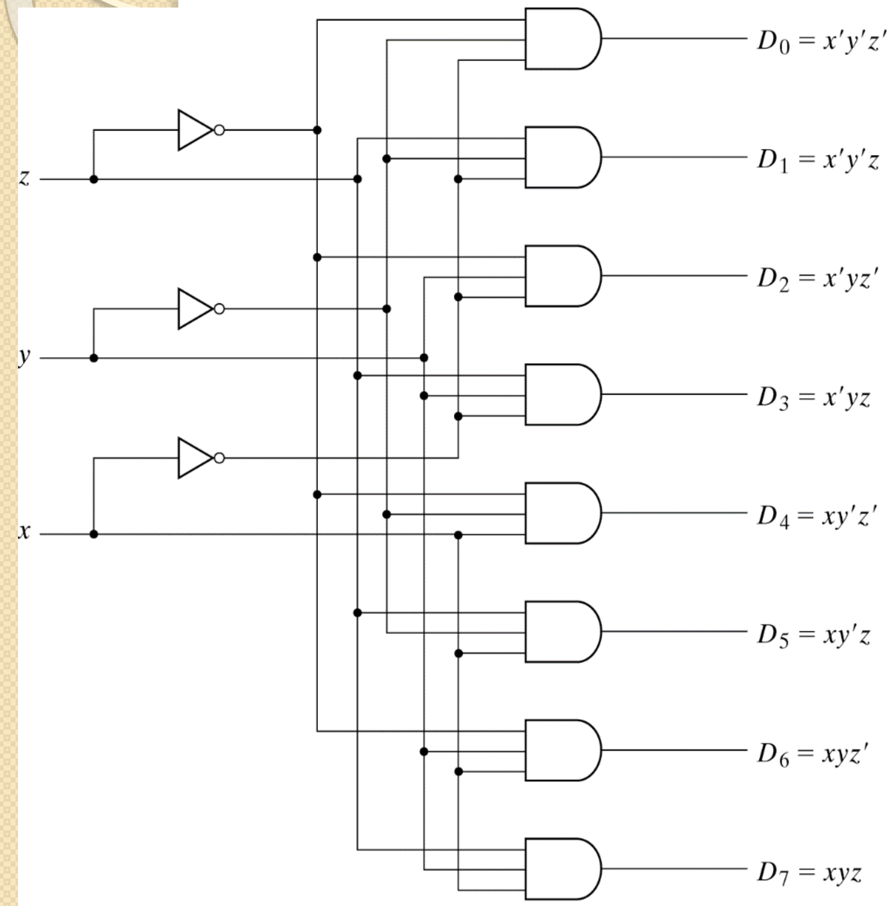


Fig. 4-18 3-to-8-Line Decoder

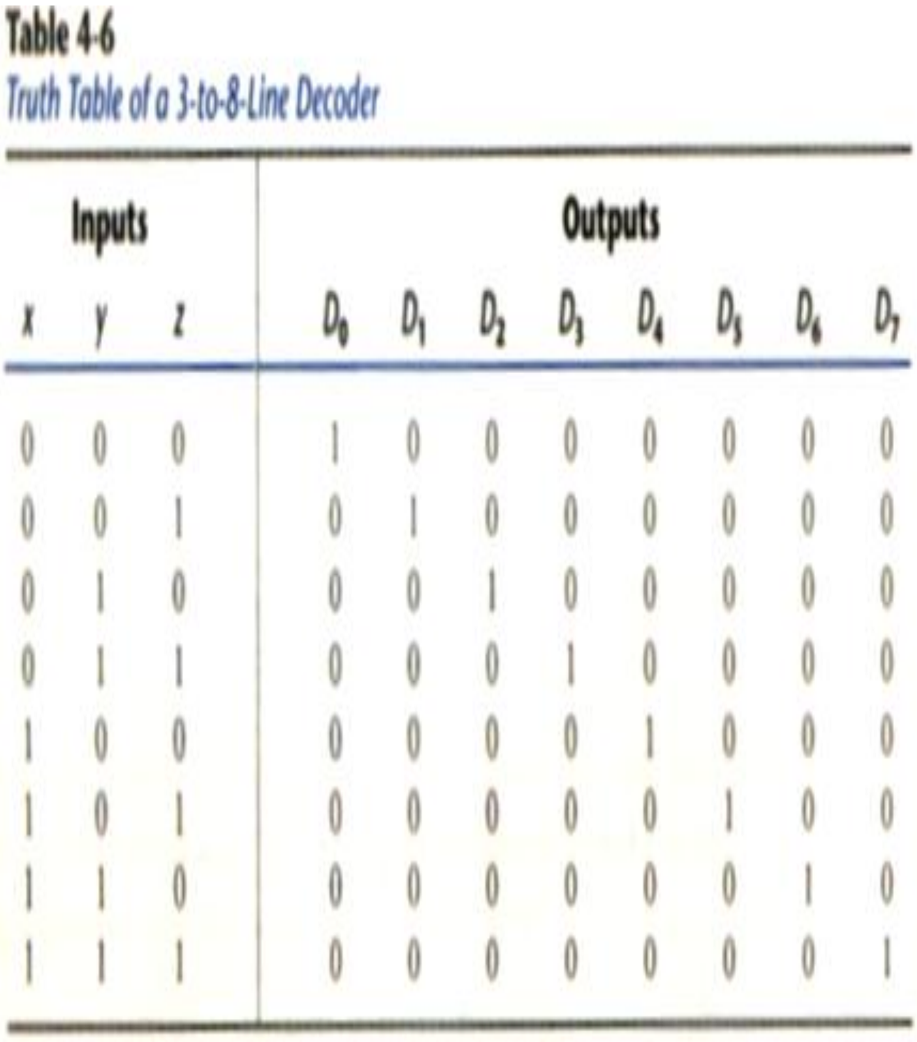


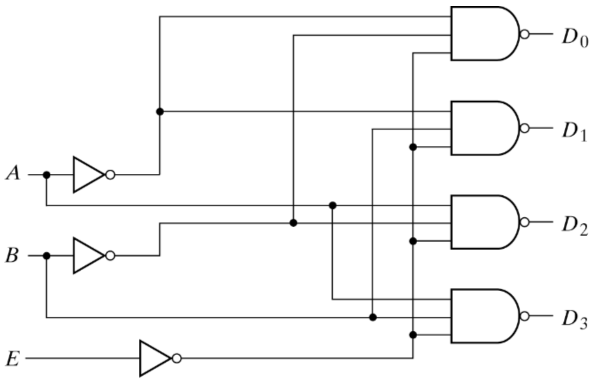
Table 4-6

Truth Table of a 3-to-8-Line Decoder

Inputs			Outputs							
x	y	z	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Decoder with enable input

- Some decoders are constructed with NAND gates, it becomes more economical to generate the decoder minterms in their complemented form.
- As indicated by the truth table , only one output can be equal to 0 at any given time, all other outputs are equal to 1 .



(a) Logic diagram

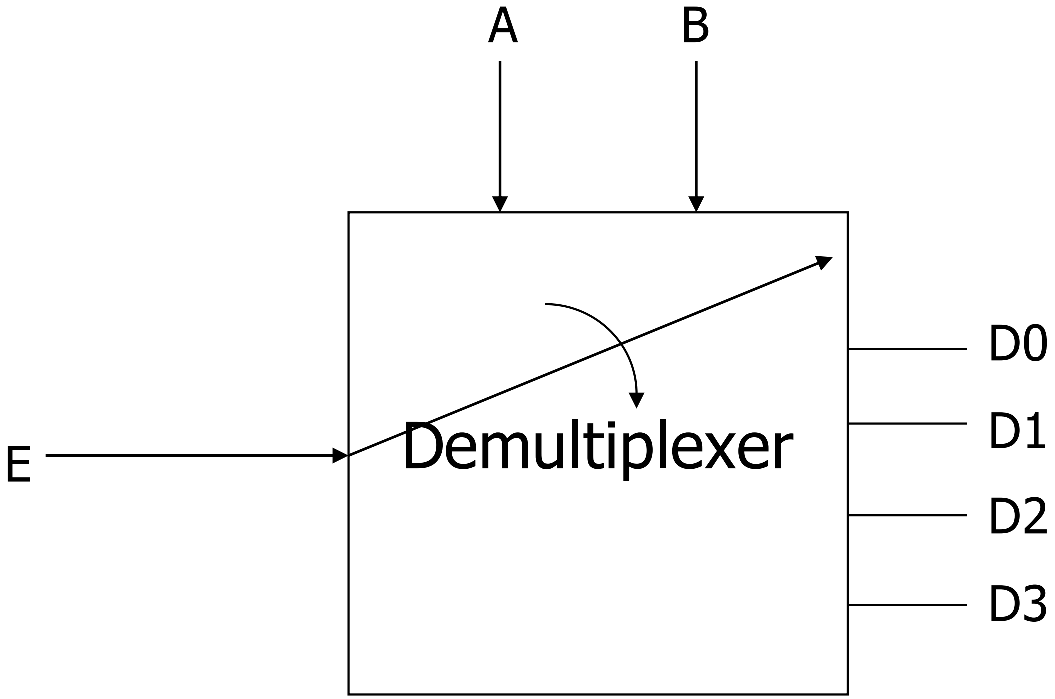
E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

(b) Truth table

Fig. 4-19 2-to-4-Line Decoder with Enable Input

Demultiplexer

- A decoder with an enable input is referred to as a decoder/demultiplexer.
- The truth table of demultiplexer is the same with decoder.



3-to-8 decoder with enable implement the 4-to-16 decoder

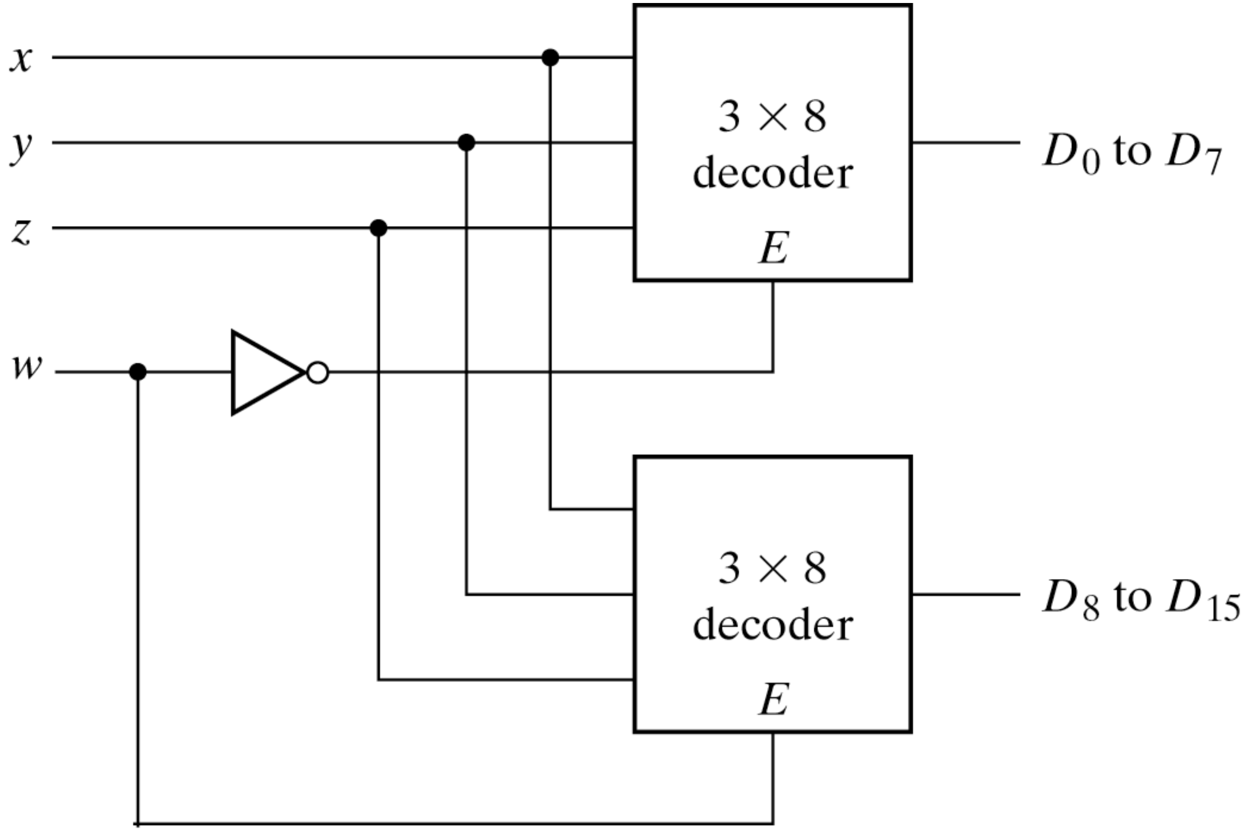


Fig. 4-20 4 × 16 Decoder Constructed with Two 3 × 8 Decoders

Implementation of a Full Adder with a Decoder

- From table 4-4, we obtain the functions for the combinational circuit in sum of minterms:

$$S(x, y, z) = \sum(1, 2, 4, 7)$$

$$C(x, y, z) = \sum(3, 5, 6, 7)$$

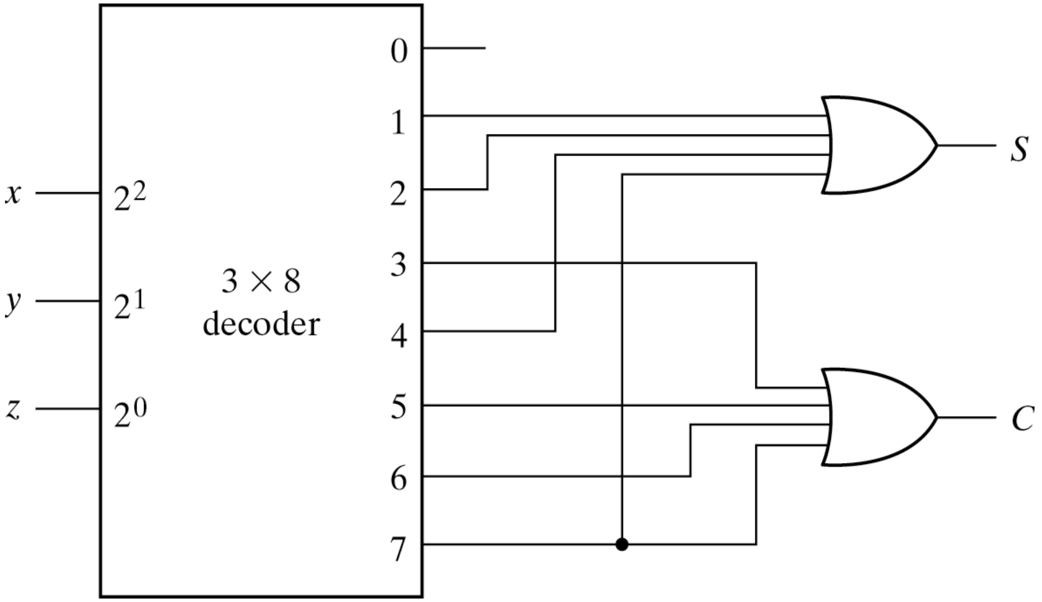


Fig. 4-21 Implementation of a Full Adder with a Decoder

4-9. Encoders

- An encoder is the inverse operation of a decoder.
- We can derive the Boolean functions by table 4-7

$$z = D_1 + D_3 + D_5 + D_7$$

$$y = D_2 + D_3 + D_6 + D_7$$

$$x = D_4 + D_5 + D_6 + D_7$$

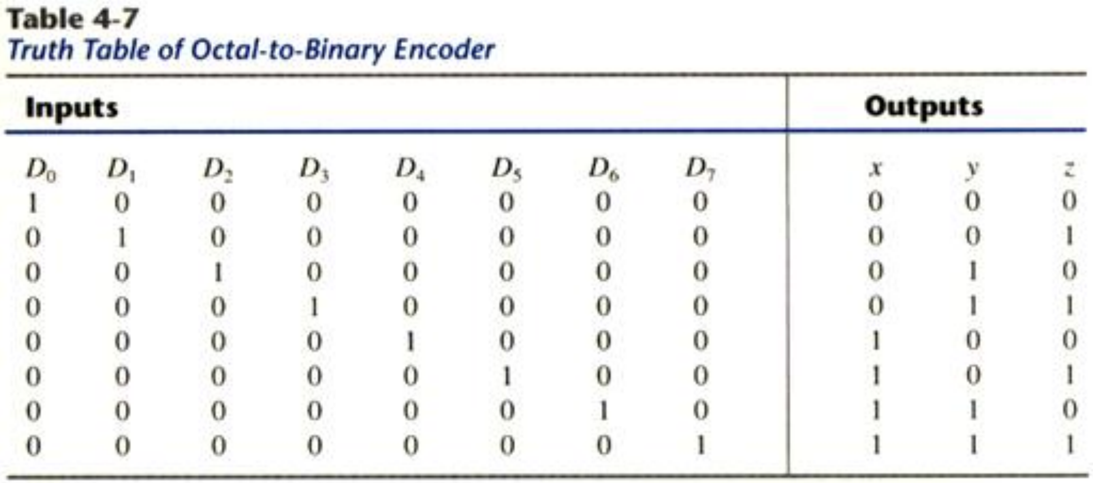


Table 4-7
Truth Table of Octal-to-Binary Encoder

Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

Priority encoder

- If two inputs are active simultaneously, the output produces an undefined combination. We can establish an input priority to ensure that only one input is encoded.
- Another ambiguity in the octal-to-binary encoder is that an output with all 0's is generated when all the inputs are 0; the output is the same as when D_0 is equal to 1.
- The discrepancy tables on Table 4-7 and Table 4-8 can resolve aforesaid condition by providing one more output to indicate that at least one input is equal to 1.

Priority encoder

V=0→no valid inputs

V=1→valid inputs

X's in output columns represent don't-care conditions

X's in the input columns are useful for representing a truth table in condensed form.

Instead of listing all 16 minterms of four variables.

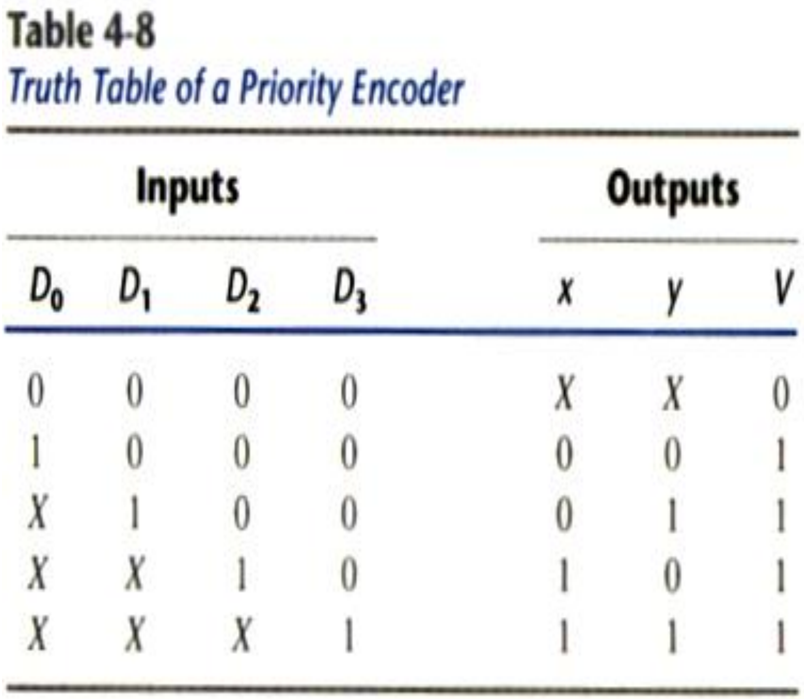


Table 4-8

Truth Table of a Priority Encoder

Inputs				Outputs		
D_0	D_1	D_2	D_3	x	y	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

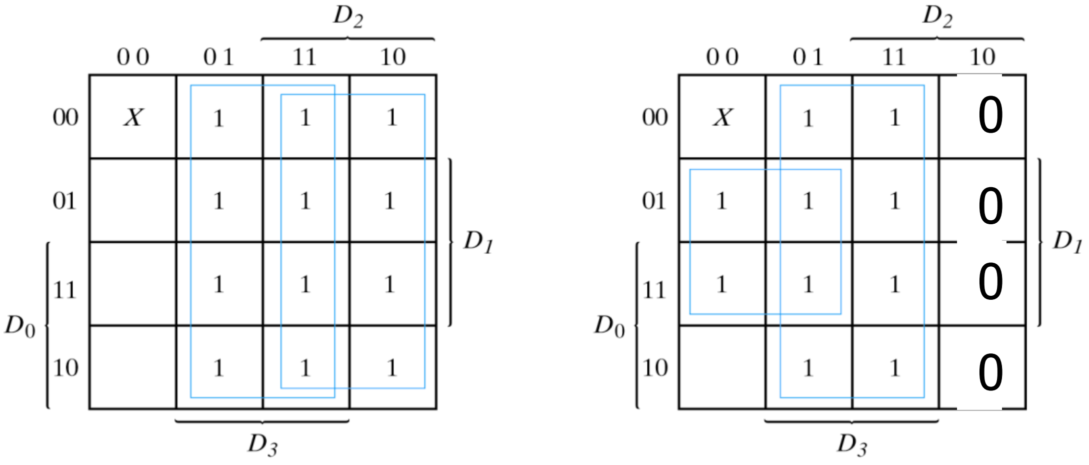
4-input priority encoder

- Implementation of table 4-8

$x = D_2 + D_3$

$y = D_3 + D_1D'_2$

$V = D_0 + D_1 + D_2 + D_3$



	00	01	11	10
00	X	1	1	1
01		1	1	1
11		1	1	1
10		1	1	1

$x = D_2 + D_3$

	00	01	11	10
00	X	1	1	0
01	1	1	1	0
11	1	1	1	0
10		1	1	0

$y = D_3 + D_1D'_2$

Fig. 4-22 Maps for a Priority Encoder

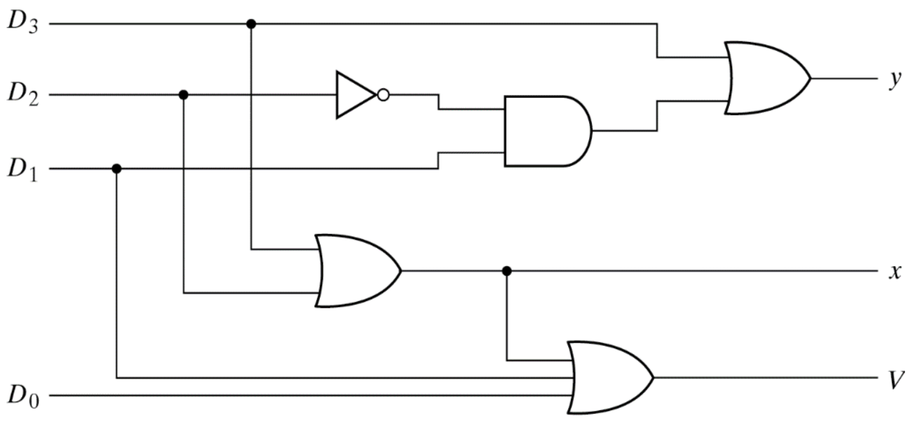


Fig. 4-23 4-Input Priority Encoder

4-10. Multiplexers

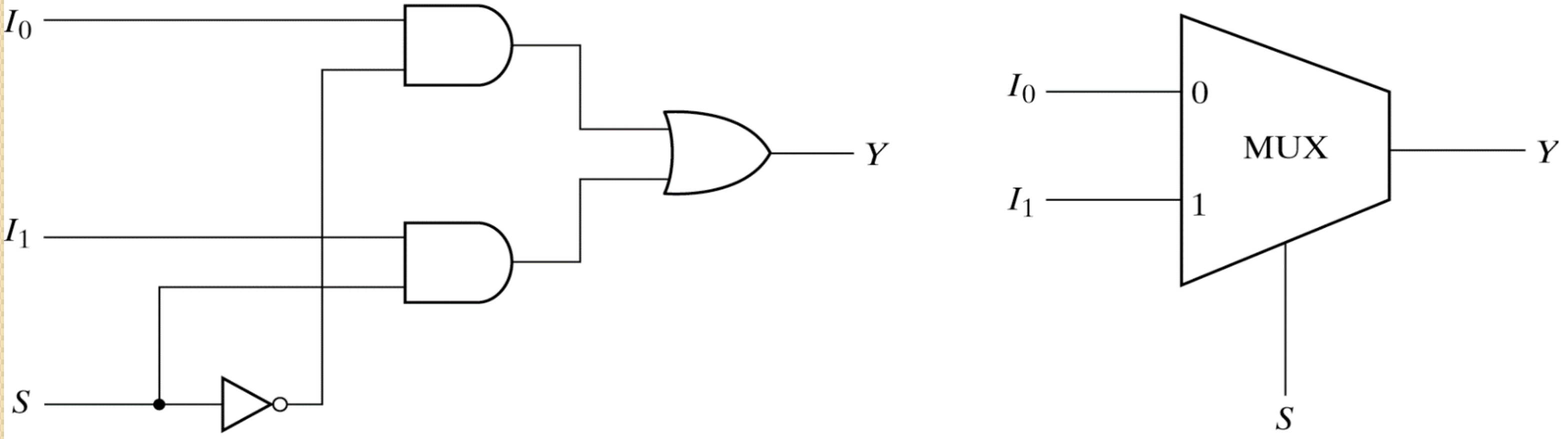
$S = 0, Y = I_0$

$S = 1, Y = I_1$

Truth Table→

S	Y
0	I_0
1	I_1

$Y = S'I_0 + SI_1$

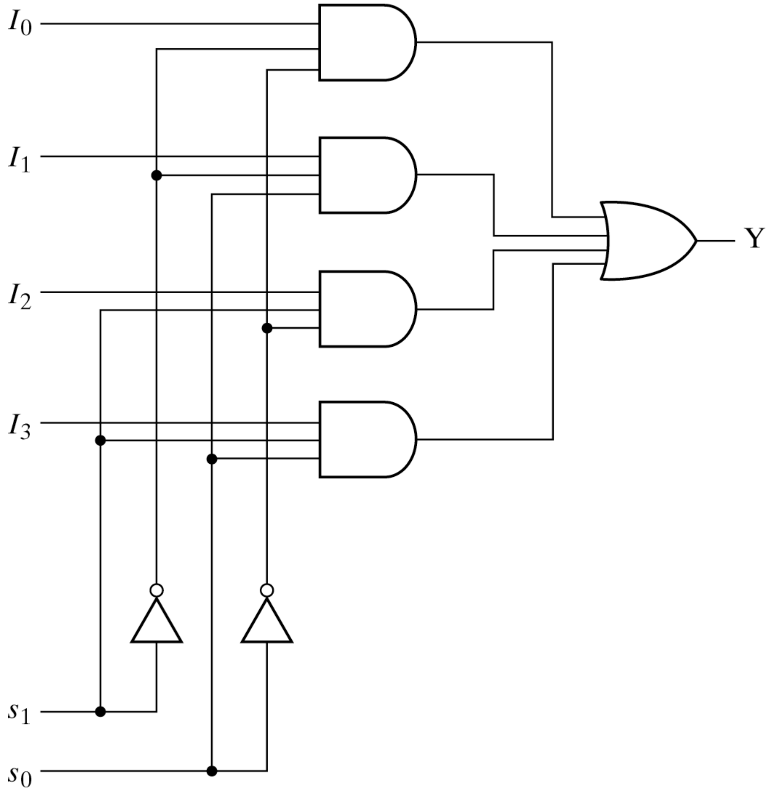


(a) Logic diagram

(b) Block diagram

Fig. 4-24 2-to-1-Line Multiplexer

4-to-1 Line Multiplexer



(a) Logic diagram

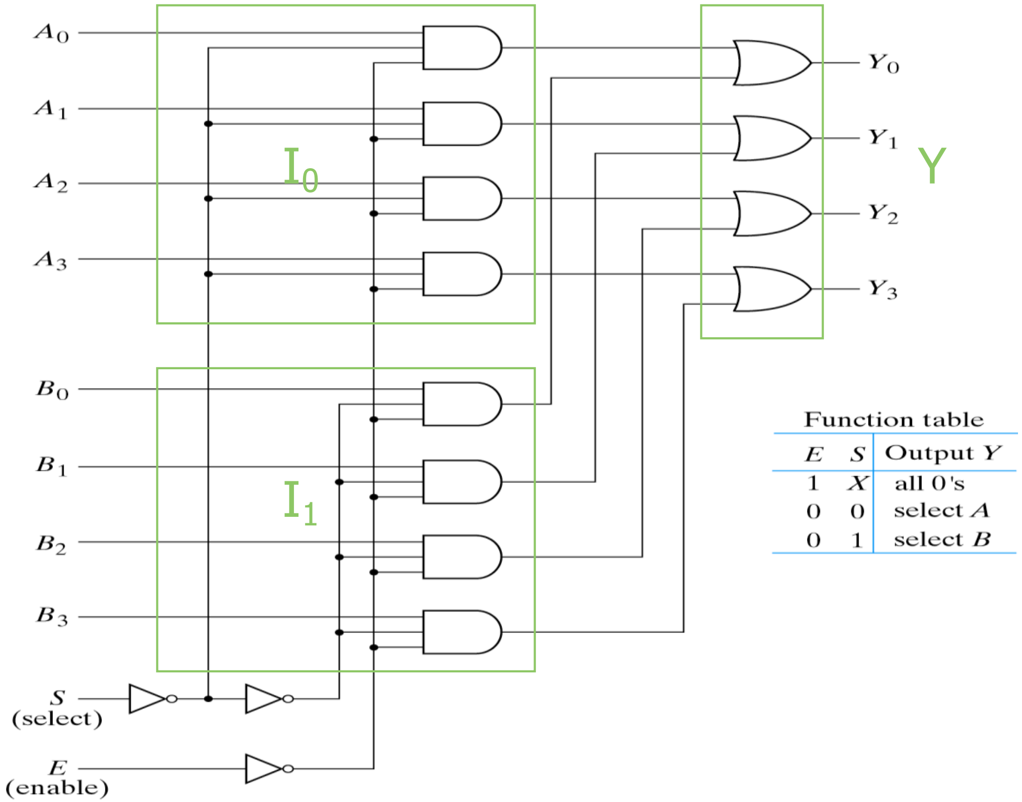
s_1	s_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

(b) Function table

Fig. 4-25 4-to-1-Line Multiplexer

Quadruple 2-to-1 Line Multiplexer

- Multiplexer circuits can be combined with common selection inputs to provide multiple-bit selection logic. Compare with Fig4-24.



