وزارة التعليم العالي والبحث العلمي

جامعة الفرات الأوسط التقنية

المعهد التقني / النجف

قسم تقنيات الالكترونية والاتصالات

المادة : الدوائر الرقمية

المرحلة : الأولى

تدريسي المادة : م.م زينب نعمان حمدي

اليميل الجامعي للتدريسي / Zainab.hamdi@atu.edu.iq

العام الدراسي 2026-2025

Middle Euphrates University - Technology

Technical Institute / Najaf

Department of Electronic Technologies and Communications

Instructor : Zainab Noman Hamdi

Details of theoretical vocabulary	The week
-Numerical systems of all kinds – binary, decimal, octet, sixteenth - Systems conversions / binary conversion to the rest of the systems and vice versa, decimal conversion to the rest of the systems and vice versa -Arithmetic operations in the binary system (addition - subtraction - division - multiplication) –Use complement1 and 2 in binary subtraction	First – Second – Third – Fourth Week
- Basic logic gates: AND gate, OR gate, NOT gate, logical operations of each gate with truth table and logical code -Composite gates: NAND gate, NOR gate, logical operations for each gate with truth table and logical code and used to build basic gates (AND, OR, NOT) - Exclusive gates: X-OR, X-NOR , logical operations of each gate with truth table and logical code with examples.	Fifth - sixth - seventh
-Boolean algebra - the laws and rules of Boolean algebra and their use in simplifying algebraic equations and logical circuits and finding the truth table. De Morcan's theorem – used to simplify algebraic equations, logical circuits and finding the truth table. - Write the logical equation from the truth table - either using the sum of the multiplied terms (Sum- of- product) or the factorial of the set terms (product-of - sum) and the conversion between them.	Week Eight-Nine
Carnot map (for two variables, three variables, four - variables): how to transfer the truth table to the K-map (K-	Week Ten-Eleven

map), various examples of simplifying equations and logical circuits using Carnot map, spin and entangle property.	and twelfth
- Arithmetic logical circuits of all kinds (half sum circle and perfect collector - half subtractor circle and complete subtractor - parallel addition to add two binary numbers - parallel addition to subtract two binary numbers Complement method 1s and 2s - different examples	Thirteenth - Fourteenth - Fifteenth
Comparisons: (single-rank comparative - two-rank comparator) - different examples - drawing the logical circle and its block.	Sixteenth
-Flip-flop vibrators :(S-R, D, JK, T) Adding synchronous pulses CLOK-PLUS to vibrators of all kinds.	Seventeenth-Eighteenth week
Comparison between different vibrators – examples of - linking vibrators with the shape of the output pulse depending on the input pulse and the synchronous pulse CP for each type.	Nineteenth
- Counters (general idea) Ascending ripple counter, descending ripple counter, decimal ripple counter. -Ascending and descending ripple counter. -Consecutive synchronous counter, parallel synchronous counter	Twentieth - Twenty-first - Twenty-second
Records of displacement of all kinds (input sequence - output sequence, input sequence - output parallel, parallel input - output sequence.	Twenty-third – Twenty-fourth
Memory circuits type Ram - ROM , memory description, memory capacity, mass chart of the main types of memory, electronic and logical circuits of memory.	Twenty-fifth – twenty-sixth
- Conversion