Function table

E	S	Output Y
1	X	all 0's
0	0	select A
0	1	select B

Fig. 4-26 Quadruple 2-to-1-Line Multiplexer

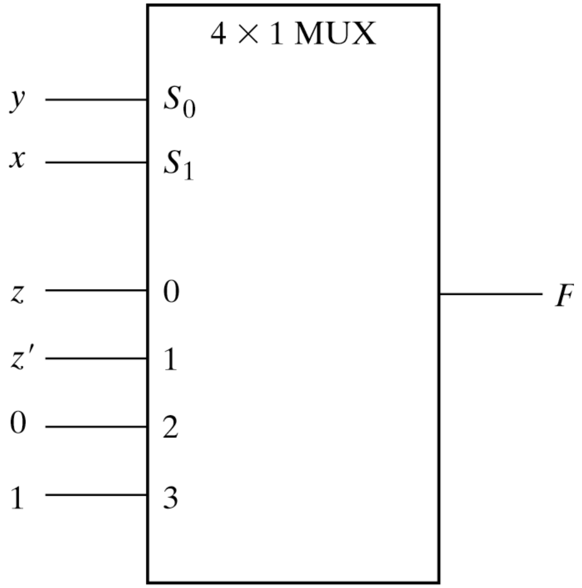
Boolean function implementation

- A more efficient method for implementing a Boolean function of n variables with a multiplexer that has n-1 selection inputs.

$$F(x, y, z) = \Sigma(1,2,6,7)$$

x	y	z	F	
0	0	0	0	$F = z$
0	0	1	1	
0	1	0	1	$F = z'$
0	1	1	0	
1	0	0	0	$F = 0$
1	0	1	0	
1	1	0	1	$F = 1$
1	1	1	1	

(a) Truth table



(b) Multiplexer implementation

Fig. 4-27 Implementing a Boolean Function with a Multiplexer

4-input function with a multiplexer

F(A, B, C, D) = Σ(1, 3, 4, 11, 12, 13, 14, 15)

A	B	C	D	F	
0	0	0	0	0	$F = D$
0	0	0	1	1	
0	0	1	0	0	$F = D$
0	0	1	1	1	
0	1	0	0	1	$F = D'$
0	1	0	1	0	
0	1	1	0	0	$F = 0$
0	1	1	1	0	
1	0	0	0	0	$F = 0$
1	0	0	1	0	
1	0	1	0	0	$F = D$
1	0	1	1	1	
1	1	0	0	1	$F = 1$
1	1	0	1	1	
1	1	1	0	1	$F = 1$
1	1	1	1	1	

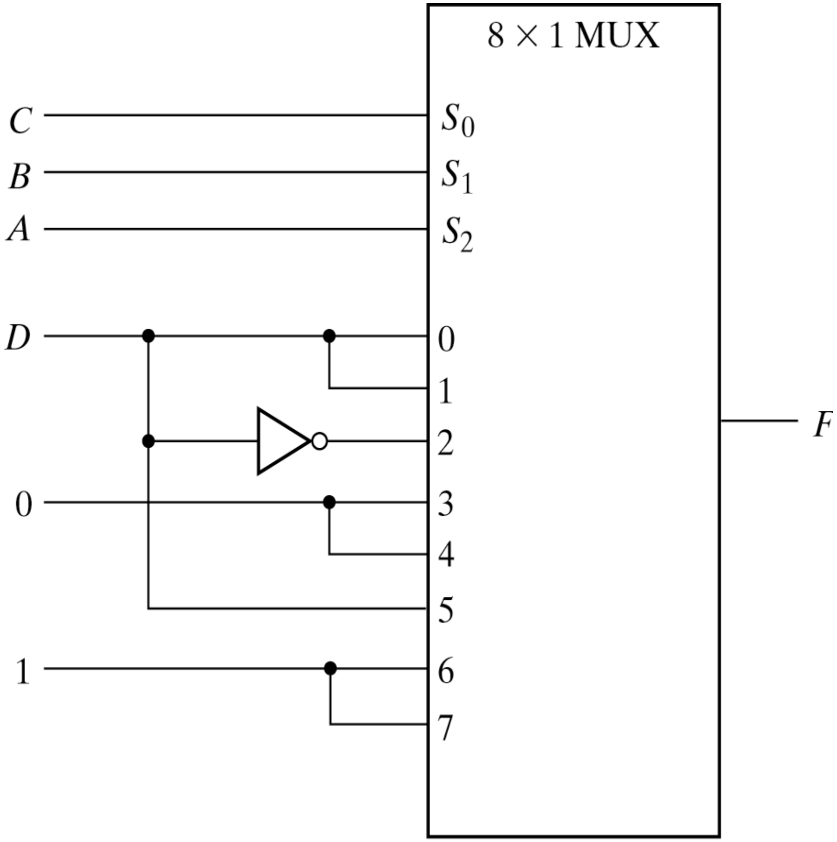


Fig. 4-28 Implementing a 4-Input Function with a Multiplexer

Three-State Gates

- A multiplexer can be constructed with three-state gates.

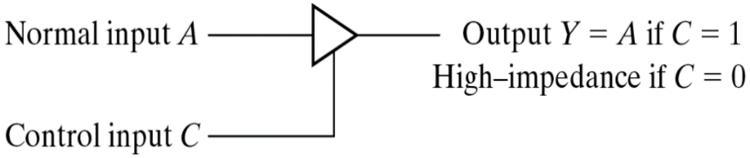
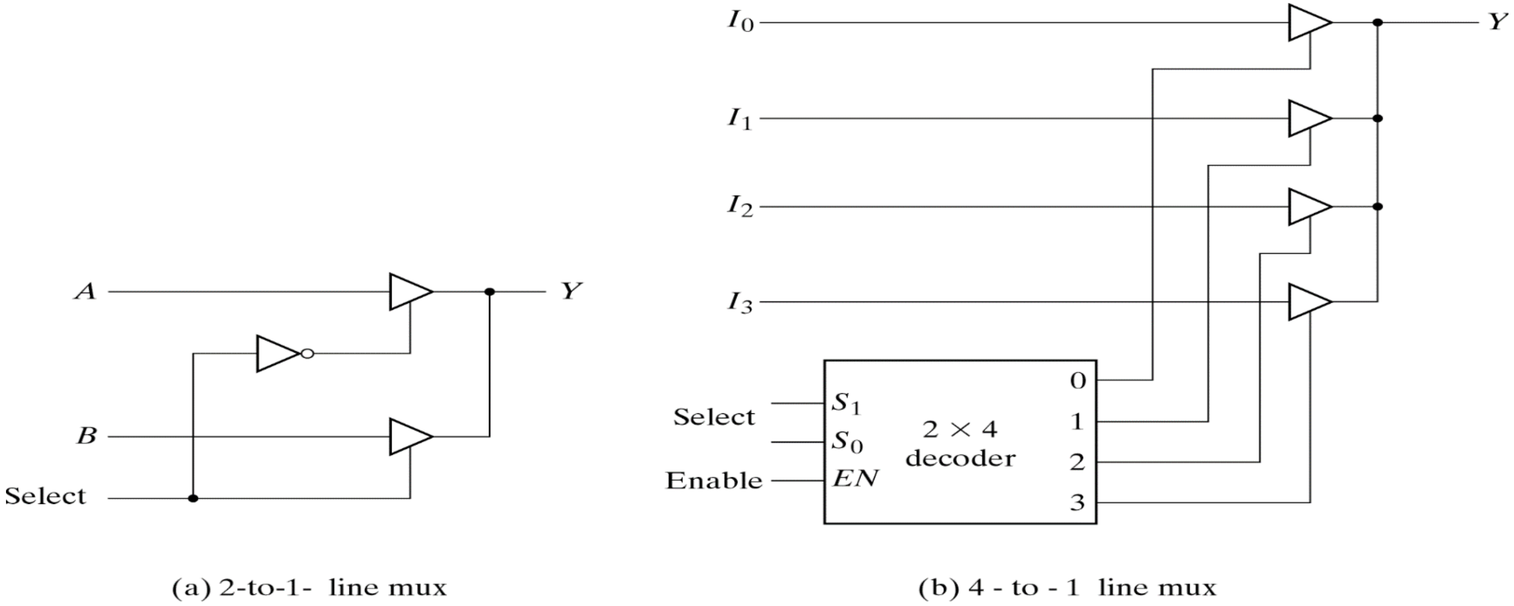


Fig. 4-29 Graphic Symbol for a Three-State Buffer



(a) 2-to-1- line mux

(b) 4 - to - 1 line mux

Fig. 4-30 Multiplexers with Three-State Gates

4-11. HDL for combinational circuits

- A module can be described in any one of the following modeling techniques:

1. Gate-level modeling using instantiation of primitive gates and user-defined modules.
2. Dataflow modeling using continuous assignment statements with keyword assign.
3. Behavioral modeling using procedural assignment statements with keyword always.

Gate-level Modeling

- A circuit is specified by its logic gates and their interconnection.
- Verilog recognizes 12 basic gates as predefined primitives.
- The logic values of each gate may be 1, 0, x(unknown), z(high-impedance).

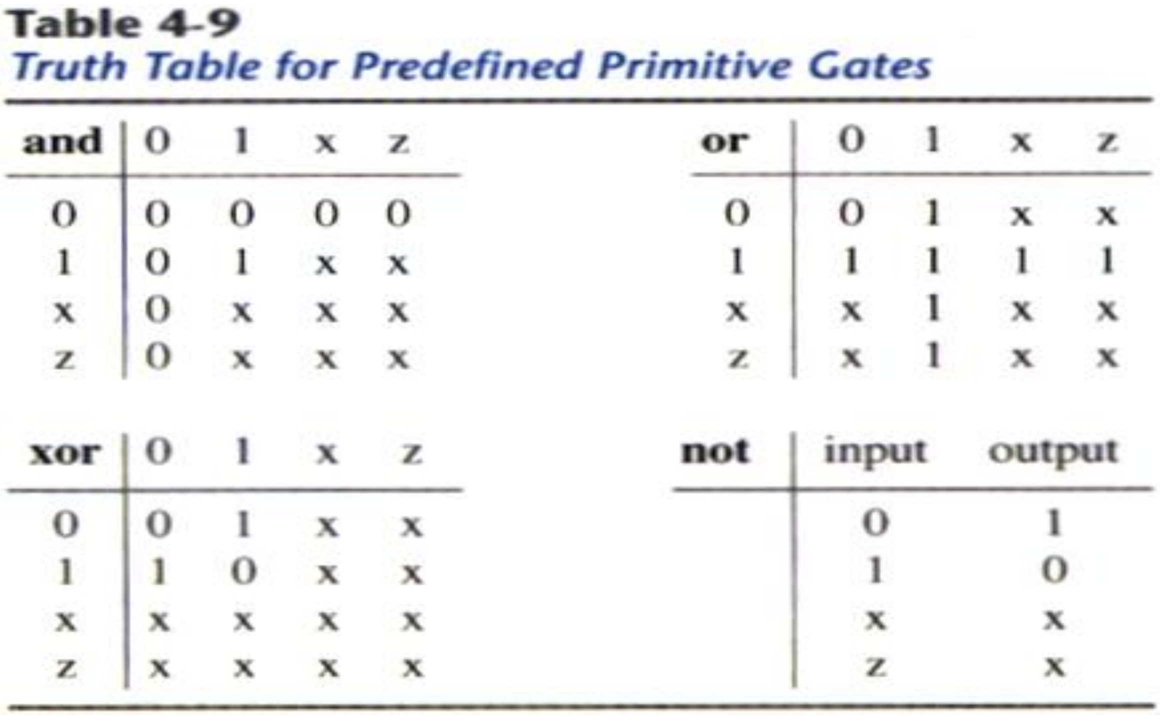


Table 4-9
Truth Table for Predefined Primitive Gates

and	0	1	x	z
0	0	0	0	0
1	0	1	x	x
x	0	x	x	x
z	0	x	x	x

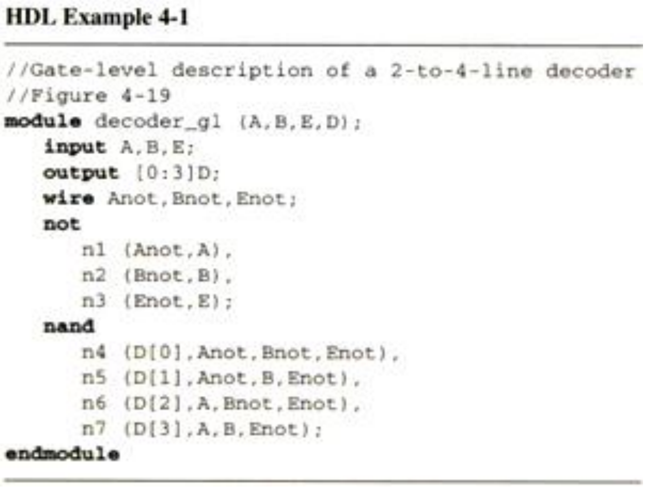
or	0	1	x	z
0	0	1	x	x
1	1	1	1	1
x	x	1	x	x
z	x	1	x	x

xor	0	1	x	z
0	0	1	x	x
1	1	0	x	x
x	x	x	x	x
z	x	x	x	x

not	input	output
	0	1
	1	0
	x	x
	z	x

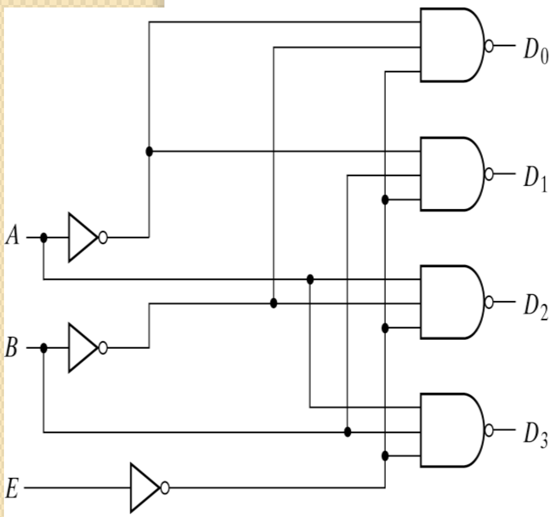
Gate-level description on Verilog code

The wire declaration is for interna



HDL Example 4-1

```
//Gate-level description of a 2-to-4-line decoder
//Figure 4-19
module decoder_gl (A,B,E,D);
   input A,B,E;
   output [0:3]D;
   wire Anot,Bnot,Enot;
   not
      n1 (Anot,A),
      n2 (Bnot,B),
      n3 (Enot,E);
   nand
      n4 (D[0],Anot,Bnot,Enot),
      n5 (D[1],Anot,B,Enot),
      n6 (D[2],A,Bnot,Enot),
      n7 (D[3],A,B,Enot);
endmodule
```



(a) Logic diagram

E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

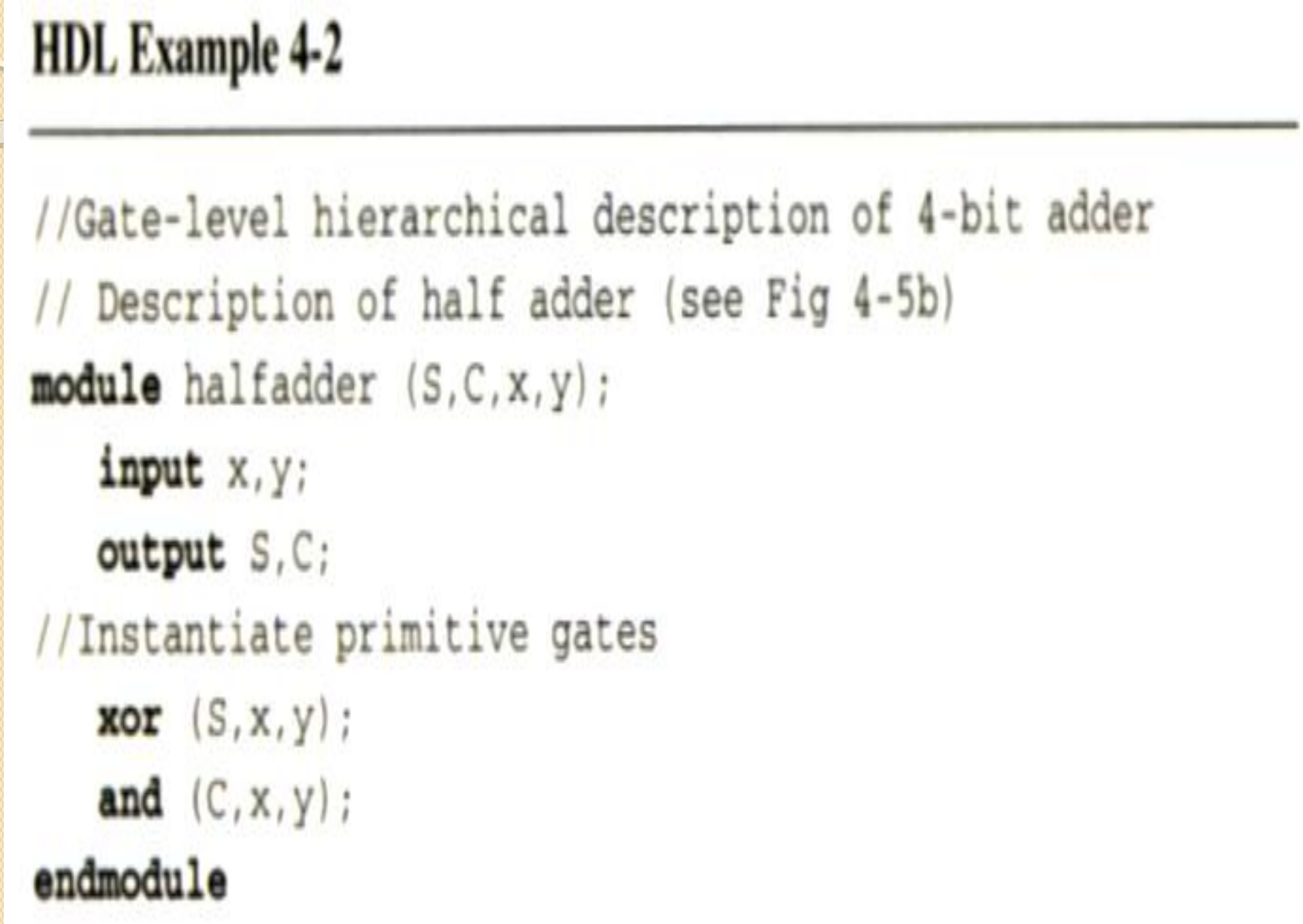
(b) Truth table

Fig. 4-19 2-to-4-Line Decoder with Enable Input

Design methodologies

- There are two basic types of design methodologies: top-down and bottom-up.
- Top-down: the top-level block is defined and then the sub-blocks necessary to build the top-level block are identified.(Fig.4-9 binary adder)
- Bottom-up: the building blocks are first identified and then combined to build the top-level block.(Example 4-2 4-bit adder)

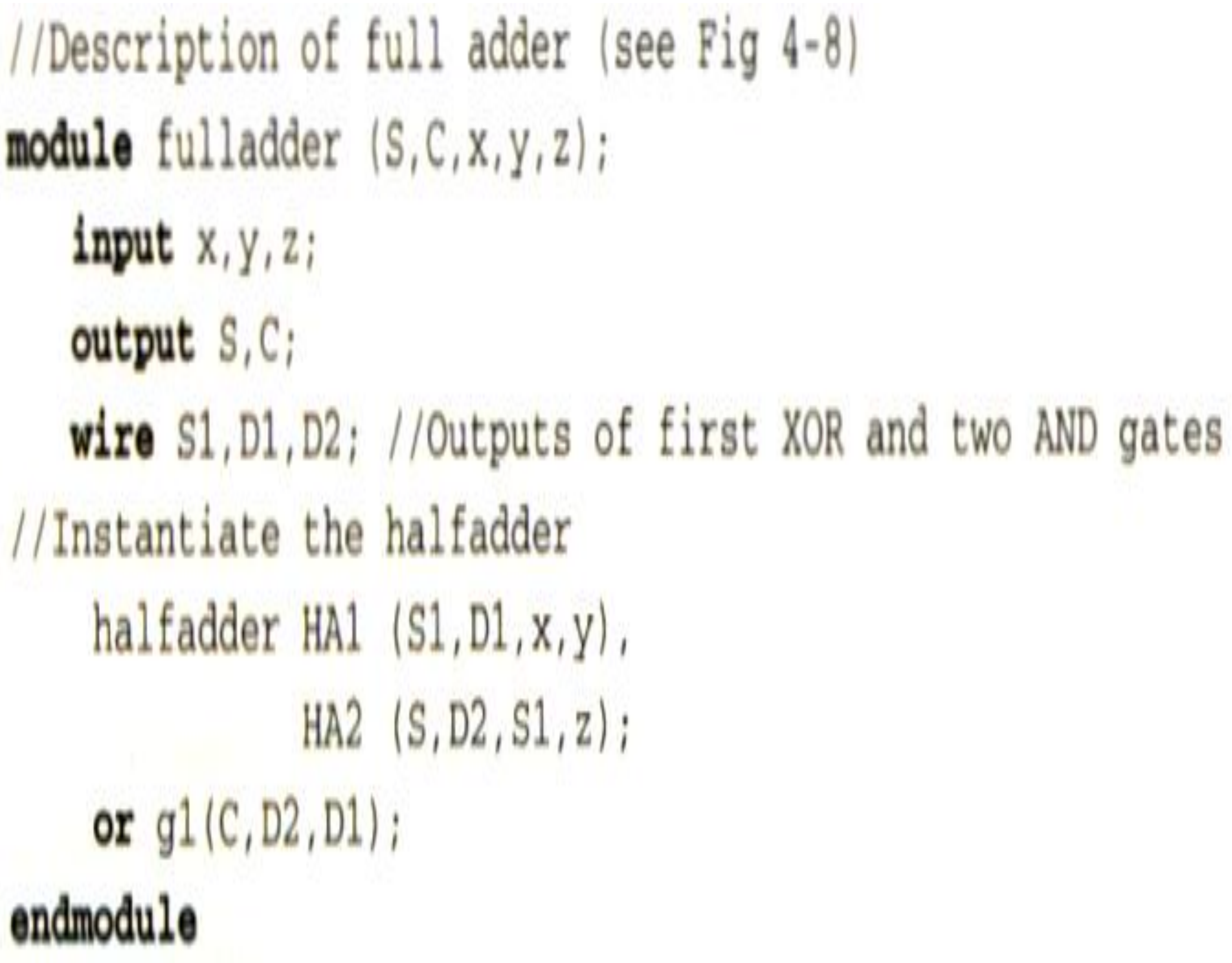
A bottom-up hierarchical description



HDL Example 4-2

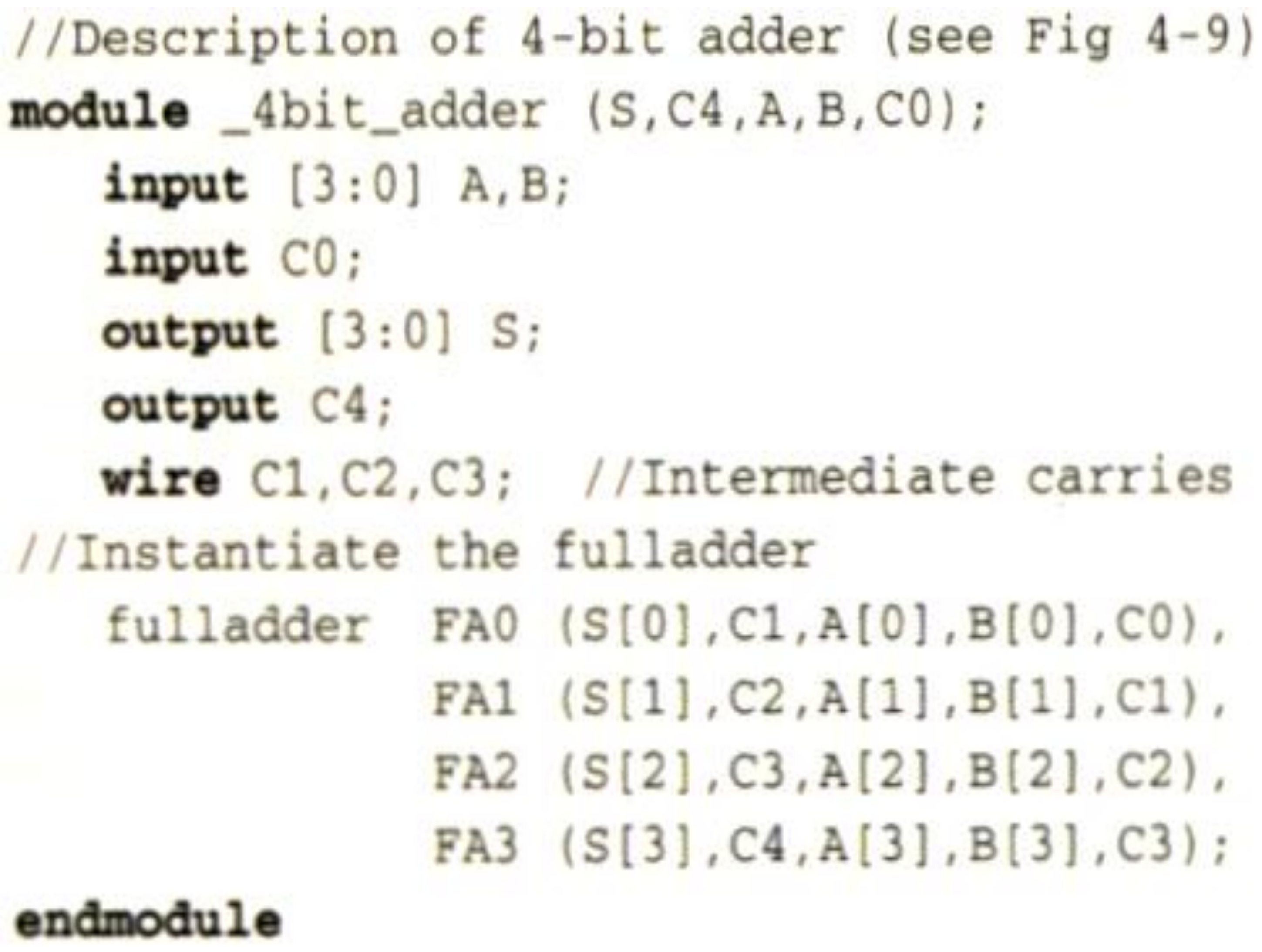
```
//Gate-level hierarchical description of 4-bit adder
// Description of half adder (see Fig 4-5b)
module halfadder (S,C,x,y);
   input x,y;
   output S,C;
//Instantiate primitive gates
   xor (S,x,y);
   and (C,x,y);
endmodule
```

Full-adder



```
//Description of full adder (see Fig 4-8)
module fulladder (S,C,x,y,z);
   input x,y,z;
   output S,C;
   wire S1,D1,D2; //Outputs of first XOR and two AND gates
//Instantiate the halfadder
    halfadder HA1 (S1,D1,x,y),
              HA2 (S,D2,S1,z);
    or g1(C,D2,D1);
endmodule
```

4-bit adder



```
//Description of 4-bit adder (see Fig 4-9)
module _4bit_adder (S,C4,A,B,C0);
   input [3:0] A,B;
   input C0;
   output [3:0] S;
   output C4;
   wire C1,C2,C3;  //Intermediate carries
//Instantiate the fulladder
   fulladder  FA0 (S[0],C1,A[0],B[0],C0),
              FA1 (S[1],C2,A[1],B[1],C1),
              FA2 (S[2],C3,A[2],B[2],C2),
              FA3 (S[3],C4,A[3],B[3],C3);
endmodule
```

Three state gates

Gates statement: gate name(output, input, control)

>> bufif1(OUT,A, control);

A = OUT when control = 1, OUT = z when control = 0;

>> notif0(Y, B, enable);

Y = B’ when enable = 0, Y = z when enable = 1;

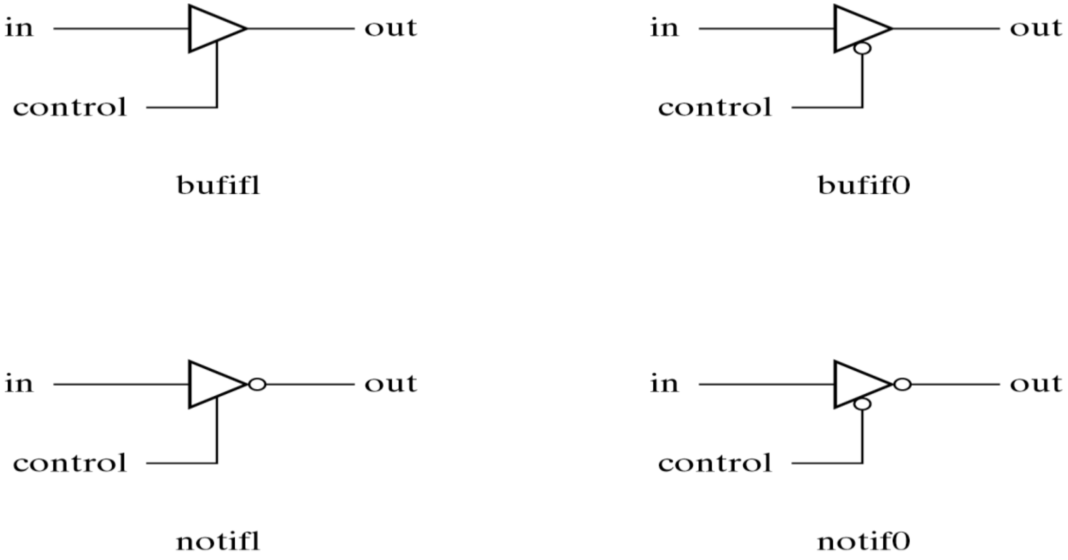


Fig. 4-31 Three-State Gates

2-to-1 multiplexer

- HDL uses the keyword tri to indicate that the output has multiple drivers.

```
module muxtri (A, B, select, OUT);
  input A,B,select;
  output OUT;
  tri OUT;
  bufif1  (OUT,A,select);
  bufif0 (OUT,B,select);
endmodule
```

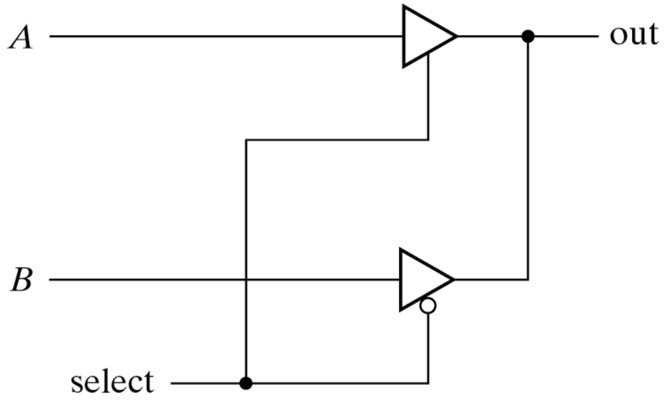


Fig. 4-32 2-to-1-Line Multiplexer with Three-State Buffers

Dataflow modeling

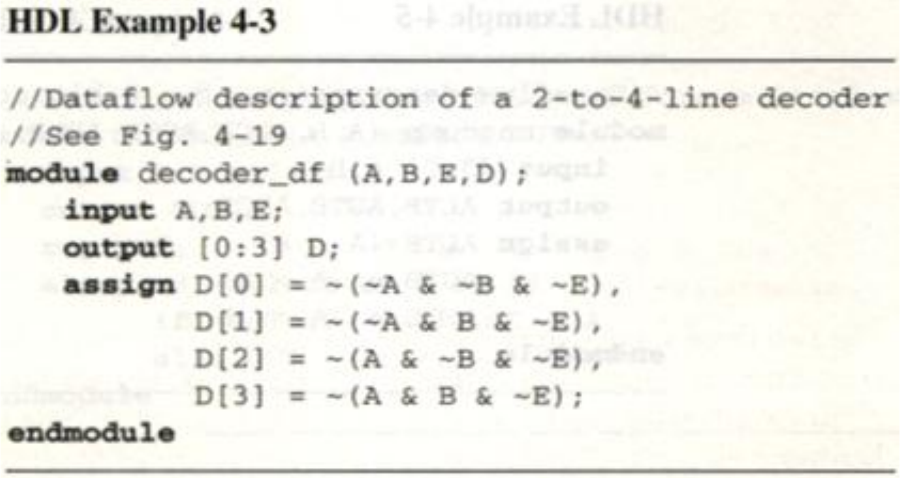
- It uses a number of operators that act on operands to produce desired results. Verilog HDL provides about 30 operator types.

Table 4-10

Symbol	Operation
+	binary addition
–	binary subtraction
&	bit-wise AND
|	bit-wise OR
^	bit-wise XOR
~	bit-wise NOT
==	equality
>	greater than
<	less than
{ }	concatenation
?:	conditional

Dataflow modeling

- A continuous assignment is a statement that assigns a value to a net.
- The data type net is used in Verilog HDL to represent a physical connection between circuit elements.
- A net defines a gate output declared by an output or wire.



HDL Example 4-3

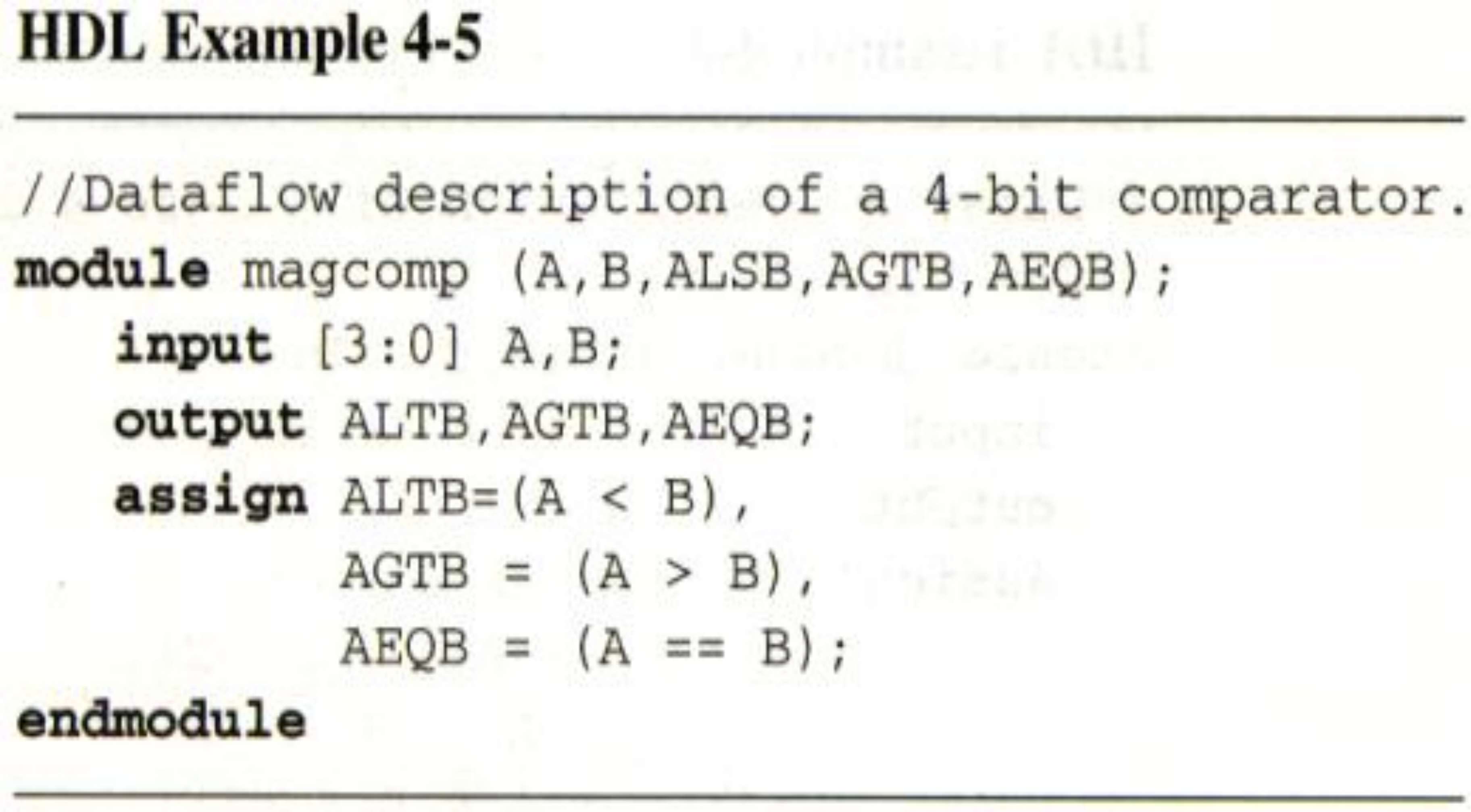
```
//Dataflow description of a 2-to-4-line decoder
//See Fig. 4-19
module decoder_df (A,B,E,D);
   input A,B,E;
   output [0:3] D;
   assign D[0] = ~(~A & ~B & ~E),
          D[1] = ~(~A & B & ~E),
          D[2] = ~(A & ~B & ~E),
          D[3] = ~(A & B & ~E);
endmodule
```

Dataflow description of 4-bit adder

HDL Example 4-4

```
//Dataflow description of 4-bit adder
module binary_adder (A,B,Cin,SUM,Cout);
input  [3:0]  A,B;
input  Cin;
output  [3:0]  SUM;
output  Cout;
assign  {Cout,SUM} = A + B +Cin;
endmodule
```

Data flow description of a 4-bit comparator

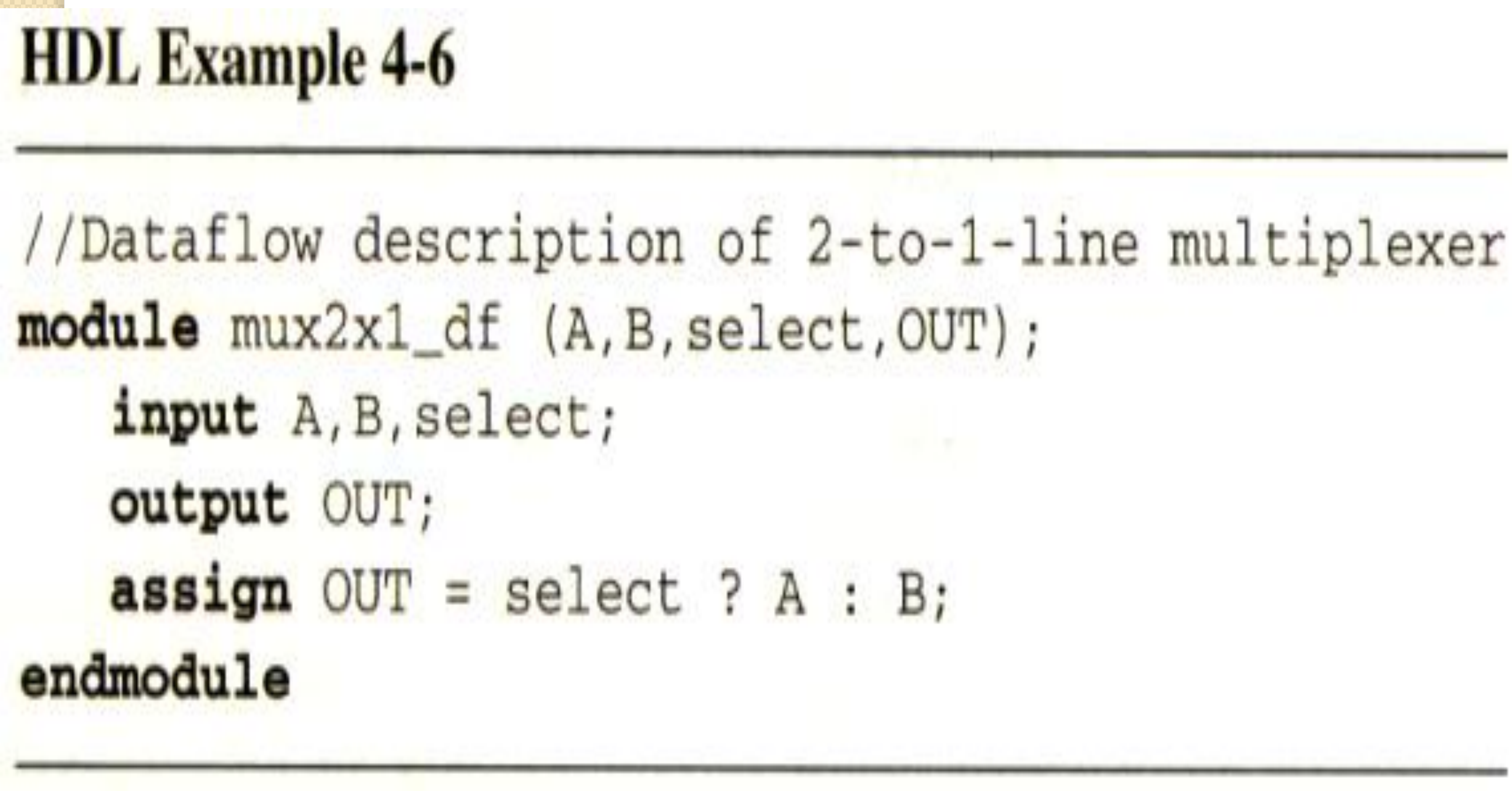


HDL Example 4-5

```
//Dataflow description of a 4-bit comparator.
module magcomp (A,B,ALSB,AGTB,AEQB);
   input [3:0] A,B;
   output ALTB,AGTB,AEQB;
   assign ALTB=(A < B),
          AGTB = (A > B),
          AEQB = (A == B);
endmodule
```

Dataflow description of 2-1 multiplexer

- Conditional operator(? :)
- Condition? true-expression : false-expression;



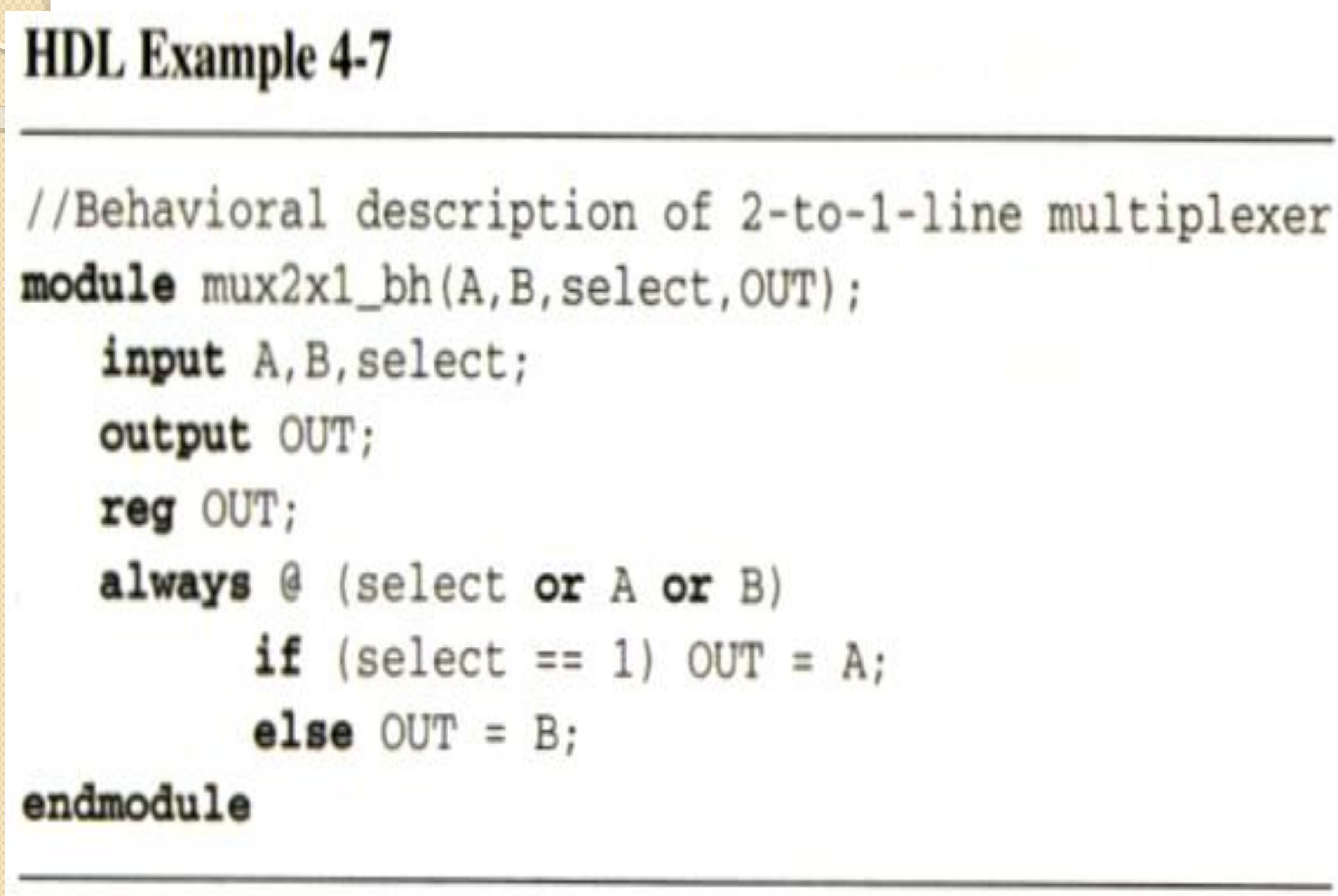
HDL Example 4-6

```
//Dataflow description of 2-to-1-line multiplexer
module mux2x1_df (A,B,select,OUT);
   input A,B,select;
   output OUT;
   assign OUT = select ? A : B;
endmodule
```

Behavioral modeling

- It is used mostly to describe sequential circuits, but can be used also to describe combinational circuits.
- Behavioral descriptions use the keyword always followed by a list of procedural assignment statements.
- The target output of procedural assignment statements must be of the reg data type. Contrary to the wire data type, where the target output of an assignment may be continuously updated, a reg data type retains its value until a new value is assigned.

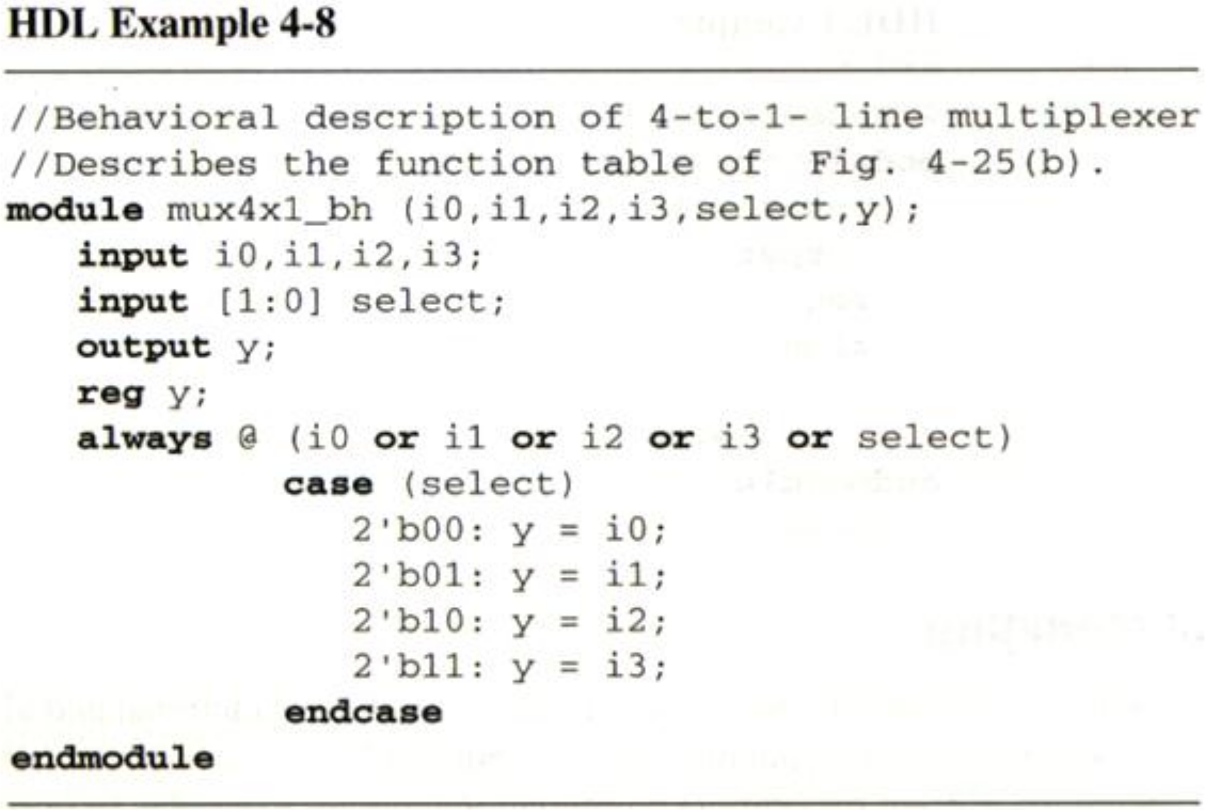
Behavioral description of 2-1 multiplexer



HDL Example 4-7

```
//Behavioral description of 2-to-1-line multiplexer
module mux2x1_bh(A,B,select,OUT);
   input A,B,select;
   output OUT;
   reg OUT;
   always @ (select or A or B)
         if (select == 1) OUT = A;
         else OUT = B;
endmodule
```

4-to-1-line Multiplexer



HDL Example 4-8

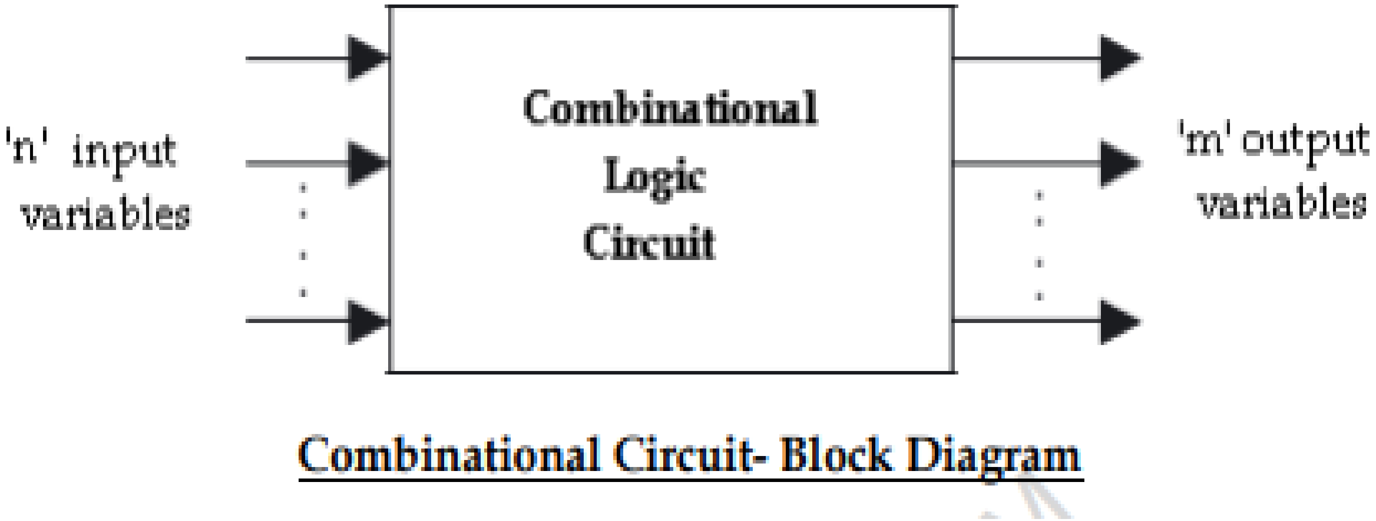
```
//Behavioral description of 4-to-1- line multiplexer
//Describes the function table of  Fig. 4-25(b).
module mux4x1_bh (i0,i1,i2,i3,select,y);
   input i0,i1,i2,i3;
   input [1:0] select;
   output y;
   reg y;
   always @ (i0 or i1 or i2 or i3 or select)
            case (select)
               2'b00: y = i0;
               2'b01: y = i1;
               2'b10: y = i2;
               2'b11: y = i3;
            endcase
endmodule
```

Unit-III & IV

Synchronous and Asynchronous Sequential Circuits

Combinational Logic

- Combinational Logic:
  - Output depends only on current input
  - Has no memory



Combinational Circuit- Block Diagram

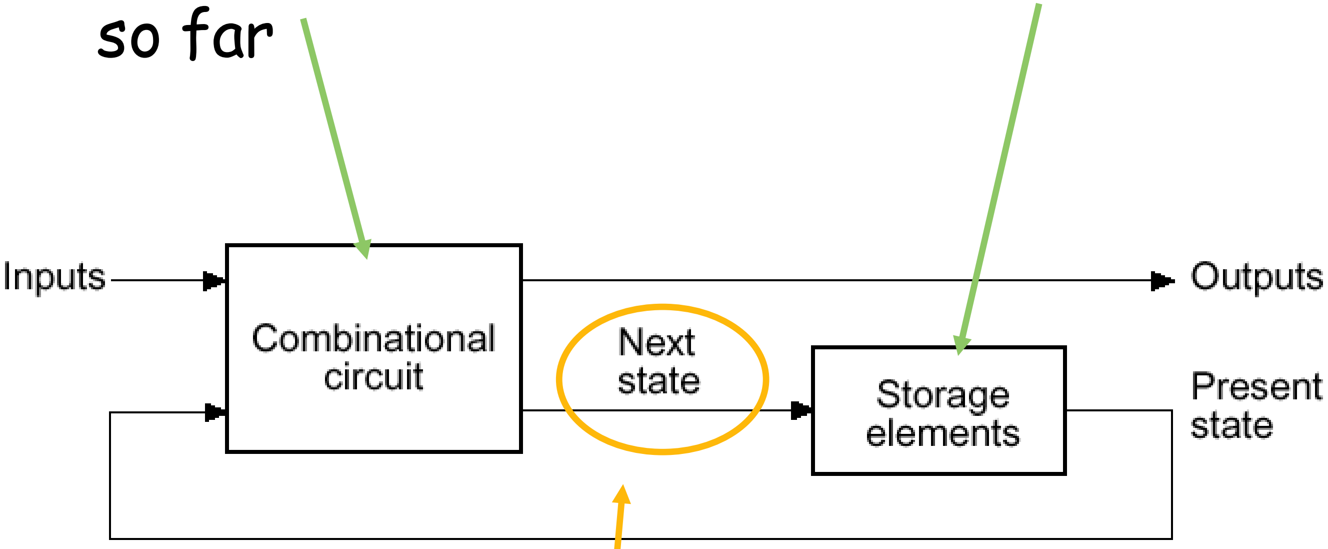
Sequential Logic

- Sequential Logic:
  - Output depends not only on current input but also on past input values, e.g., design a counter
  - Need some type of memory to remember the past input values

Sequential Circuits

Circuits that we have learned so far

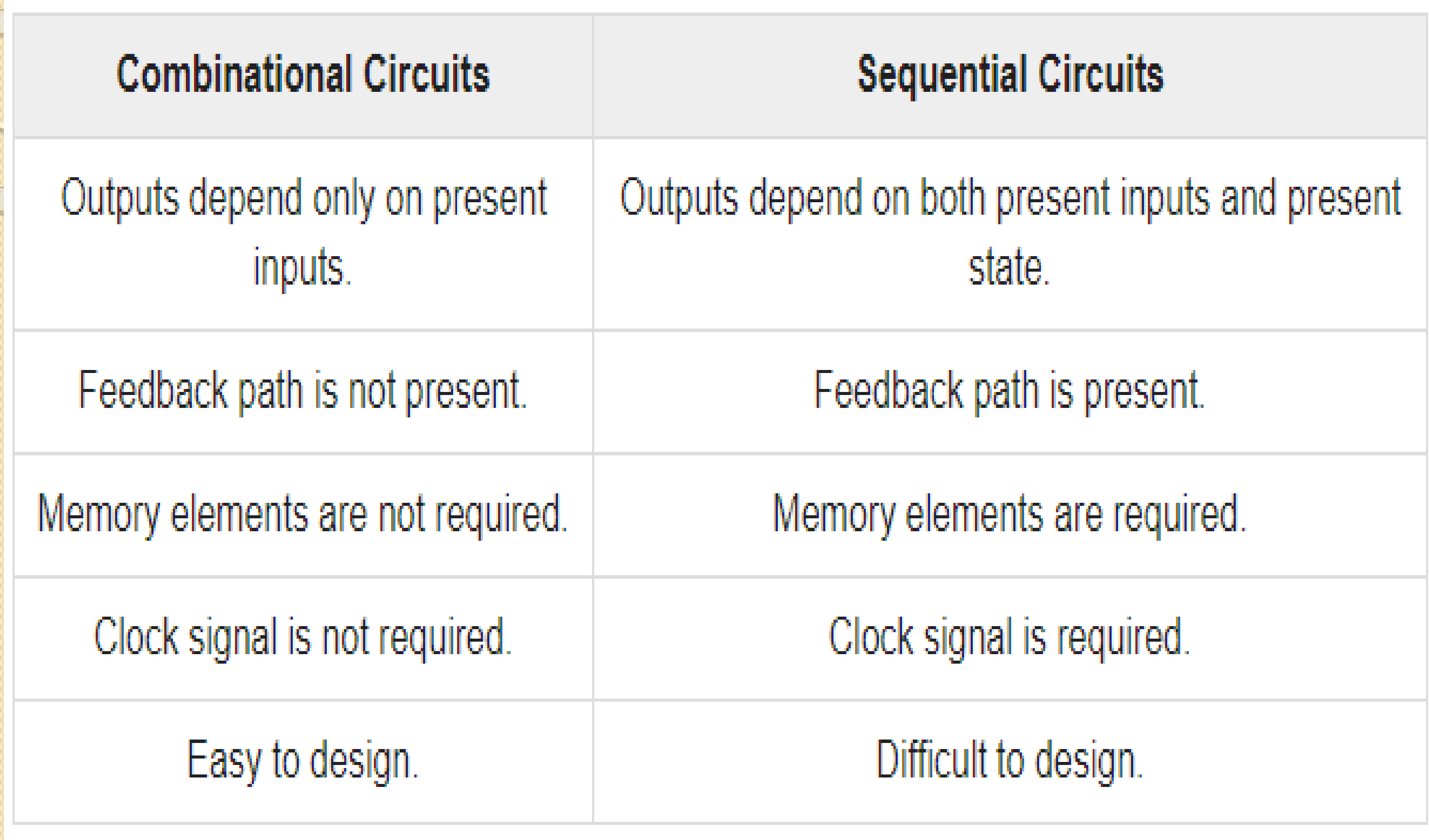
Information Storing Circuits



Timed "States"

Sequential Logic: Concept

- Sequential Logic circuits remember past inputs and past circuit state.
- Outputs from the system are "fed back" as new inputs
  - With gate delay and wire delay
- The storage elements are circuits that are capable of storing binary information: memory.



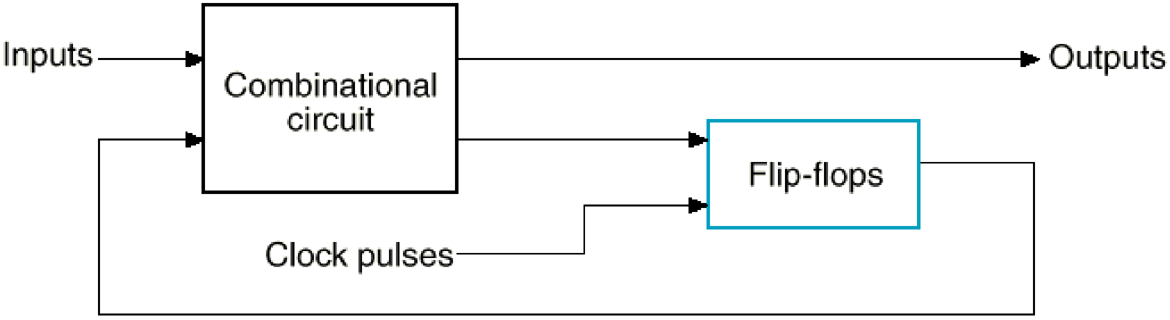
Combinational Circuits	Sequential Circuits
Outputs depend only on present inputs.	Outputs depend on both present inputs and present state.
Feedback path is not present.	Feedback path is present.
Memory elements are not required.	Memory elements are required.
Clock signal is not required.	Clock signal is required.
Easy to design.	Difficult to design.

Synchronous vs. Asynchronous

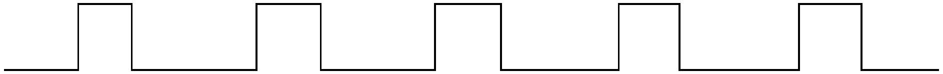
There are two types of sequential circuits:

- **Synchronous** sequential circuit: circuit output changes only at some discrete instants of time. This type of circuits achieves synchronization by using a timing signal called the *clock*.
- **Asynchronous** sequential circuit: circuit output can change at **any** time (clockless).

Synchronous Sequential Circuits: Flip flops as state memory



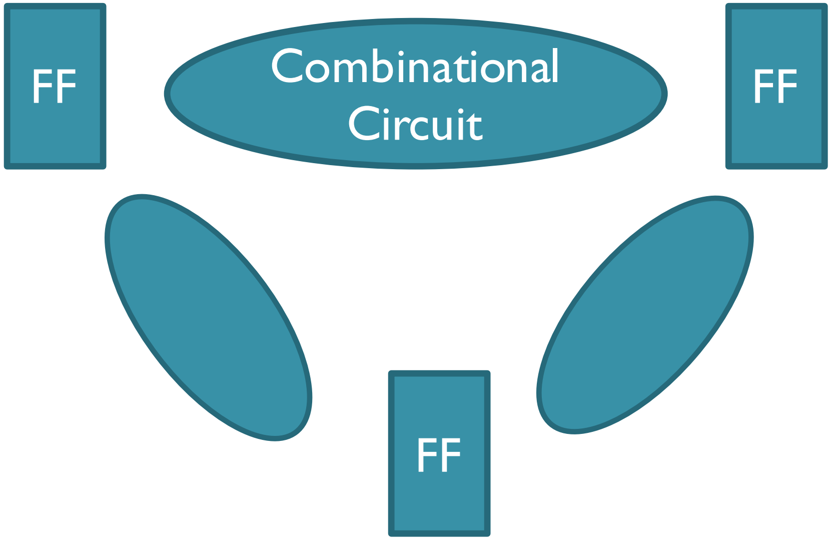
(a) Block diagram



(b) Timing diagram of clock pulses

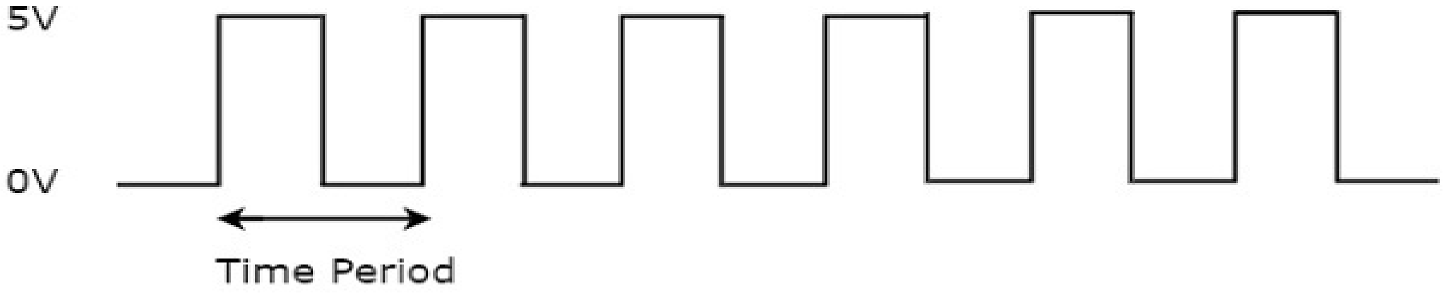
- The flip-flops receive their inputs from the combinational circuit and also from a clock signal with pulses that occur at fixed intervals of time, as shown in the timing diagram.

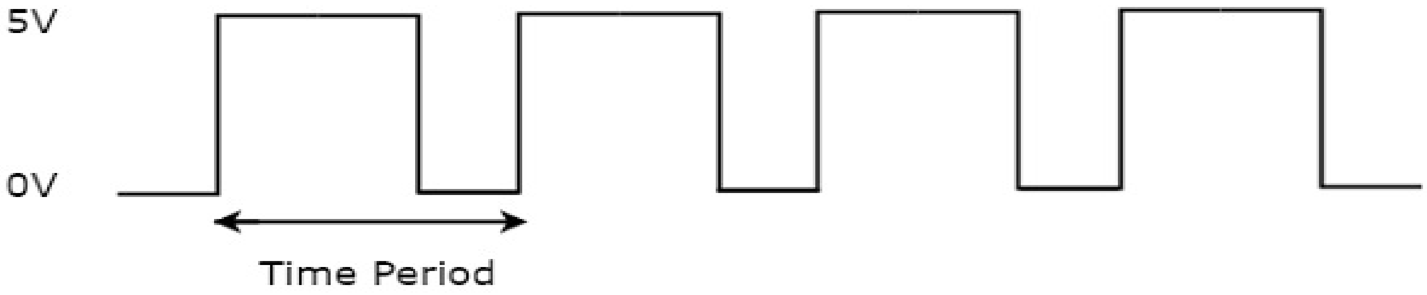
Clock Period



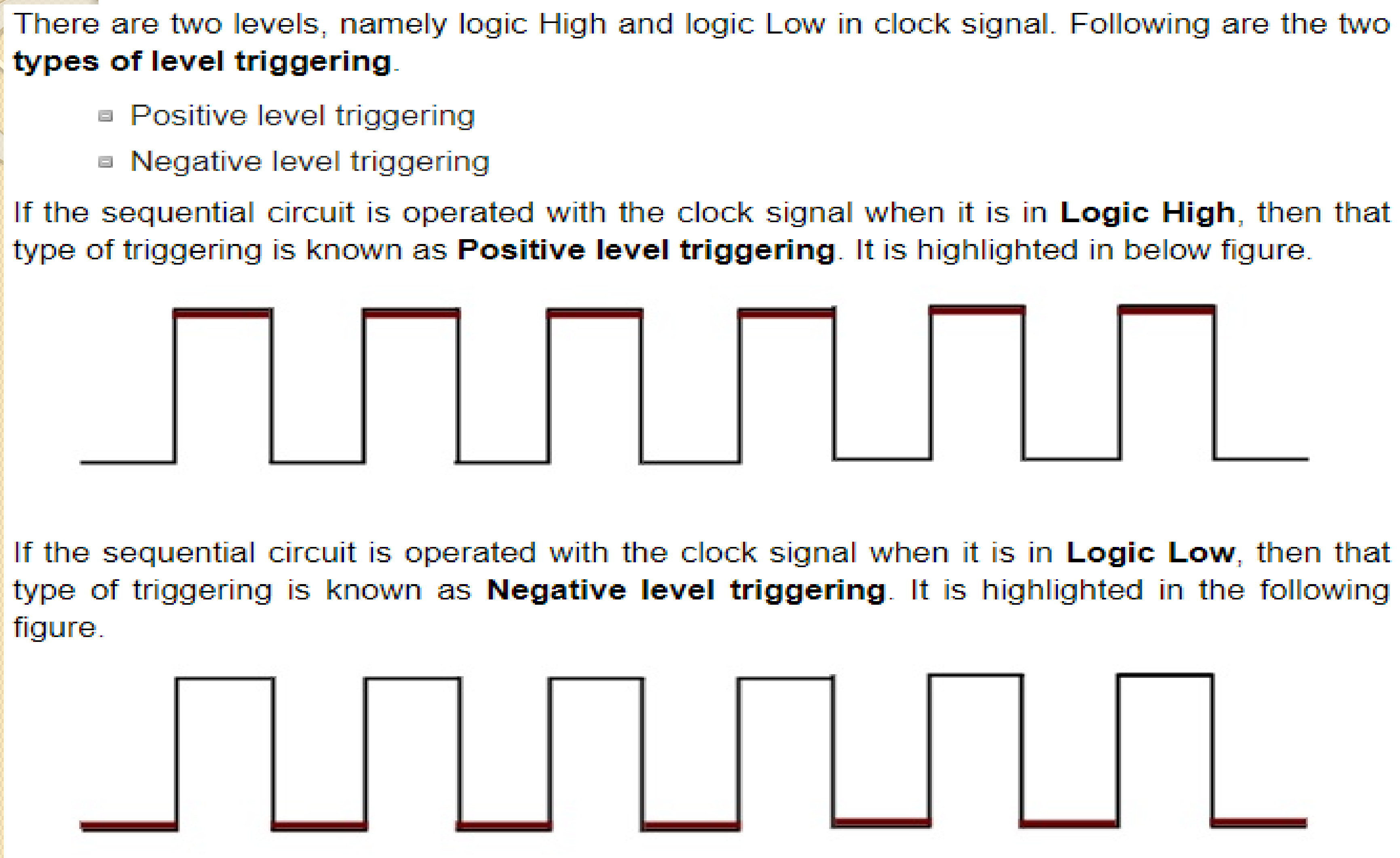
Smallest clock period = largest combinational circuit delay between any two directly connected FF, subjected to impact of FF setup time.

CLOCK SIGNAL





TYPES OF TRIGGERING



There are two levels, namely logic High and logic Low in clock signal. Following are the two **types of level triggering**.

- Positive level triggering
- Negative level triggering

If the sequential circuit is operated with the clock signal when it is in **Logic High**, then that type of triggering is known as **Positive level triggering**. It is highlighted in below figure.

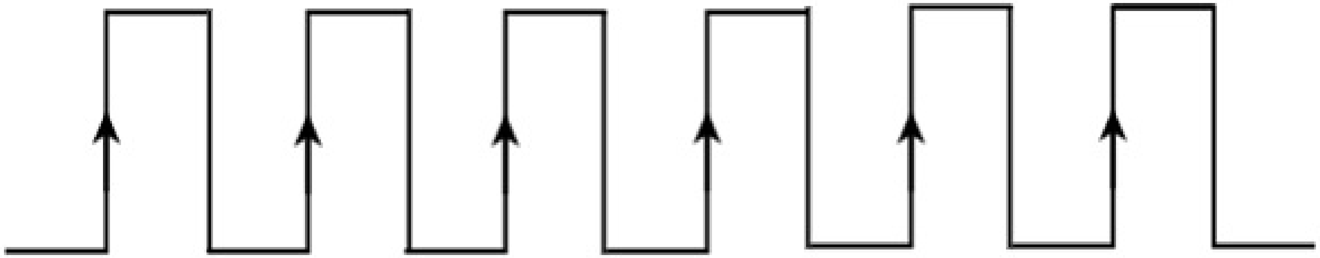
If the sequential circuit is operated with the clock signal when it is in **Logic Low**, then that type of triggering is known as **Negative level triggering**. It is highlighted in the following figure.

EDGE TRIGERRING

- There are two types of transitions that occur in clock signal. That means, the clock signal transitions either from Logic Low to Logic High or Logic High to Logic Low.
- Following are the two **types of edge triggering** based on the transitions of clock signal.
- Positive edge triggering
- Negative edge triggering

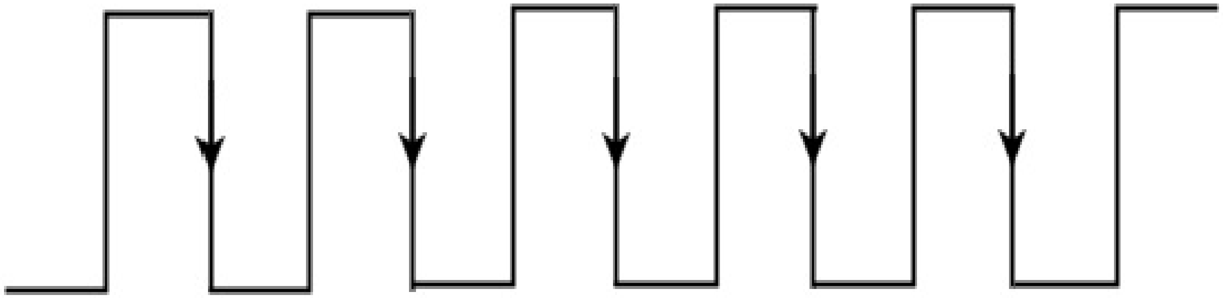
POSITIVE EDGE TRIGGERING

- If the sequential circuit is operated with the clock signal that is transitioning from Logic Low to Logic High, then that type of triggering is known as **Positive edge triggering**. It is also called as rising edge triggering. It is shown in the following figure.



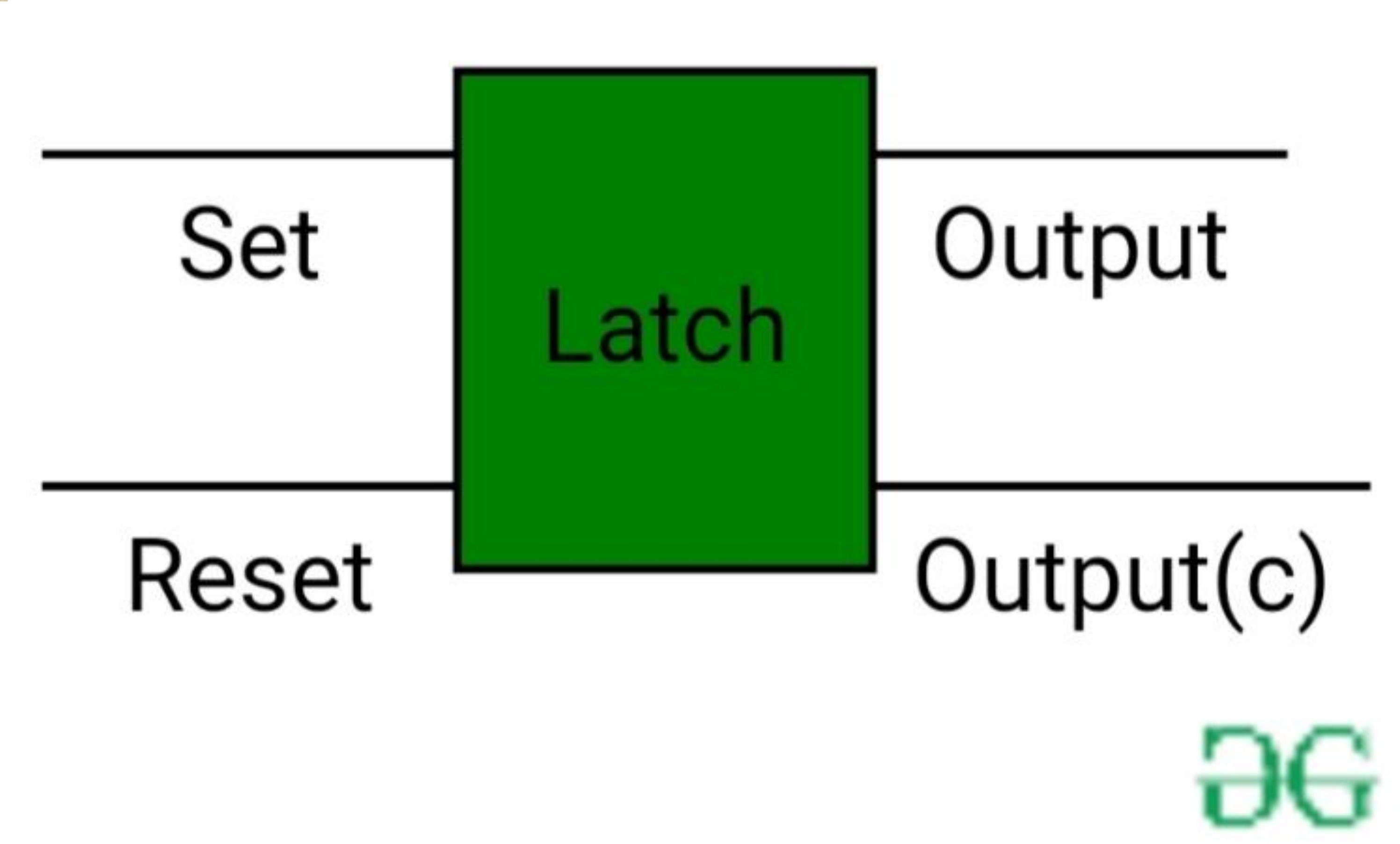
NEGATIVE EDGE TRIGGERING

- If the sequential circuit is operated with the clock signal that is transitioning from Logic High to Logic Low, then that type of triggering is known as **Negative edge triggering**. It is also called as falling edge triggering. It is shown in the following figure.



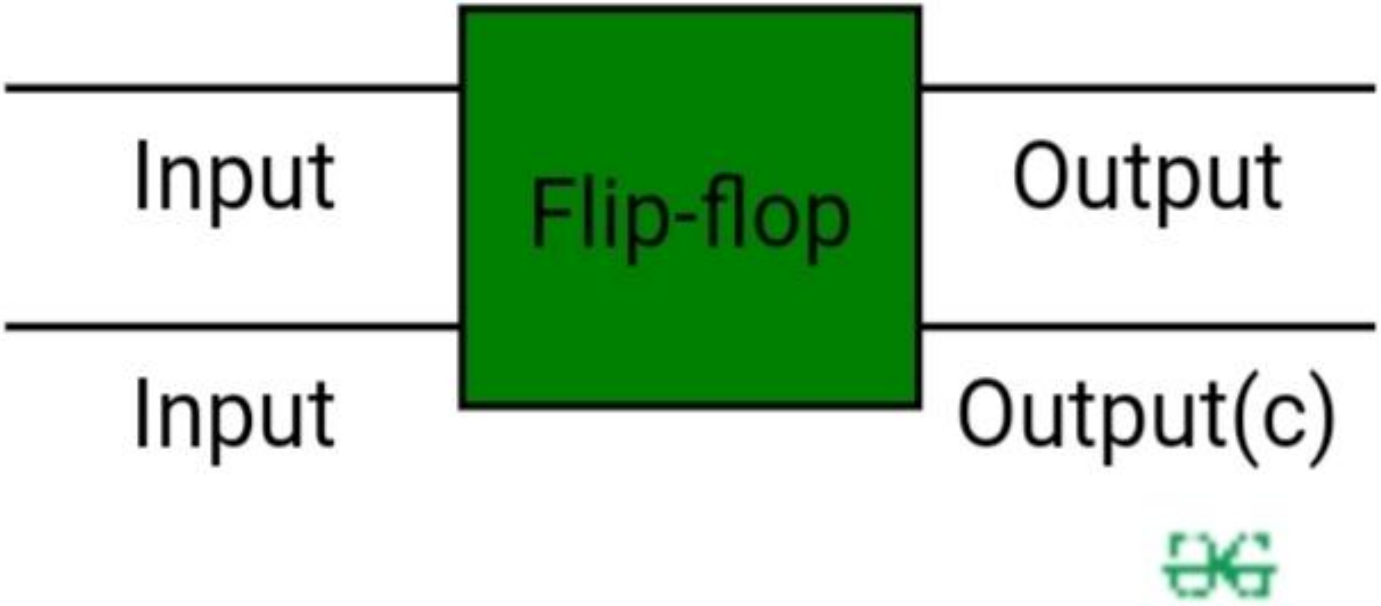
LATCH

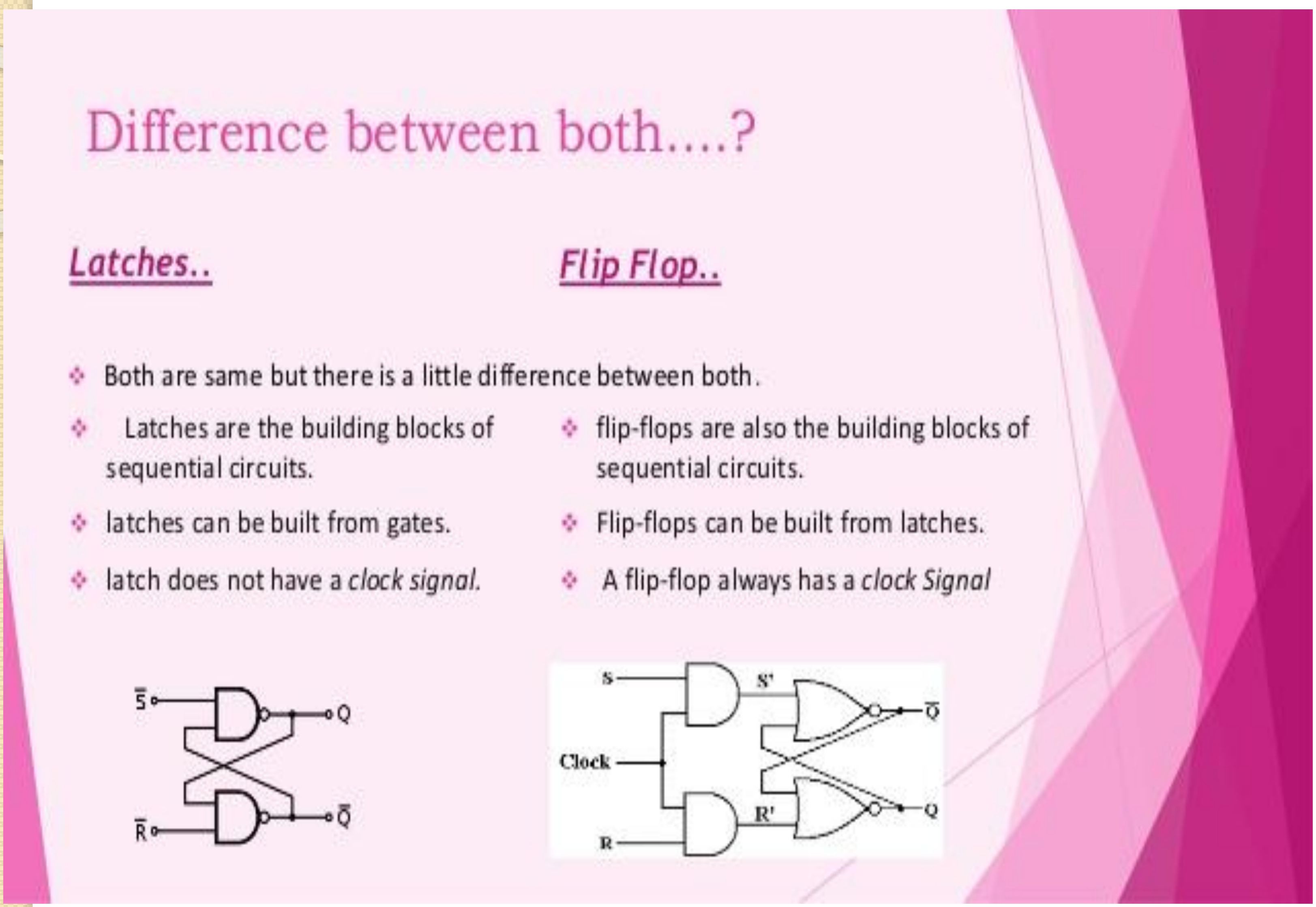
- Latch is one kind of a logic circuit, and it is also known as a bistable-multivibrator.
- Latches are useful for the design of the asynchronous sequential circuit
- The working of these circuits can be done in 2-states based on the enable signal being high or else low.
- When the latch circuit is the in an active high state, then both the i/ps are low.
- Similarly, when the latch circuit is then an active low state, then both the i/ps are high.
- The latches can be classified into different types which include
- SR Latch, Gated S-R Latch, D latch, Gated D Latch, JK Latch, and T Latch.


Set
Latch
Output
Reset
Output(c)

FLIPFLOP

- Flip flops are also considered as sequential logic circuits as their present output value depends on present and past input and past output.
- when input is changed from one value to another then the stored bit changes only when there is a change in the clock signal either from low level to high or high to a low level.
- Therefore, we can say a flip changes the output according to input but with respect to the clock signal.


Input
Flip-flop
Output
Input
Output(c)



Difference between both….?

Latches..

Flip Flop..

- Both are same but there is a little difference between both.

Latches..

- Latches are the building blocks of sequential circuits.
- latches can be built from gates.
- latch does not have a *clock signal.*

Flip Flop..

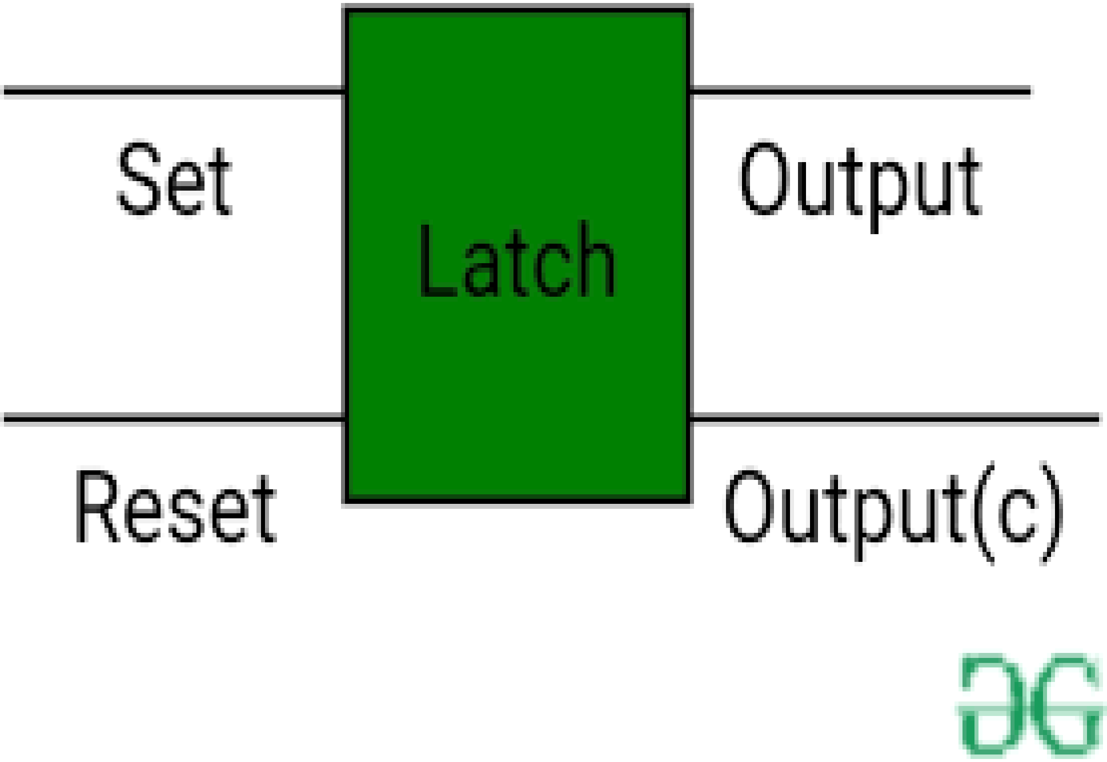
- flip-flops are also the building blocks of sequential circuits.
- Flip-flops can be built from latches.
- A flip-flop always has a *clock Signal*

S.No.	FLIP	LATCHES
1	Flip-flop is a bistable device i.e., it has two stable states that are represented as 0 and 1.	Latch is also a bistable device whose states are also represented as 0 and 1.
2	It is a edge triggered device.	It is a level triggered device.
3	Gates like NOR, NOT,AND, NAND are building blocks of flip flops.	These are also made up of gates.
4	They are classified into asynchronous or synchronous flipflops.	There is no such classification in latches.
5	It checks the inputs but changes the output only at times defined by the clock signal or any other control signal.	It checks the inputs continuously and responds to the changes in inputs immediately.
6	More power is consumed by the Flip-Flop.	Less power is consumed by the Latches.
7	ex:D Flip-flop, JK Flip-flop	ex:SR Latch, D Latch

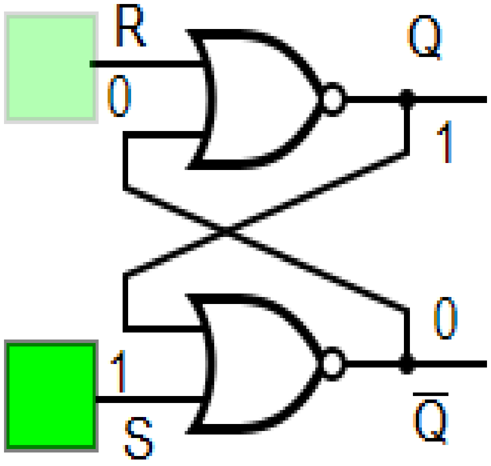
SR LATCH

- A bistable multivibrator has two stable states, as indicated by the prefix bi in its name.
- Typically, one state is referred to as set and the other as reset. The simplest bistable device, therefore, is known as a set-reset, or S-R, latch.
- To create an S-R latch, we can wire two NOR gates in such a way that the output of one feeds back to the input of another, and vice versa, like this:

LOGIC SYMBOL



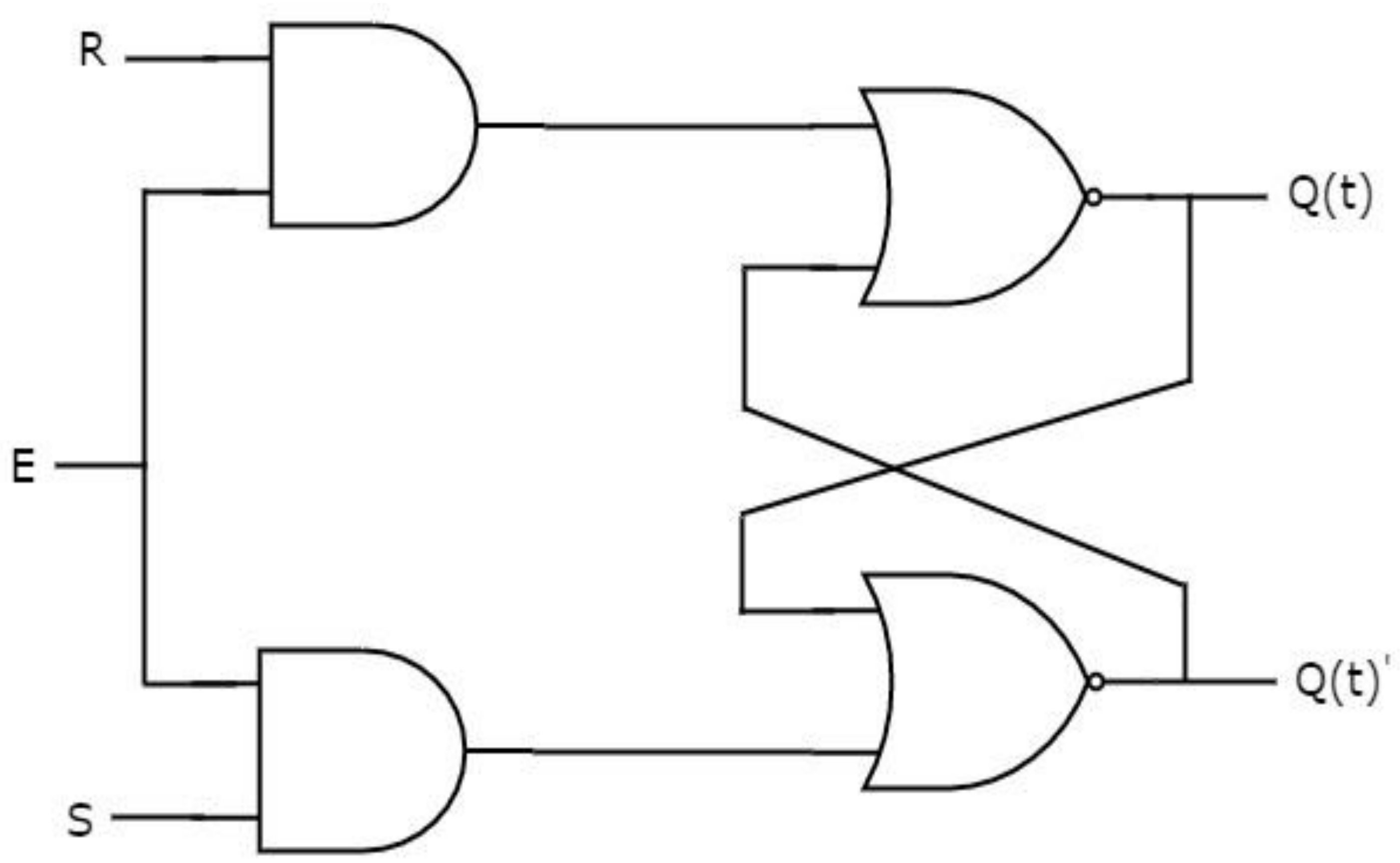
NOR BASED SR LATCH



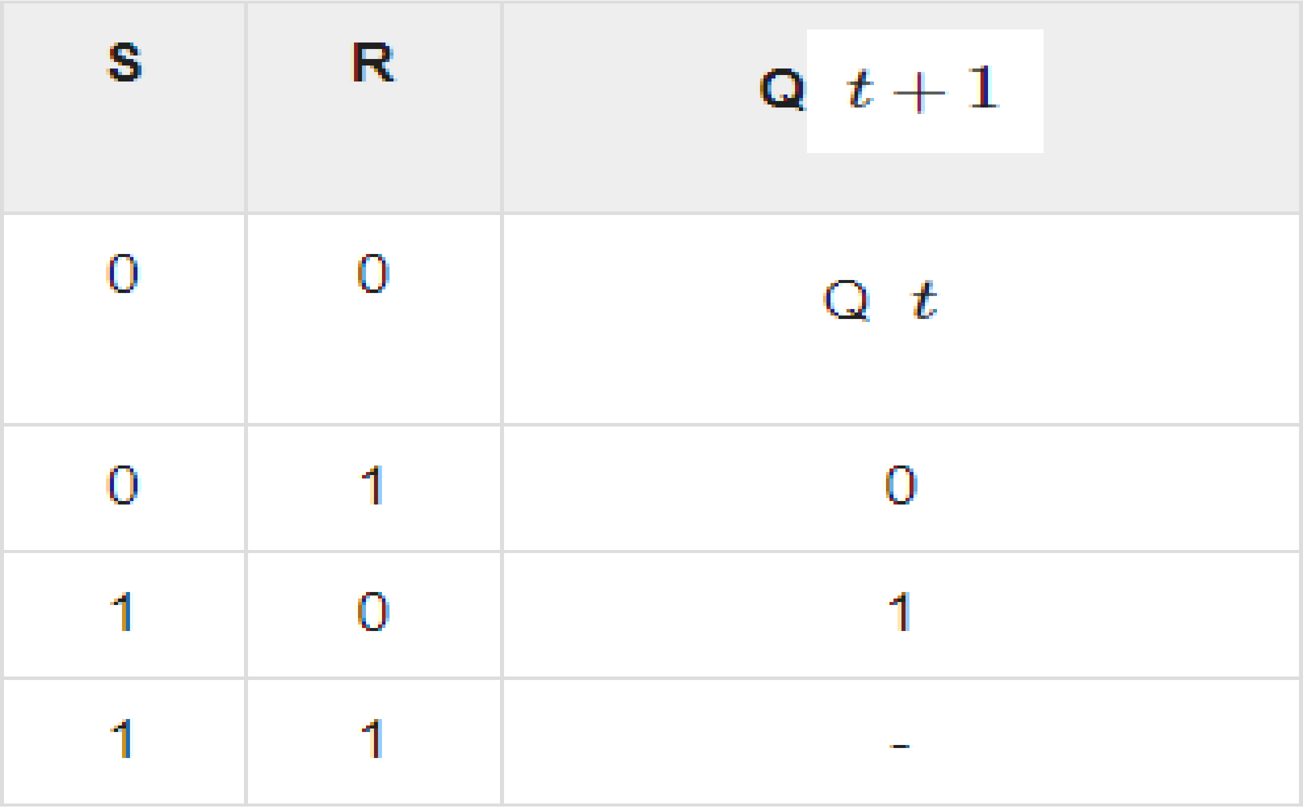
TRUTH TABLE

INPUTS		OUTPUTS	
S	R	Q	$\overline{Q}$
0	0	Q_0	$\overline{Q}_0$
0	1	0	1
1	0	1	0
1	1	X	X

CIRCUIT DIAGRAM-SR Latch



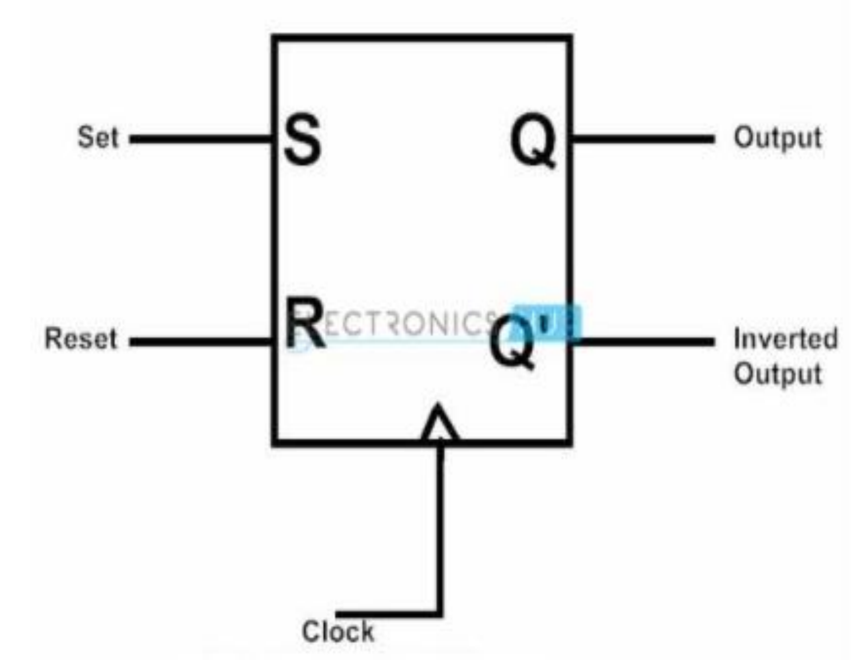
TRUTH TABLE

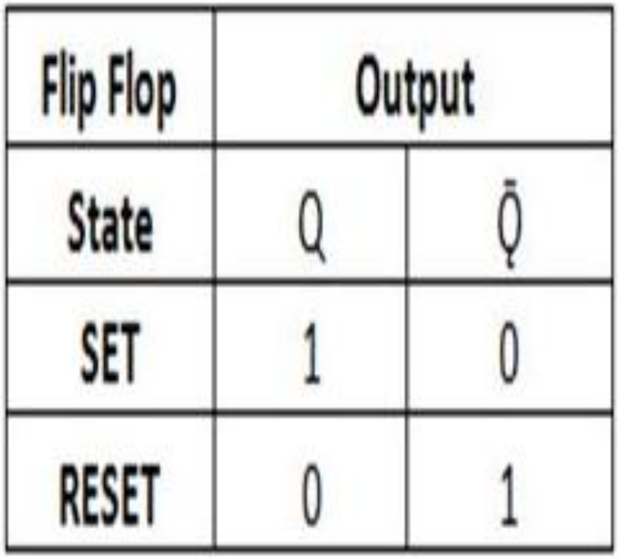


S	R	Q $t+1$
0	0	Q t
0	1	0
1	0	1
1	1	-

SR FLIPFLOP

- The S and R in SR flip – flop means 'SET' and 'RESET' respectively. Hence it is also called Set – Reset flip – flop. The symbolic representation of the SR Flip Flop is shown below.

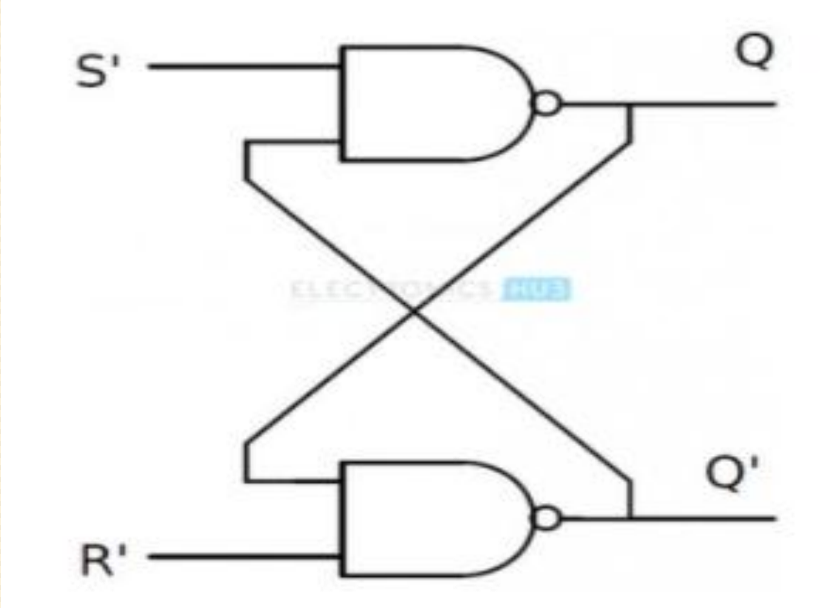


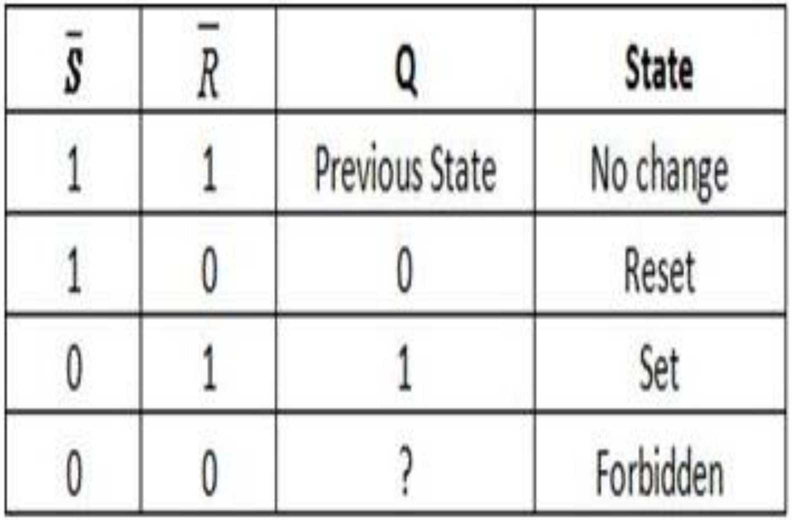


Flip Flop	Output	
State	Q	$\bar{Q}$
SET	1	0
RESET	0	1

UNCLOCKED S-R FLIP-FLOP USING NAND GATE

- SR flip flop can be designed by cross coupling of two NAND gates. It is an active low input SR flip – flop. The circuit of SR flip – flop using NAND gates is shown in below figure

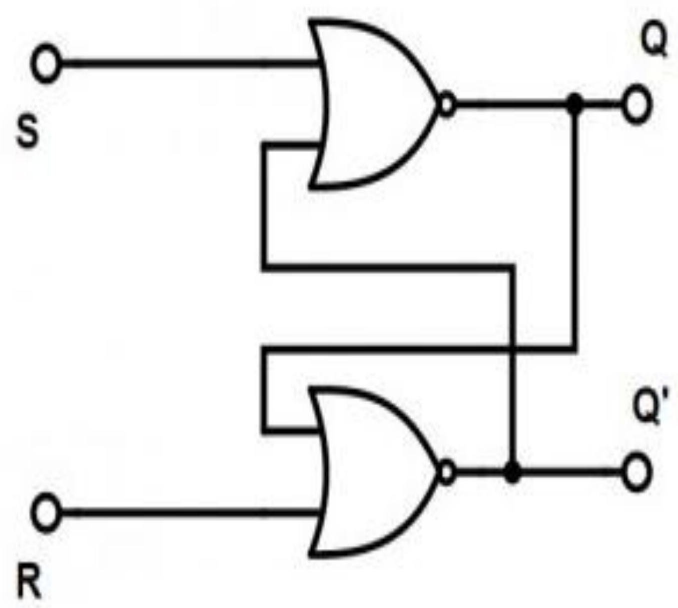


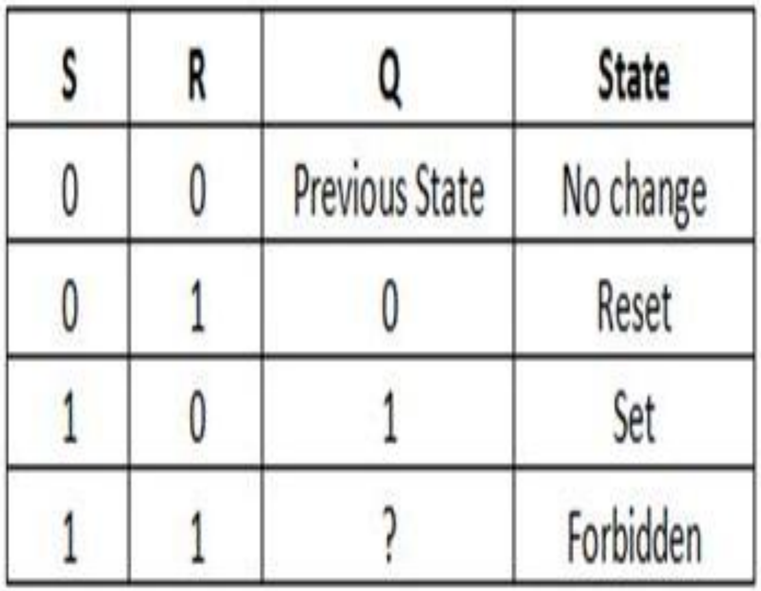


$\bar{S}$	$\bar{R}$	Q	State
1	1	Previous State	No change
1	0	0	Reset
0	1	1	Set
0	0	?	Forbidden

UNCLOCKED S R FLIP-FLOP USING NOR GATE

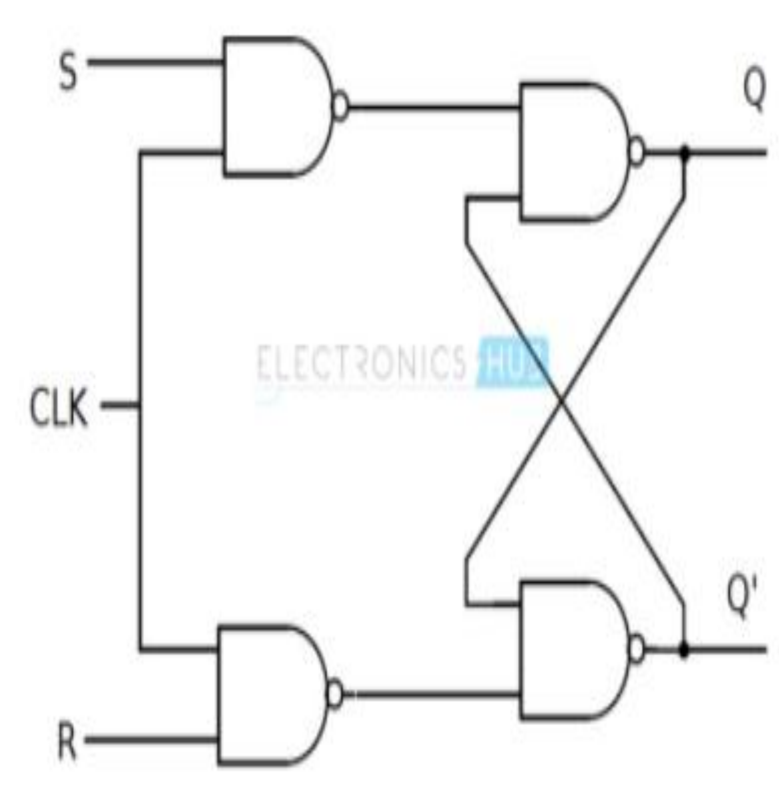
- SR flip flop can also be designed by cross coupling of two NOR gates. It is an active high input SR flip – flop. The circuit of SR flip – flop using NOR gates is shown in below figure.

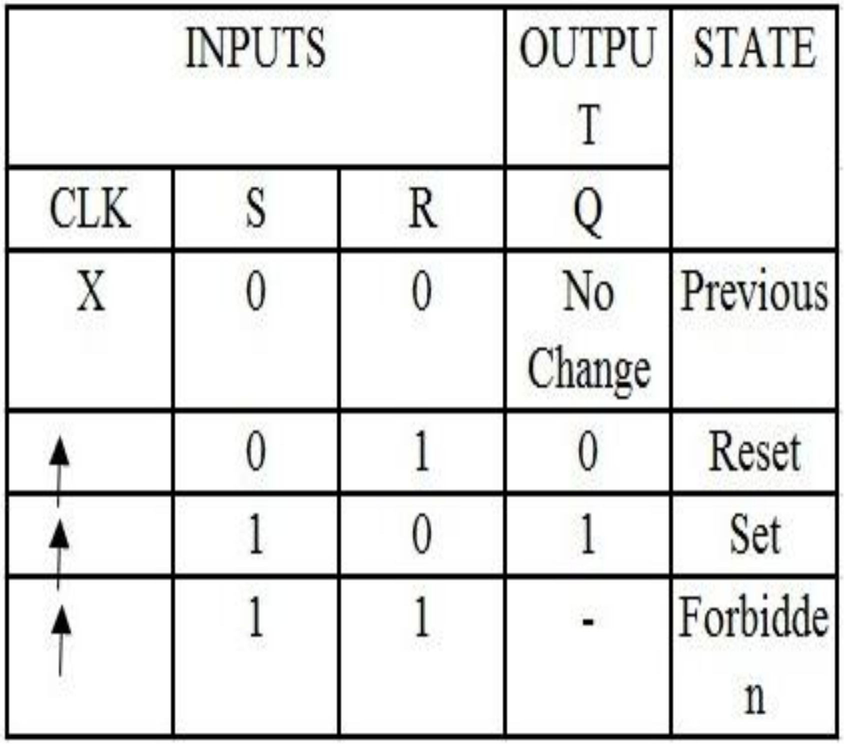




S	R	Q	State
0	0	Previous State	No change
0	1	0	Reset
1	0	1	Set
1	1	?	Forbidden

CLOCKED SR FLIP – FLOPS

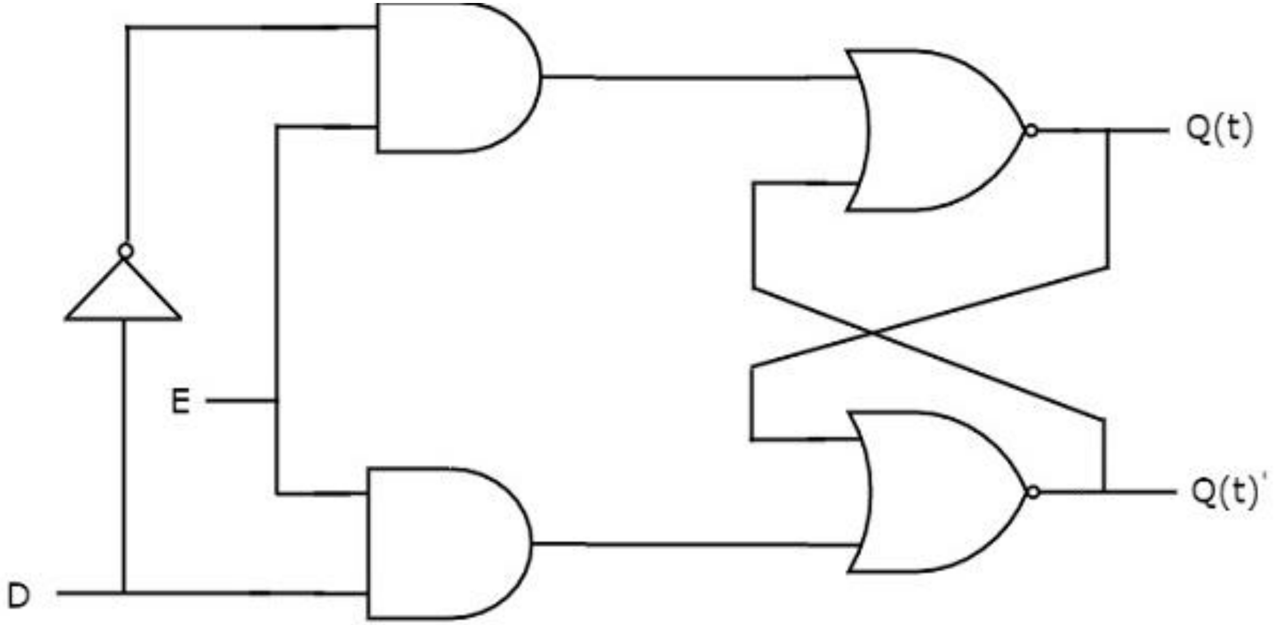




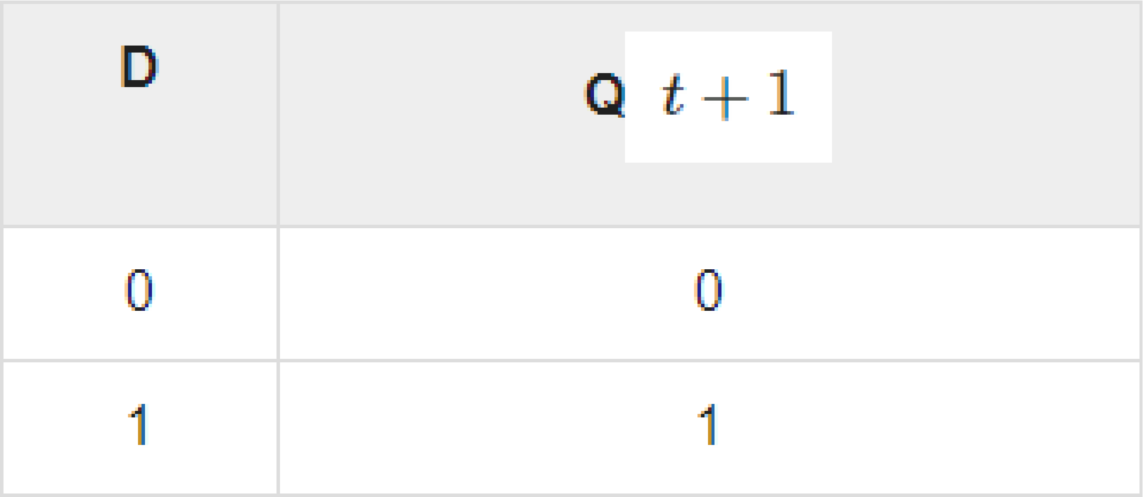
INPUTS			OUTPUT	STATE
CLK	S	R	Q	
X	0	0	No Change	Previous
↑	0	1	0	Reset
↑	1	0	1	Set
↑	1	1	-	Forbidden

D Latch

- There is one drawback of SR Latch. That is the next state value can’t be predicted when both the inputs S & R are one. So, we can overcome this difficulty by D Latch. It is also called as Data Latch. The **circuit diagram** of D Latch is shown in the following figure.



TRUTH TABLE-D LATCH

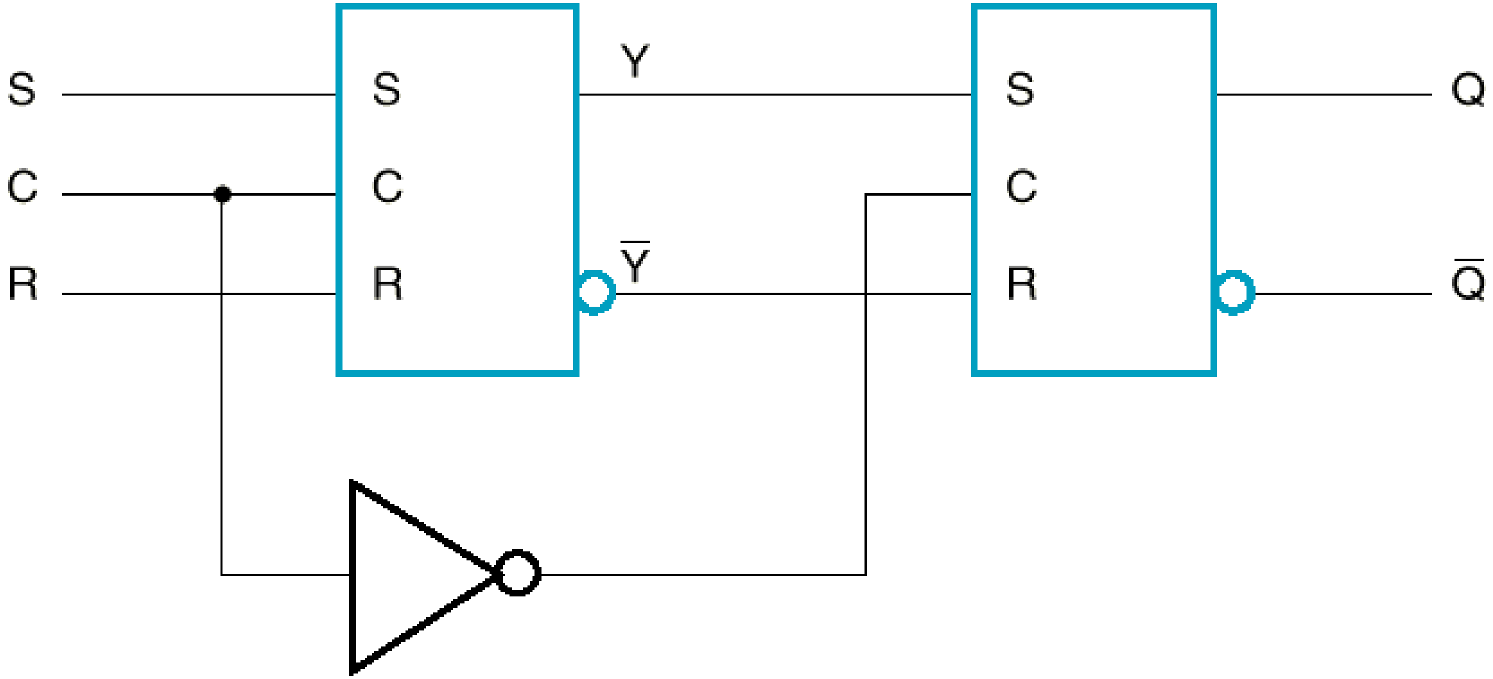


D	Q $t+1$
0	0
1	1

Flip-Flops

- Latches are “transparent” (= any change on the inputs is seen at the outputs immediately when C=1).
- This causes synchronization problems.
- Solution: use latches to create flip-flops that can respond (update) only on specific times (instead of any time).
- Types: RS flip-flop and D flip-flop

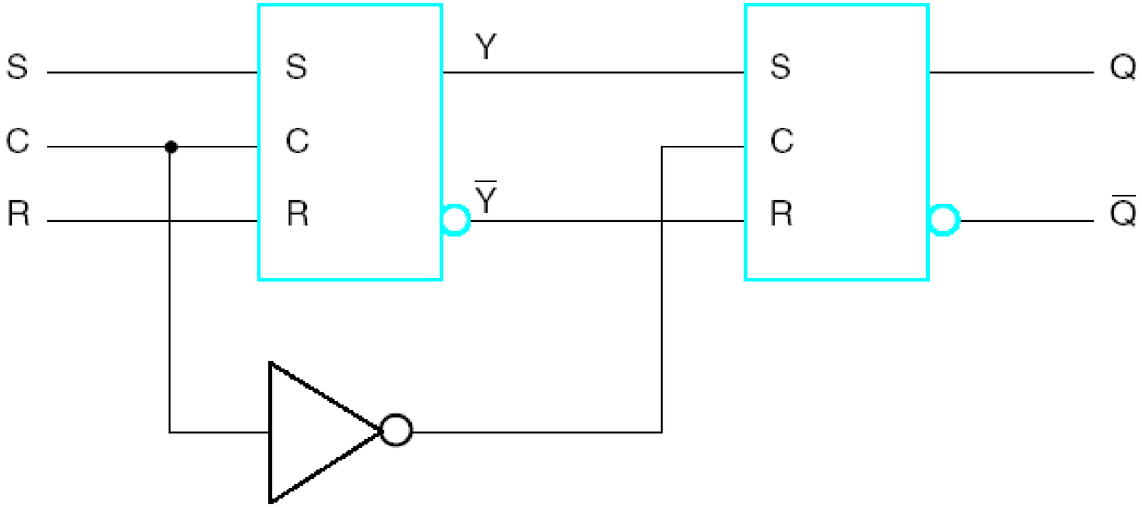
Master-Slave FF configuration using SR latches



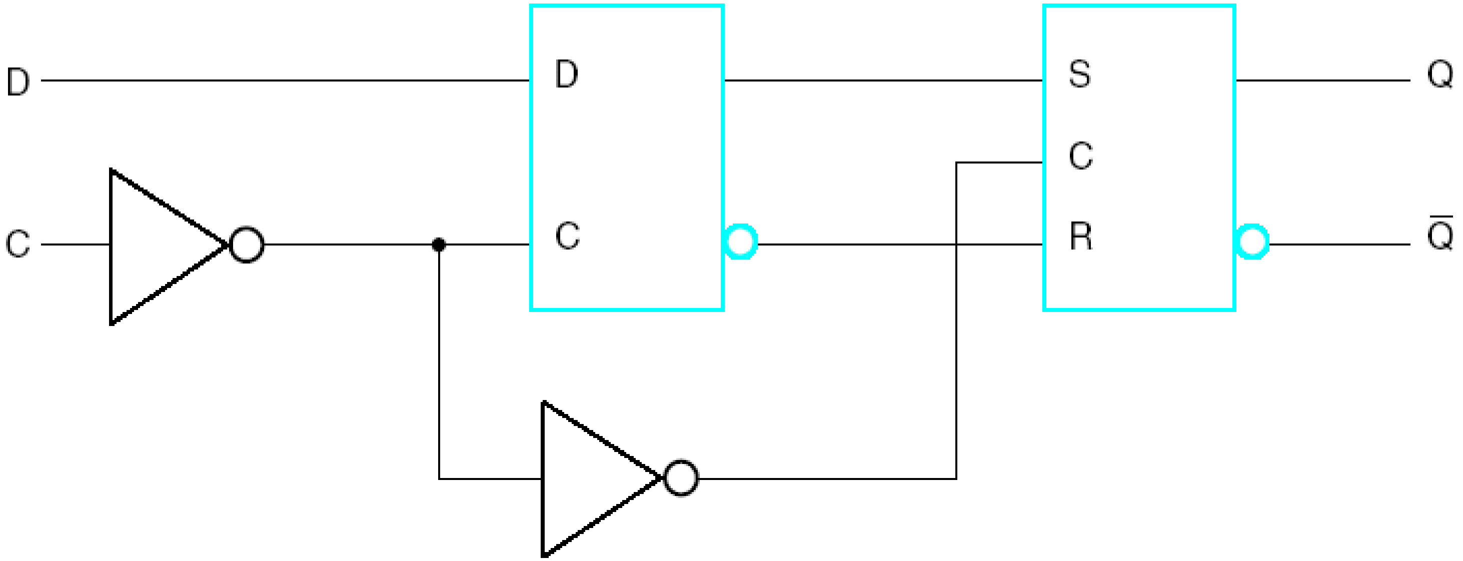
Master-Slave FF configuration using SR latches (cont.)

S	R	CLK	Q	Q'	
0	0	1	Q_0	Q_0'	Store
0	1	1	0	1	Reset
1	0	1	1	0	Set
1	1	1	1	1	Disallowed
X	X	0	Q_0	Q_0'	Store

- When C=1, master is enabled and stores *new* data, slave stores *old* data.
- When C=0, master's state passes to enabled slave, master not sensitive to new data (disabled).



D Flip-Flop



Characteristic Tables

- Defines the logical properties of a flip-flop (such as a truth table does for a logic gate).
- Q(t) – present state at time t
- Q(t+1) – next state at time t+1

Characteristic Tables (cont.)

SR Flip-Flop

S	R	Q(t+1)	Operation
0	0	Q(t)	No change/Hold
0	1	0	Reset
1	0	1	Set
1	1	?	Undefined/Invalid

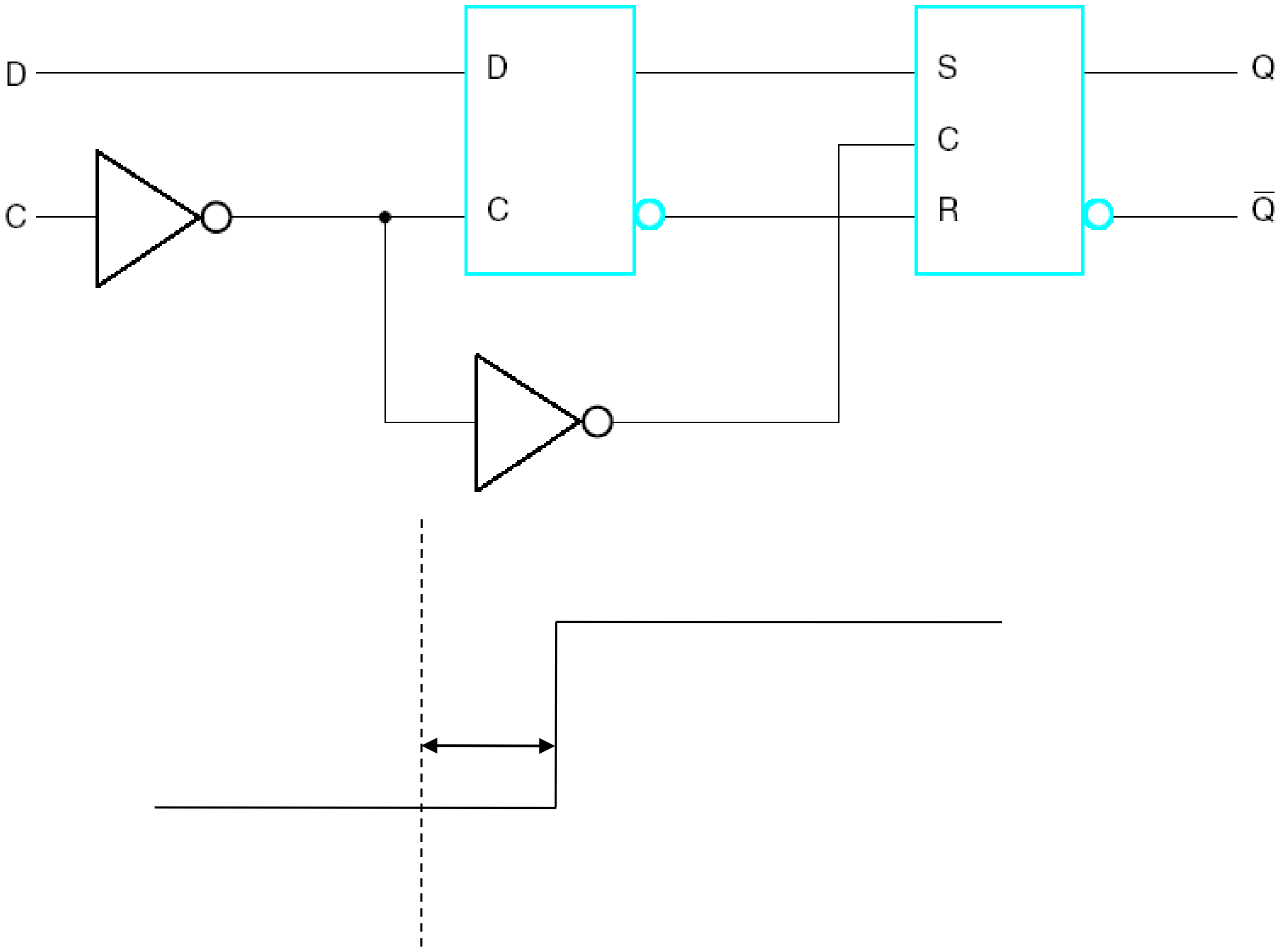
Characteristic Tables (cont.)

D Flip-Flop

D	Q(t+1)	Operation
0	0	Set
1	1	Reset

Characteristic Equation: $Q(t+1) = D(t)$

D Flip-Flop Timing Parameters



Setup time

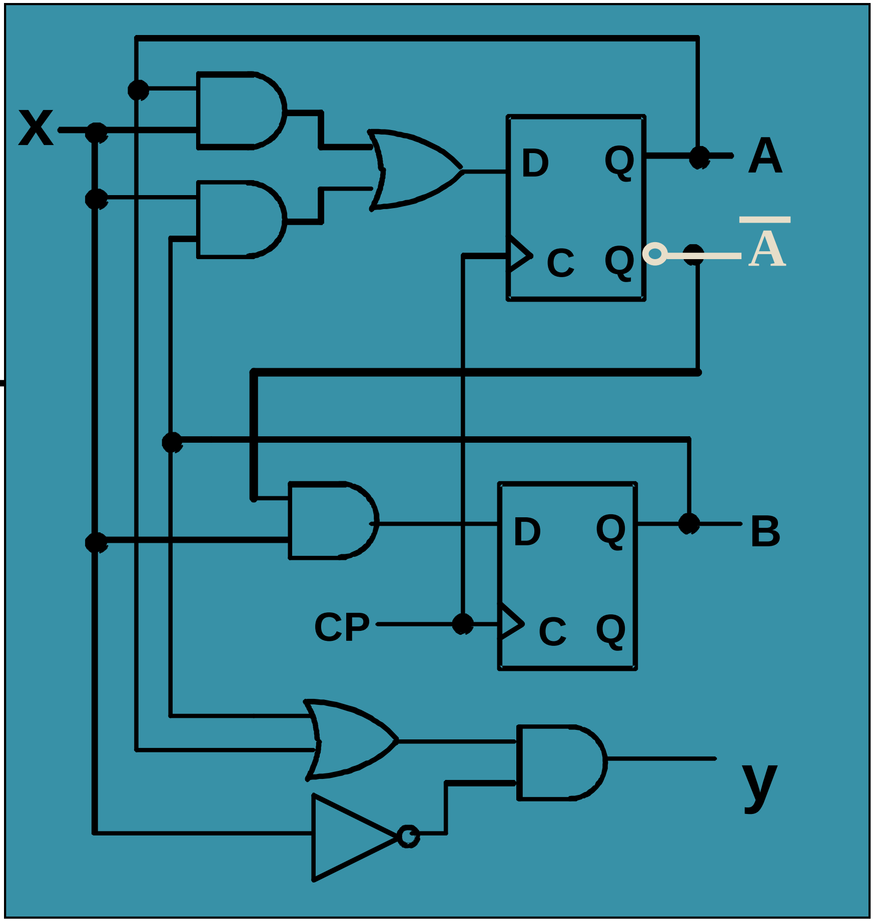
Sequential Circuit Analysis

- ***Analysis***: Consists of obtaining a suitable description that demonstrates the time sequence of inputs, outputs, and states.
- Logic diagram: Boolean gates, flip-flops (of any kind), and appropriate interconnections.
- The logic diagram is derived from any of the following:
  - Boolean Equations (FF-Inputs, Outputs)
  - State Table
  - State Diagram

Example

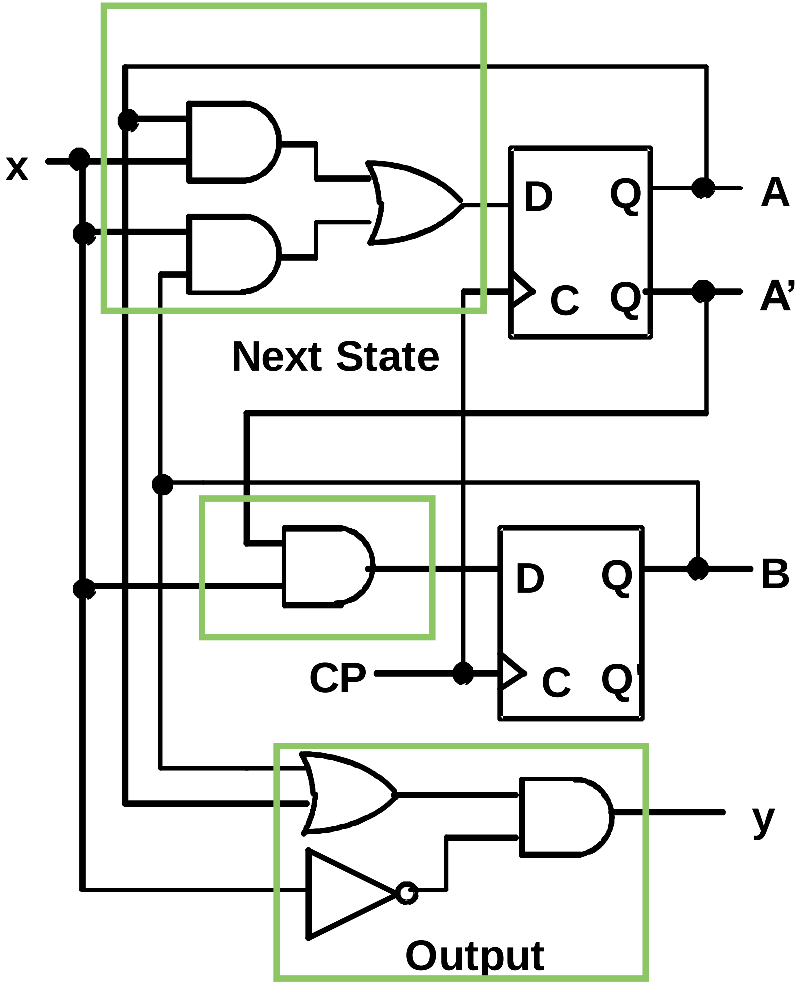
- Input: x(t)
- Output: y(t)
- State: (A(t), B(t))
- What is the Output

- What is the Next State Function?



Example (continued)

- Boolean equations for the functions:
  - A(t+1) = A(t)x(t) + B(t)x(t)
  - B(t+1) = A'(t)x(t)
  - y(t) = x'(t)(B(t) + A(t))

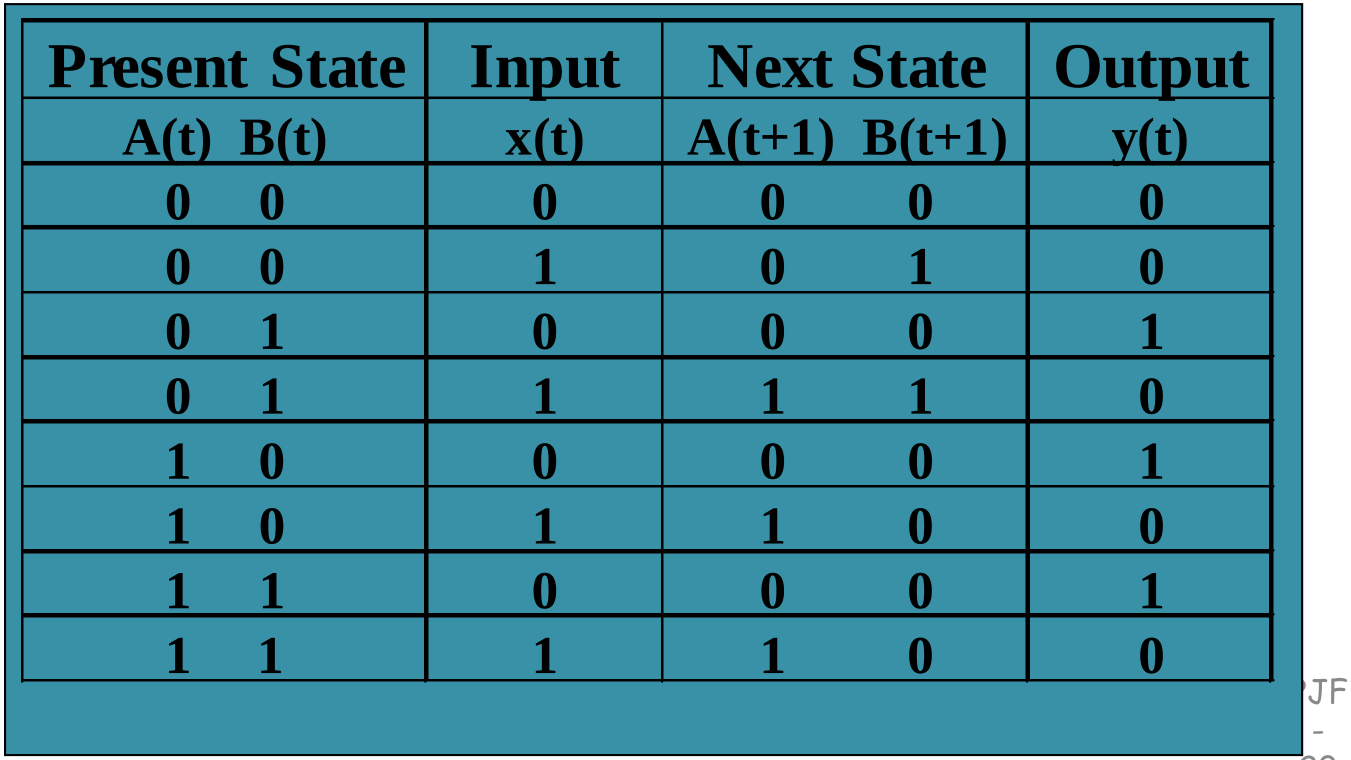


State Table Characteristics

- *State table* – a multiple variable table with the following four sections:
  - *Present State* – the values of the state variables for each allowed state.
  - *Input* – the input combinations allowed.
  - *Next-state* – the value of the state at time (t+1) based on the present state and the input.
  - *Output* – the value of the output as a function of the present state and (sometimes) the input.
- From the viewpoint of a truth table:
  - the inputs are Input, Present State
  - and the outputs are Output, Next State

Example: State Table

- The state table can be filled in using the next state and output equations:
  - $A(t+1) = A(t)x(t) + B(t)x(t)$
  - $B(t+1) = \overline{A}(t)x(t);$
  - $y(t) = \overline{x}(t)(B(t) + A(t))$



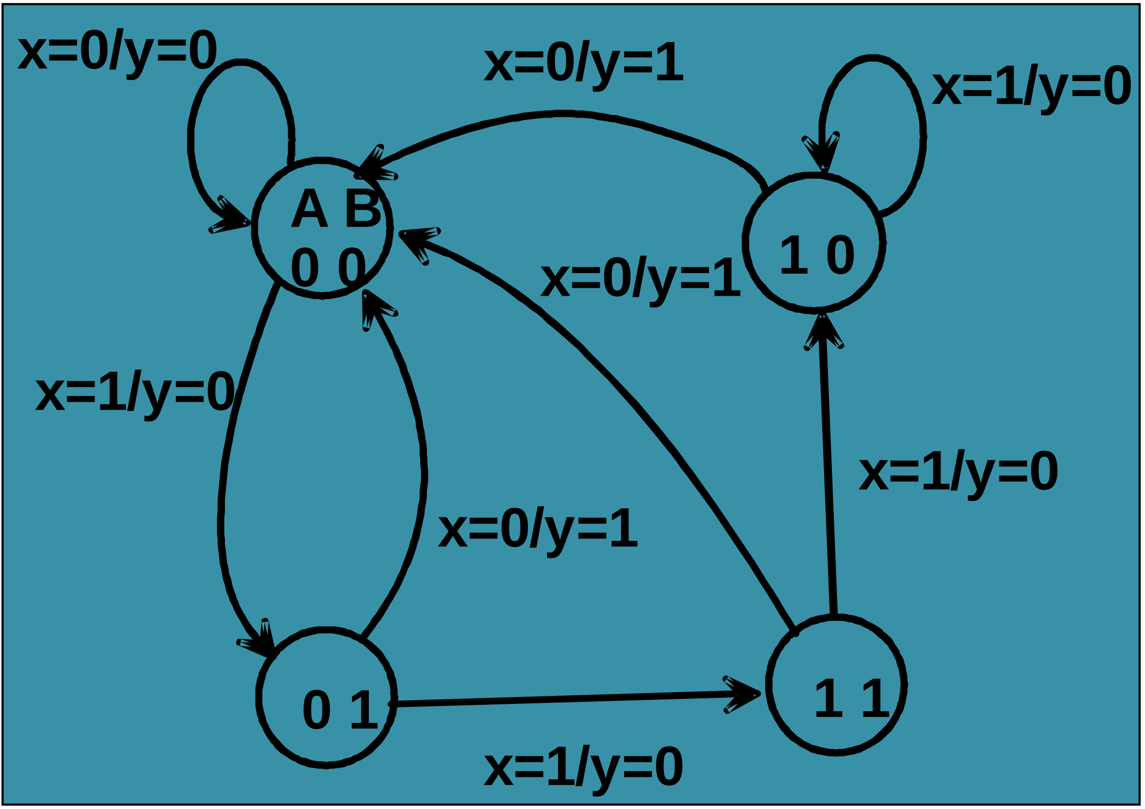
Present State		Input	Next State		Output
A(t)	B(t)	x(t)	A(t+1)	B(t+1)	y(t)
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	1	0	0

State Diagrams

- The sequential circuit function can be represented in graphical form as a state diagram with the following components:
  - A circle with the state name in it for each state
  - A directed arc from the Present State to the Next State for each state transition
  - A label on each directed arc with the Input values which causes the state transition, and
  - A label:
    - On each circle with the output value produced, or
    - On each directed arc with the output value produced.

Example: State Diagram

- Diagram gets confusing for large circuits
- For small circuits, usually easier to understand than the state table



UNIT-5 MEMORY AND PROGRAMMABLE LOGIC

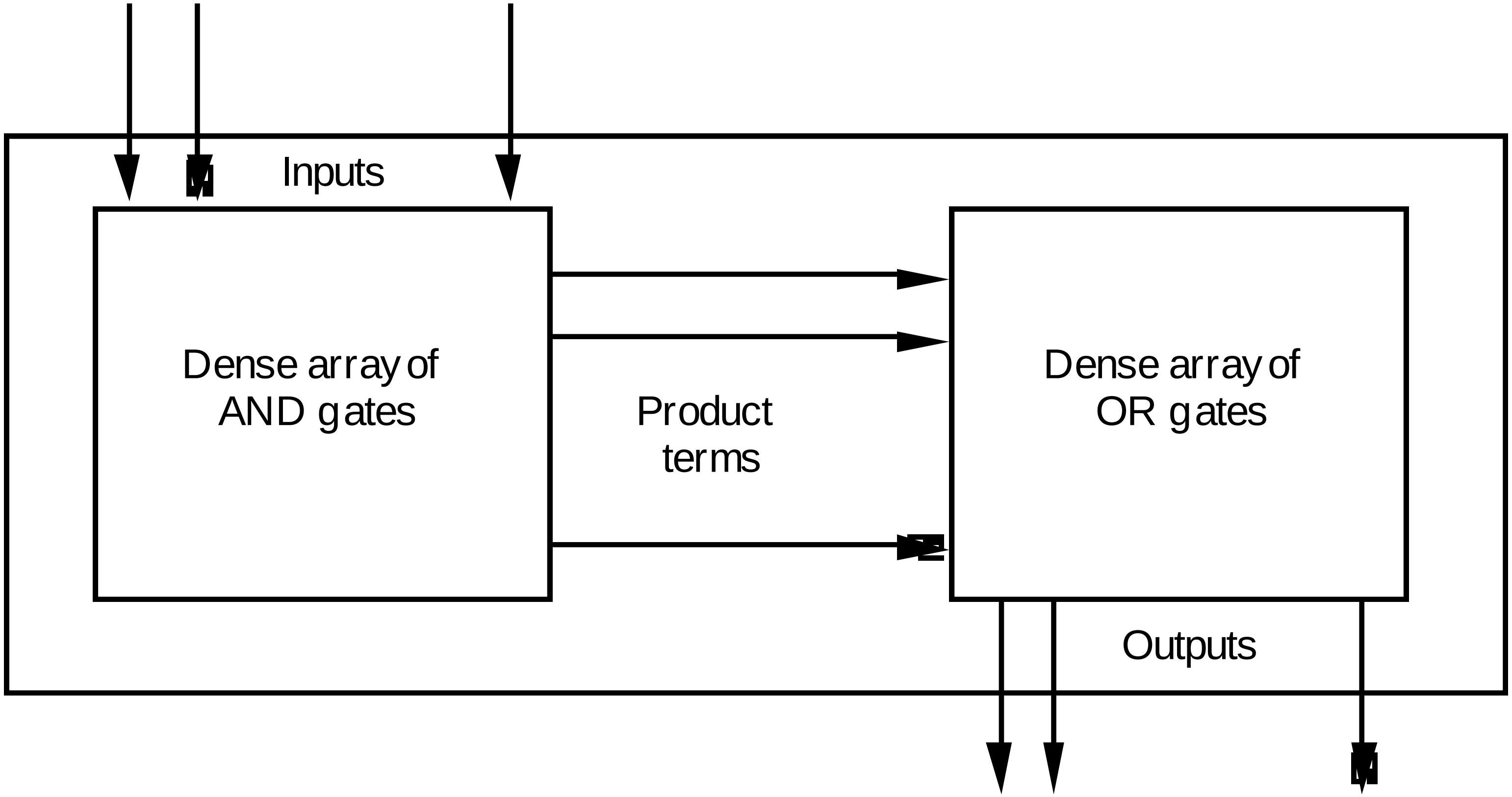
PLAs

Programmable Logic Array

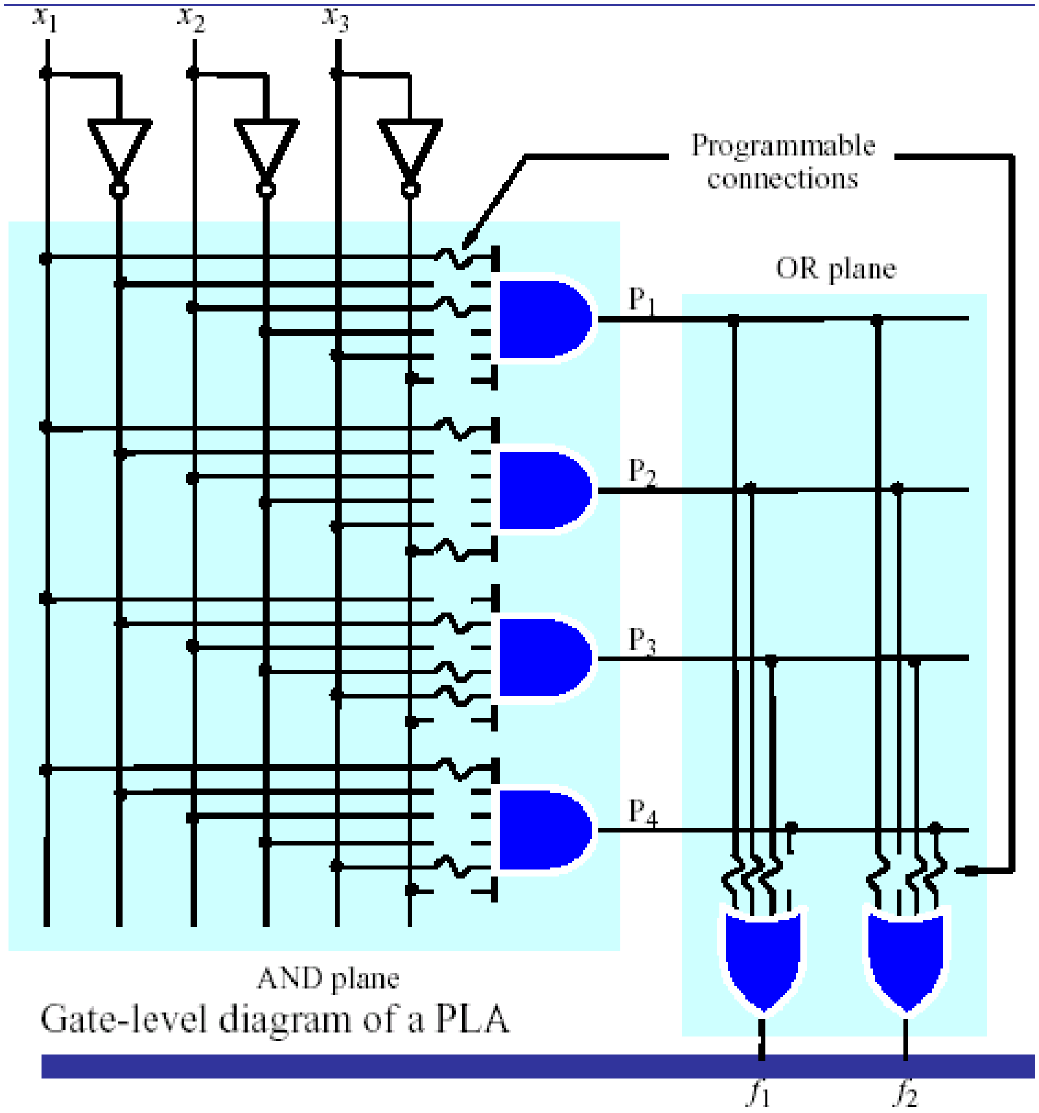
- Pre-fabricated building block of many AND/OR gates (or NOR, NAND) "Personalized" by making/ breaking connections among the gates.

- General purpose logic building blocks.

PLA

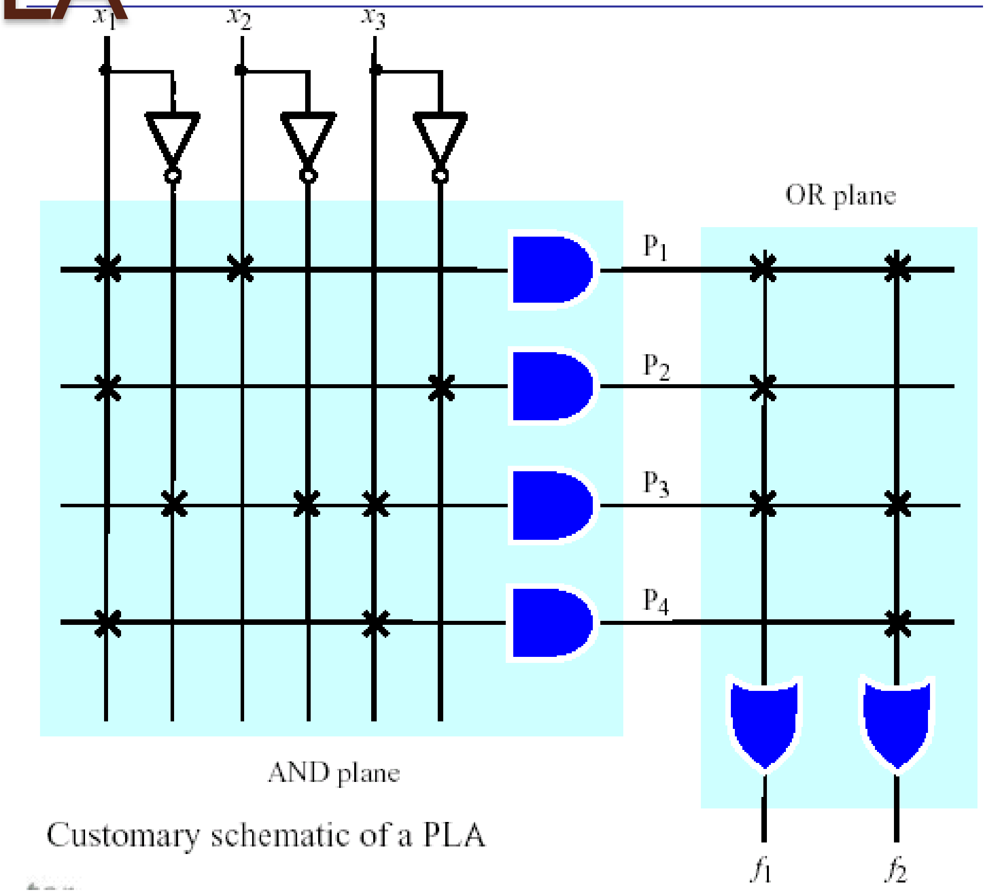

Inputs
Dense array of AND gates
Product terms
Dense array of OR gates
Outputs

PLA



Gate-level diagram of a PLA

PLA



Customary schematic of a PLA

- **A 3×2 PLA with 4 product terms.**

Design for PLA: Example

- Implement the following functions using PLA

F0 = A + B' C'
F1 = A C' + A B
F2 = B' C' + A B
F3 = B' C + A

Input Side:

1 = asserted in term
0 = negated in term
- = does not participate

Personality Matrix

Product term	Inputs			Outputs			
	A	B	C	F_0	F_1	F_2	F_3
A B	1	1	-	0	1	1	0
$\overline{B}$ C	-	0	1	0	0	0	1
A $\overline{C}$	1	-	0	0	1	0	0
$\overline{B}$ $\overline{C}$	-	0	0	1	0	1	0
A	1	-	-	1	0	0	1

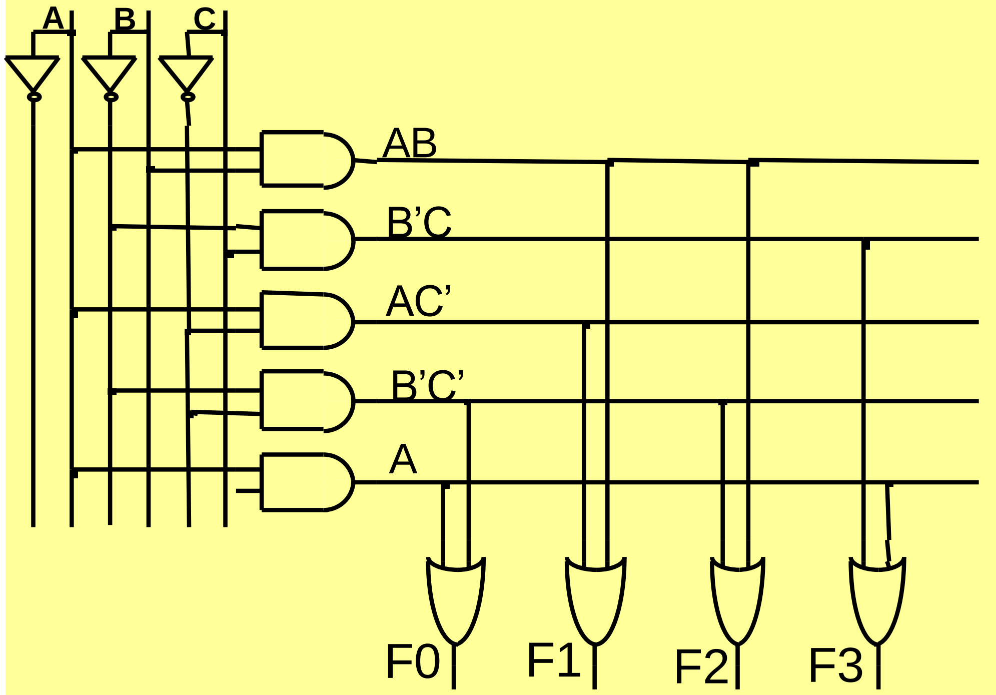
Reuse of terms

Output Side:

1 = term connected to output
0 = no connection to output

Example: Continued

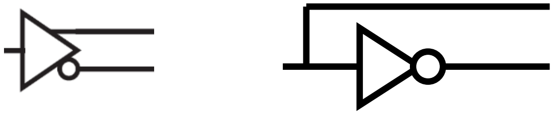
F0 = A + B' C'
F1 = A C' + A B
F2 = B' C' + A B
F3 = B' C + A



Personality Matrix

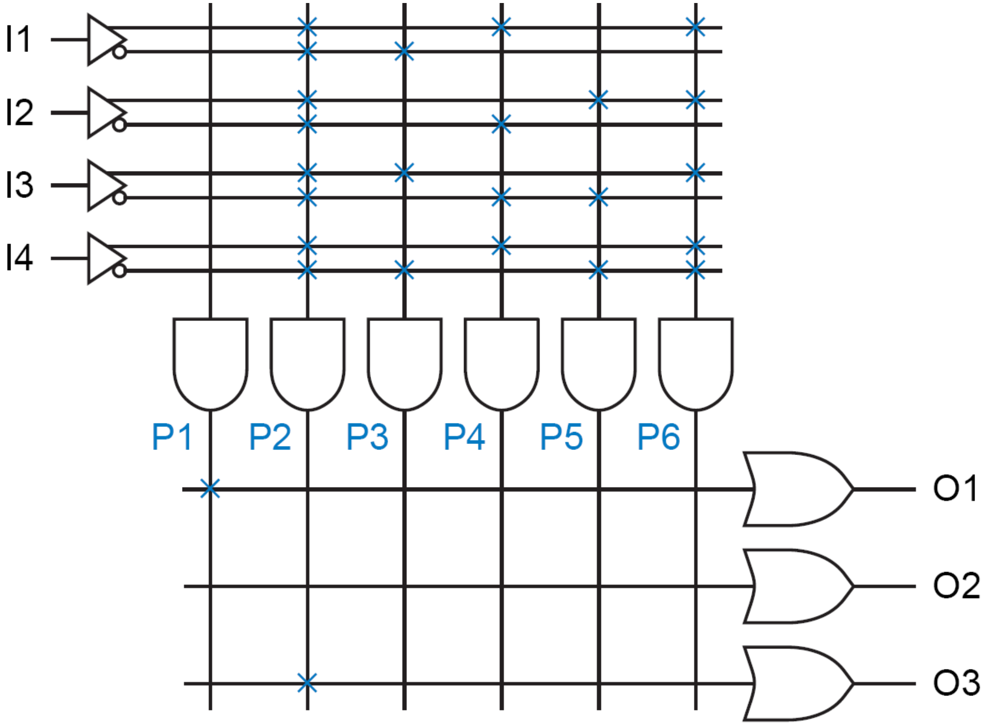
Product term	Inputs			Outputs			
	A	B	C	F_0	F_1	F_2	F_3
A B	1	1	-	0	1	1	0
$\overline{B}$ C	-	0	1	0	0	0	1
A $\overline{C}$	1	-	0	0	1	0	0
$\overline{B}$ $\overline{C}$	-	0	0	1	0	1	0
A	1	-	-	1	0	0	1

Reuse of terms



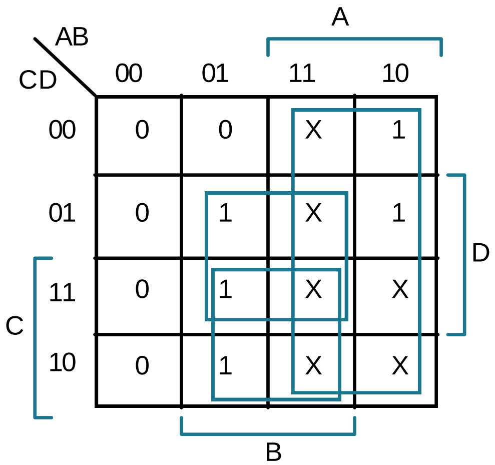
Constants

- Sometimes a PLA output must be programmed to be a constant 1 or a constant 0.
  - P1 is always 1 because its product line is connected to no inputs and is therefore always pulled HIGH;
  - this constant-1 term drives the O1 output.
- No product term drives the O2 output, which is therefore always 0.
- Another method of obtaining a constant-0 output is shown for O3.



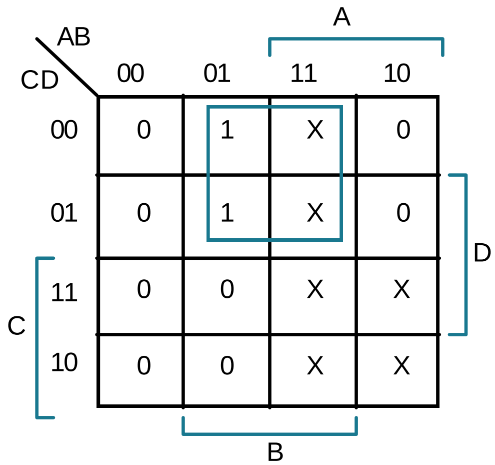
BCD to Gray Code Converter

A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	1	1	1	0
0	1	1	0	1	0	1	0
0	1	1	1	1	0	1	1
1	0	0	0	1	0	0	1
1	0	0	1	1	0	0	0
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X



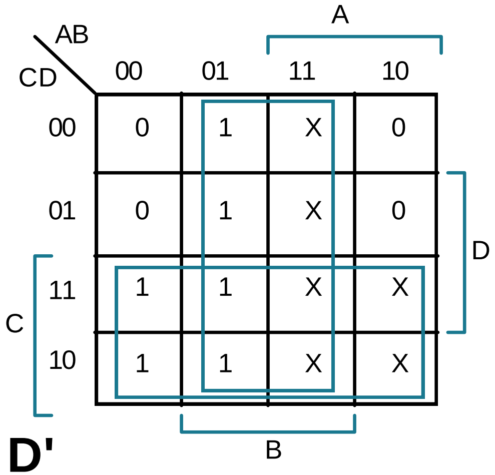
CD \ AB	00	01	11	10
00	0	0	X	1
01	0	1	X	1
11	0	1	X	X
10	0	1	X	X

K-map for W



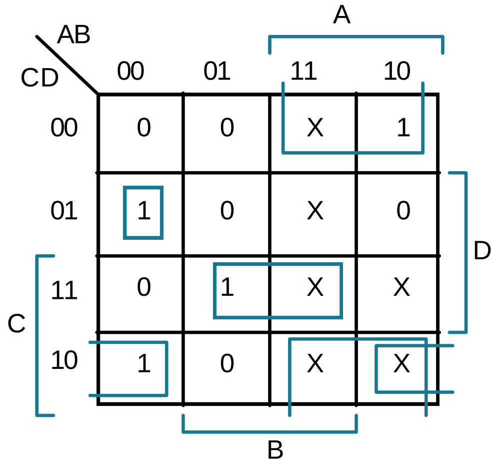
CD \ AB	00	01	11	10
00	0	1	X	0
01	0	1	X	0
11	0	0	X	X
10	0	0	X	X

K-map for X



CD \ AB	00	01	11	10
00	0	1	X	0
01	0	1	X	0
11	1	1	X	X
10	1	1	X	X

K-map for Y



CD \ AB	00	01	11	10
00	0	0	X	1
01	1	0	X	0
11	0	1	X	X
10	1	0	X	X

K-map for Z

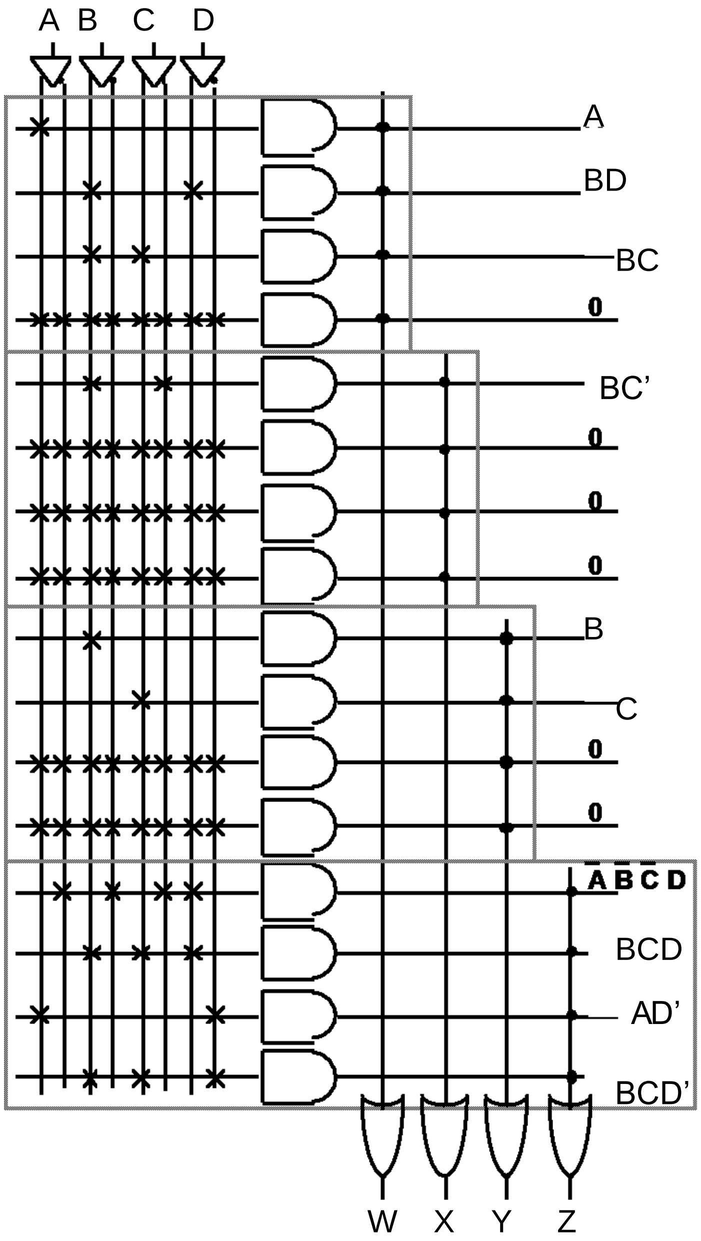
Minimized Functions:

$W = A + B\,D + B\,C$
$X = B\,C'$
$Y = B + C$
$Z = A'B'C'D + B\,C\,D + A\,D' + B'\,C\,D'$

4 product terms per each OR gate

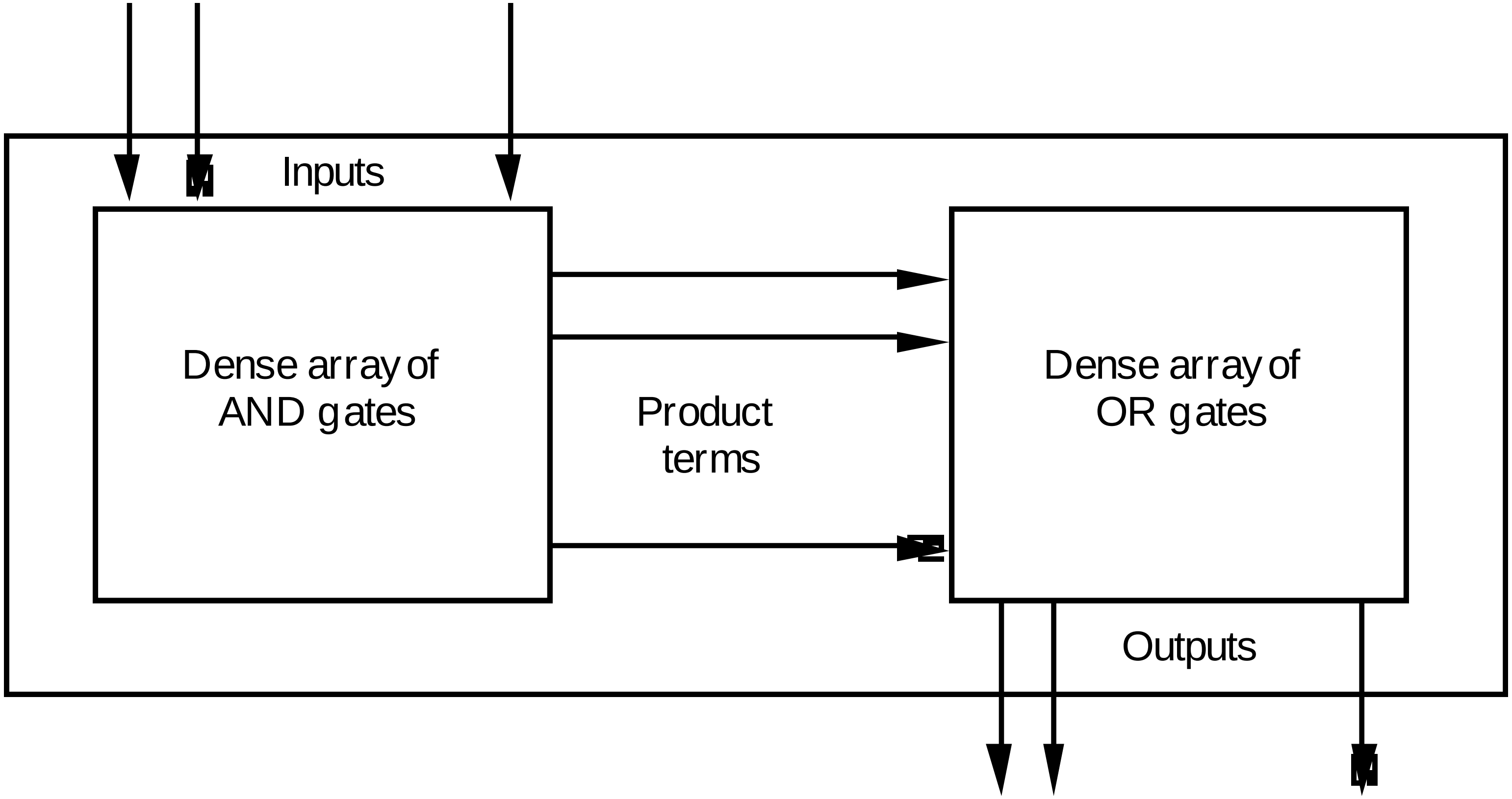
Product terms cannot be shared !

PLA achieves higher flexibility at the cost of lower speed!



PALs

- Programmable Array Logic
  - a *fixed* OR array.



PAL

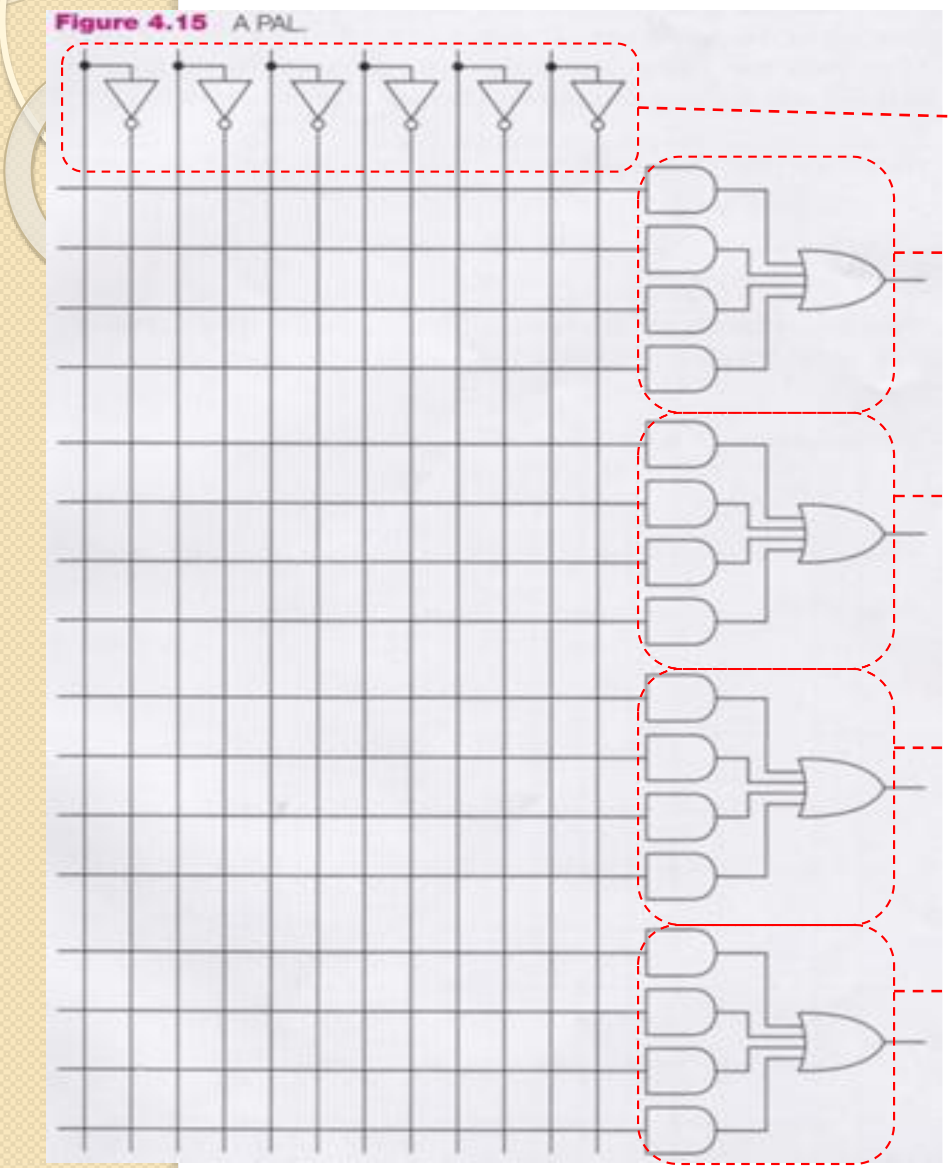


Figure 4.15 A PAL

inputs

1st output section

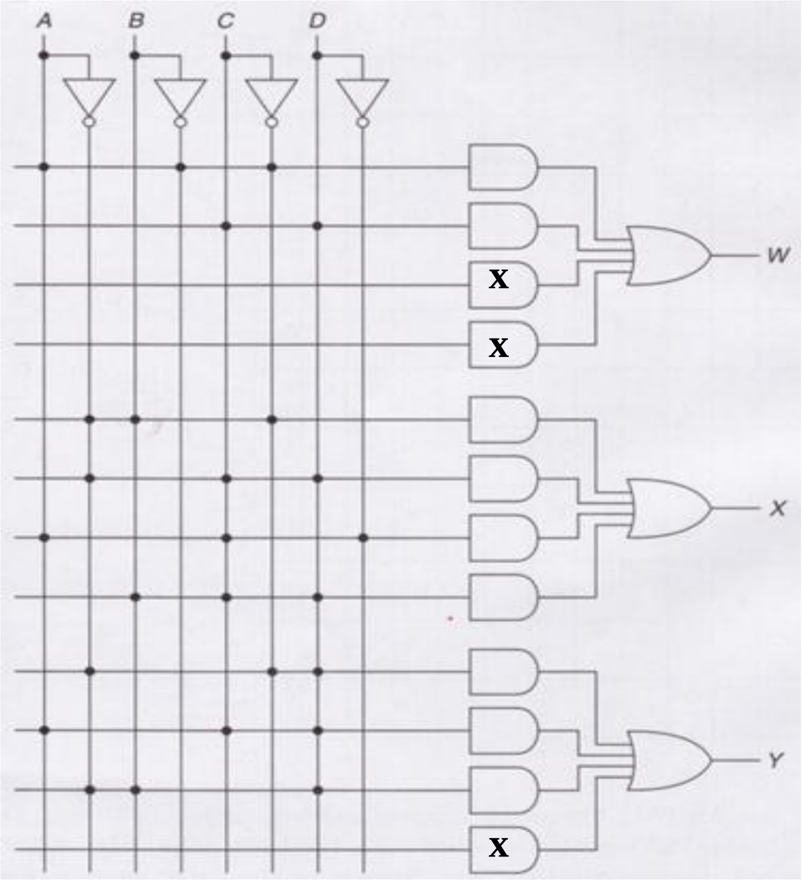
2nd output section

3rd output section

4th output section

Only functions with at most four products can be implemented

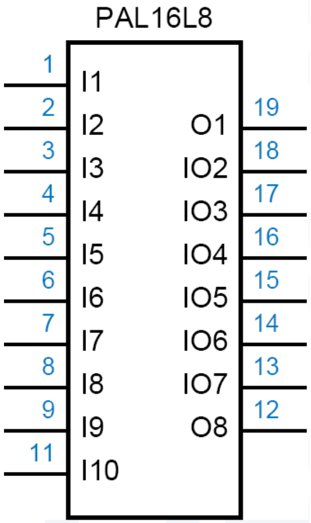
PAL

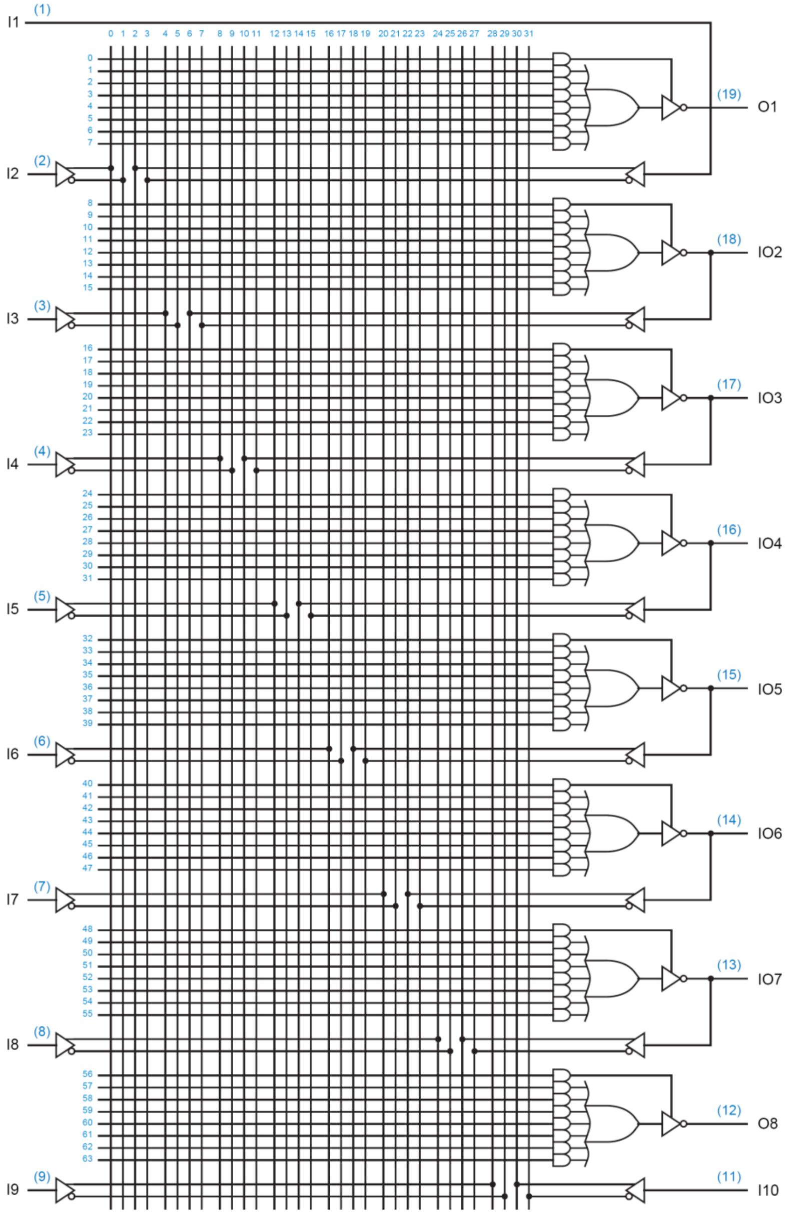


$W = AB'C' + CD$

$X = A'BC' + A'CD + ACD' + BCD$

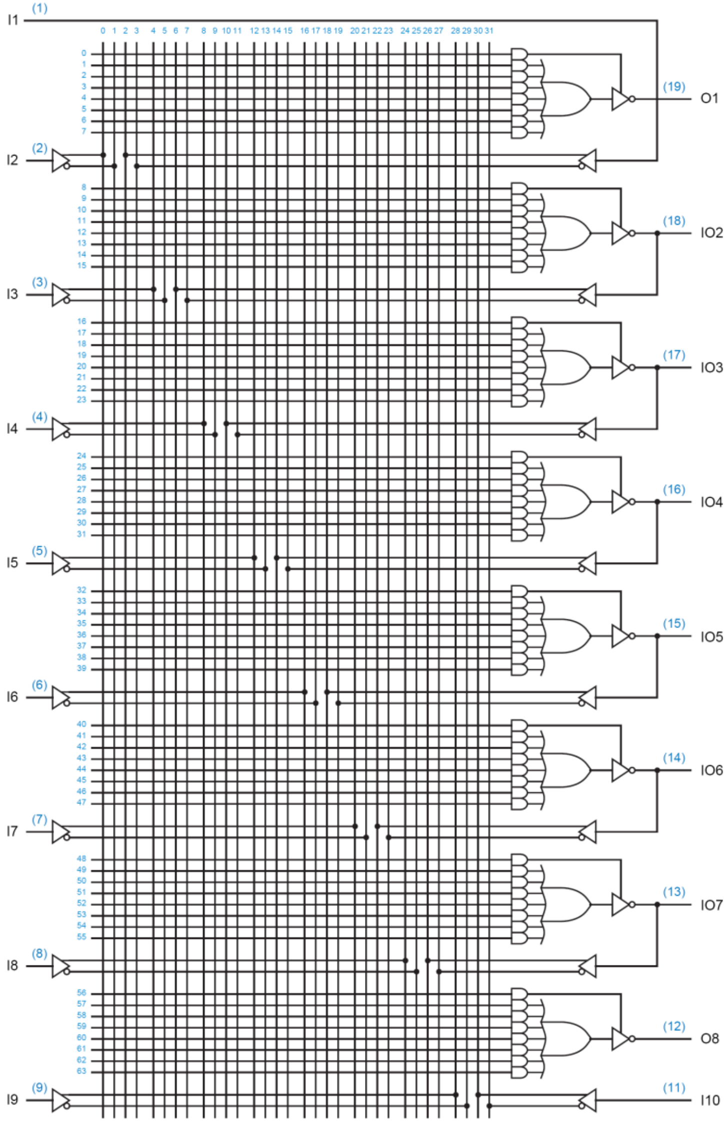
$Y = A'C'D + ACD + A'BD$


PAL16L8
1
I1
2
I2
3
I3
4
I4
5
I5
6
I6
7
I7
8
I8
9
I9
11
I10
19
O1
18
IO2
17
IO3
16
IO4
15
IO5
14
IO6
13
IO7
12
O8


I1 (1)
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
I2 (2)
I3 (3)
I4 (4)
I5 (5)
I6 (6)
I7 (7)
I8 (8)
I9 (9)
(19) O1
(18) IO2
(17) IO3
(16) IO4
(15) IO5
(14) IO6
(13) IO7
(12) O8
(11) I10

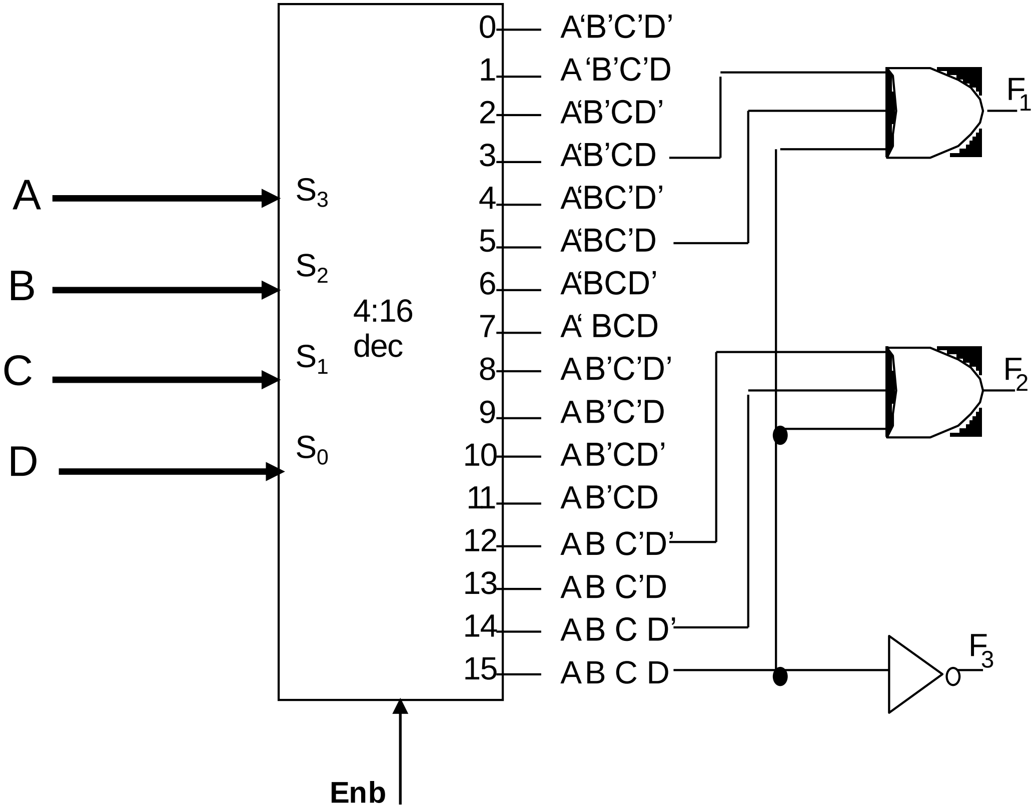
Helper Terms

- If an I/O pin’s output-control gate produces a constant 1, → the output is always enabled, but the pin may still be used as an input too.
- → outputs can be used to generate first-pass “helper terms” for logic functions that cannot be performed in a single pass with the limited number of AND terms available for a single output.



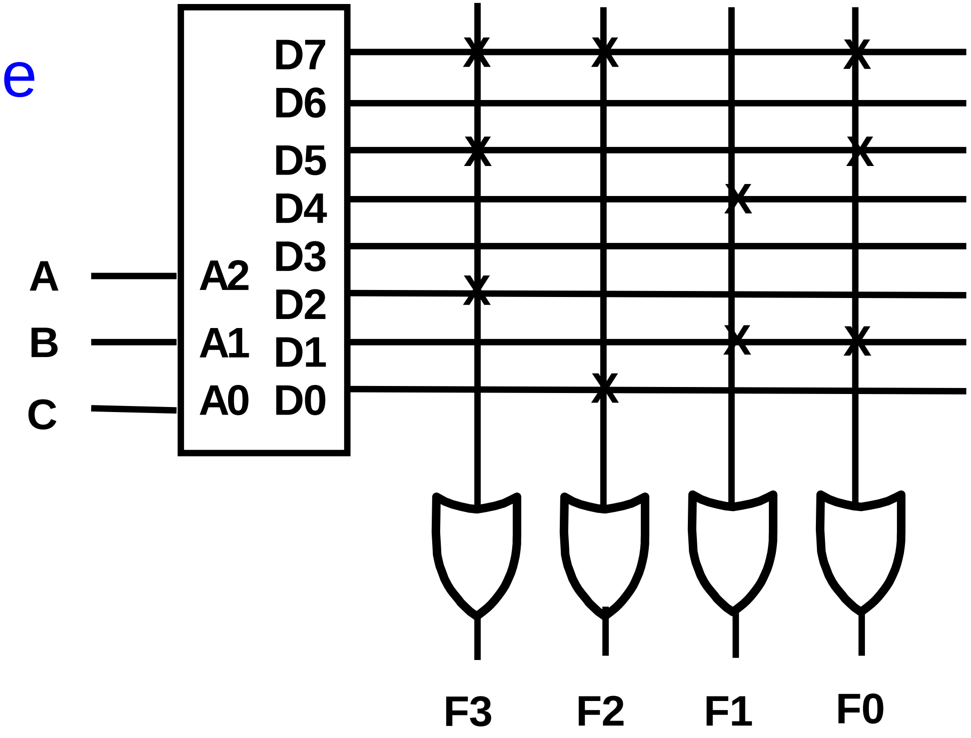
Read Only Memory (ROM)

- **Decoder**
  - Produces minterms
- **ORs**
  - Produce SOP's

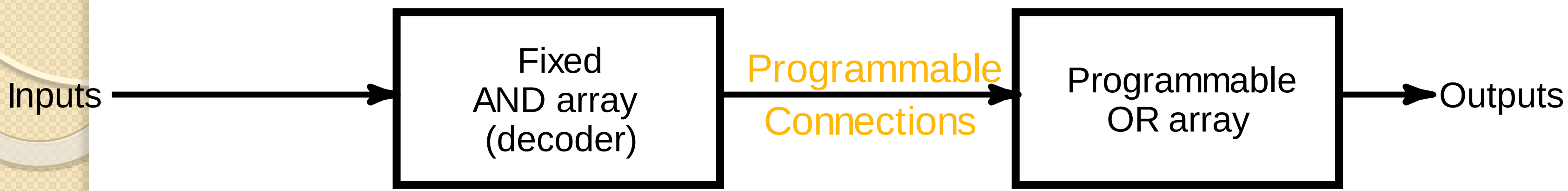


ROM

- ROM
  - A decoder
  - A set of programmable OR's



ROM vs. PLA/PAL



Inputs

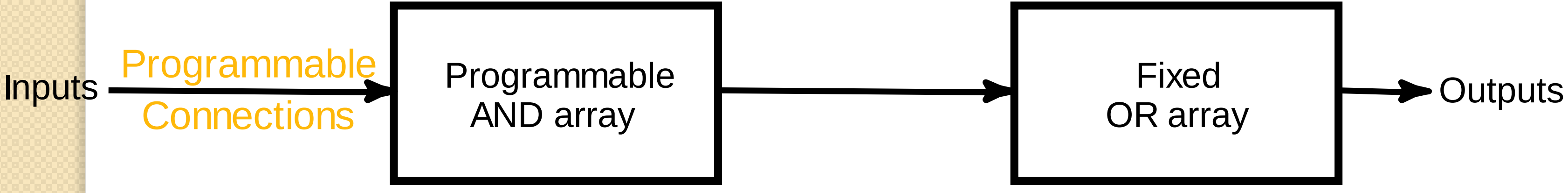
Fixed
AND array
(decoder)

Programmable
Connections

Programmable
OR array

Outputs

(a) Programmable read-only memory (PROM)



Inputs

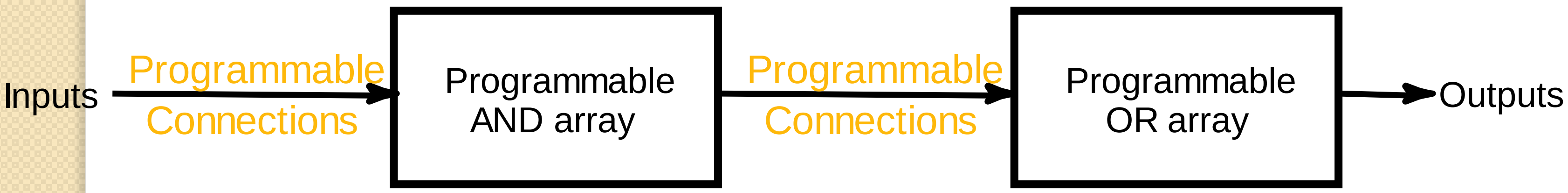
Programmable
Connections

Programmable
AND array

Fixed
OR array

Outputs

(b) Programmable array logic (PAL) device



Inputs

Programmable
Connections

Programmable
AND array

Programmable
Connections

Programmable
OR array

Outputs

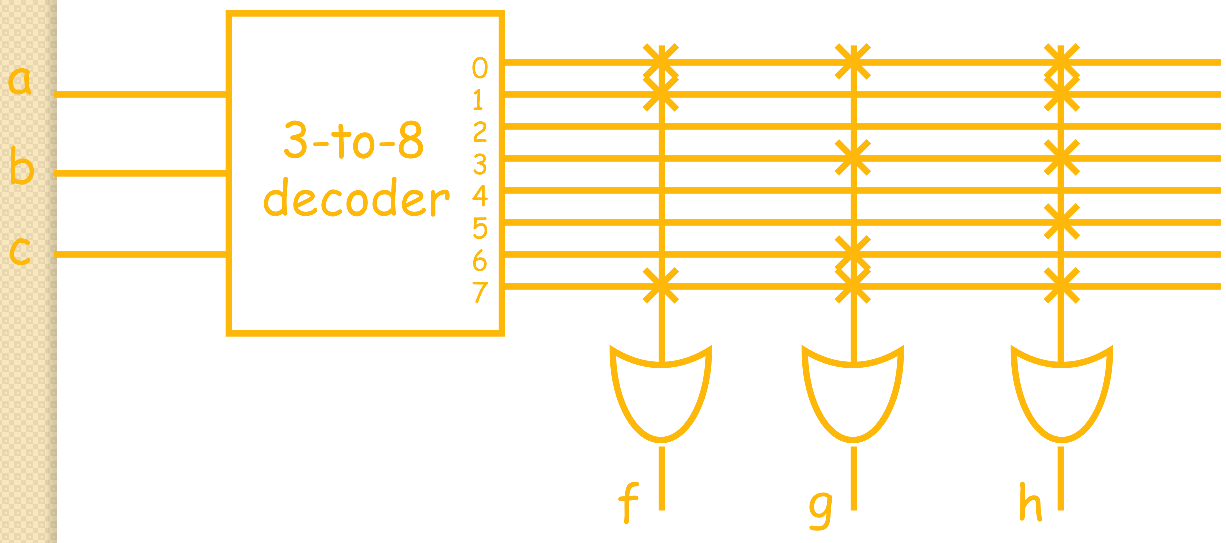
(c) Programmable logic array (PLA) device

Example

- Find a ROM-based circuit implementation for:
  - f(a,b,c) = a'b' + abc
  - g(a,b,c) = a'b'c' + ab + bc
  - h(a,b,c) = a'b' + c
- Solution:
  - Express f(), g(), and h() in ∑m() format (use truth tables)
  - Program the ROM based on the 3 ∑m()'s

Example

- There are 3 inputs and 3 outputs, thus we need a 8x3 ROM block.
  - f = ∑m(0, 1, 7)
  - g = ∑m(0, 3, 6, 7)
  - h = ∑m(0, 1, 3, 5, 7)



ROM as a Memory

- Read Only Memories (ROM) or Programmable Read Only Memories (PROM) have:
  - N input lines,
  - M output lines, and
  - 2^N decoded minterms.
- Can be viewed as a memory with the inputs as addresses of data (output values),
  - hence ROM or PROM names!

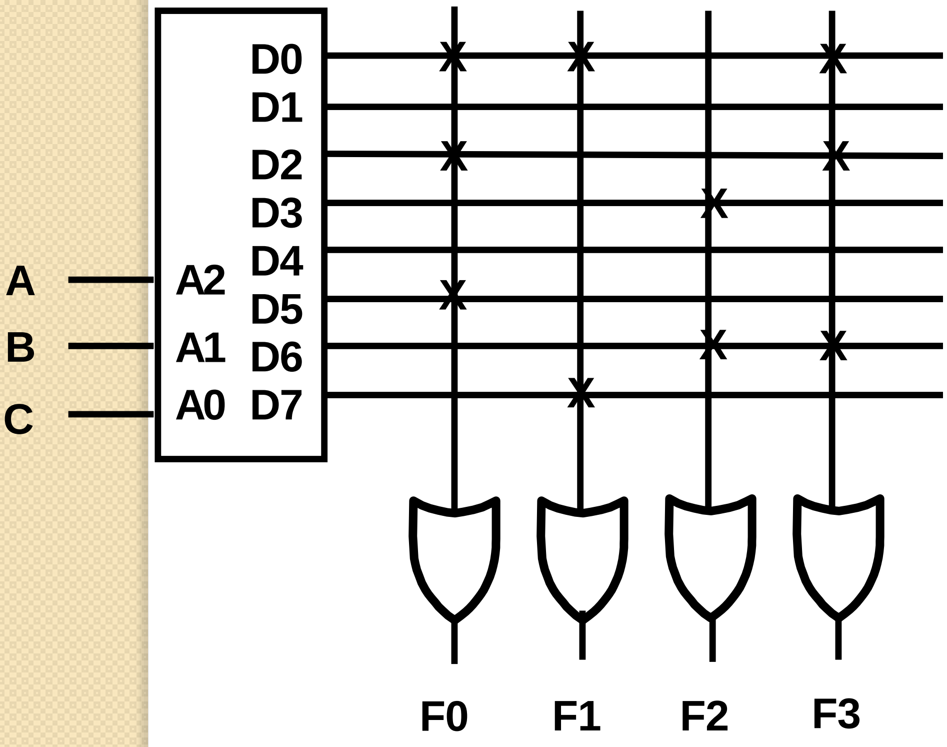
(Memories)

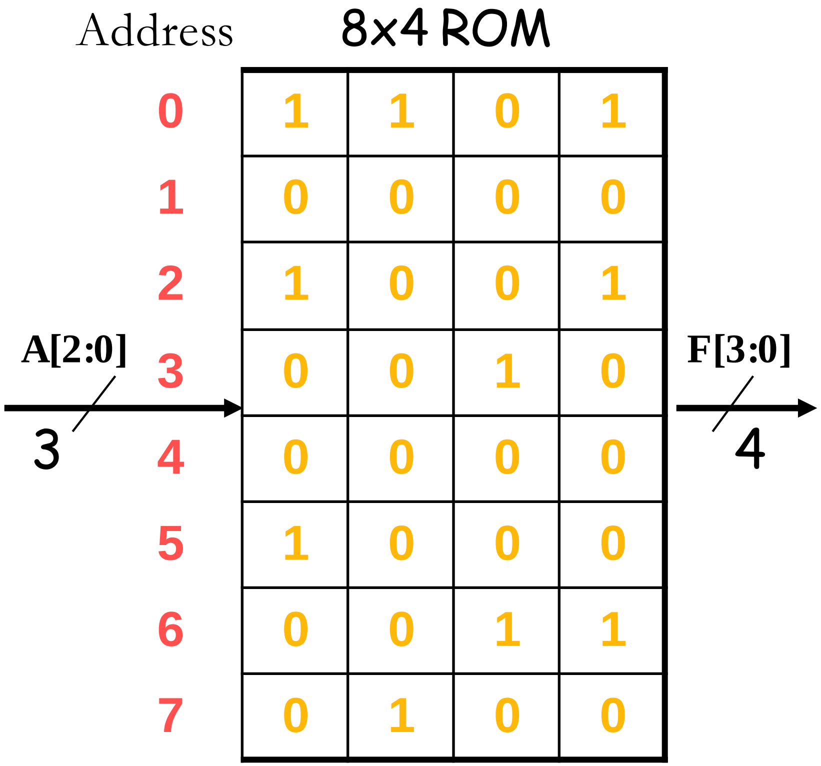
- Volatile:
  - Random Access Memory (RAM):
    - SRAM "static"
    - DRAM "dynamic"

- Non-Volatile:
  - Read Only Memory (ROM):
    - Mask ROM "mask programmable"
    - EPROM "electrically programmable"
    - EEPROM “electrically erasable electrically programmable"
    - FLASH memory - similar to EEPROM with programmer integrated on chip

ROM as Memory

- Read Example: For input (A_2,A_1,A_0) = 011, output is (F_0,F_1,F_2,F_3) = 0010.
- What are functions F_3, F_2 , F_1 and F_0 in terms of (A_2, A_1, A_0)?





Address	8x4 ROM			
0	1	1	0	1
1	0	0	0	0
2	1	0	0	1
3	0	0	1	0
4	0	0	0	0
5	1	0	0	0
6	0	0	1	1
7	0	1	0	0

Design by ROM: Example

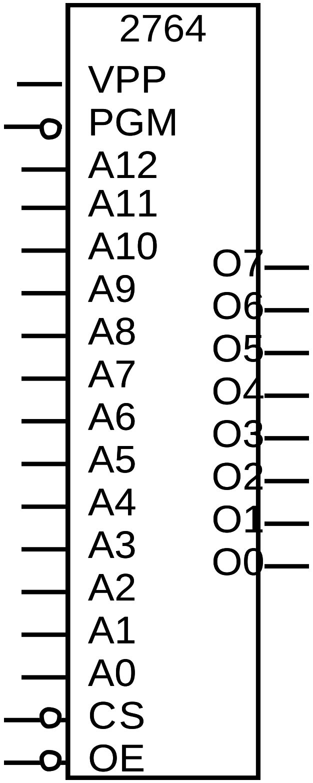
- BCD to 7 Segment Display Controller

A B C D	C0	C1	C2	C3	C4	C5	C6
0 0 0 0	1	1	1	1	1	1	0
0 0 0 1	0	1	1	0	0	0	0
0 0 1 0	1	1	0	1	1	0	1
0 0 1 1	1	1	1	1	0	0	1
0 1 0 0	0	1	1	0	0	1	1
0 1 0 1	1	0	1	1	0	1	1
0 1 1 0	1	0	1	1	1	1	1
0 1 1 1	1	1	1	0	0	0	0
1 0 0 0	1	1	1	1	1	1	1
1 0 0 1	1	1	1	0	0	1	1
1 0 1 0	X	X	X	X	X	X	X
1 0 1 1	X	X	X	X	X	X	X
1 1 0 0	X	X	X	X	X	X	X
1 1 0 1	X	X	X	X	X	X	X
1 1 1 0	X	X	X	X	X	X	X
0 1 1 1	X	X	X	X	X	X	X

Standard Devices

- Vpp and PGM are used when programming

2764 EPROM
8K x 8



THANK YOU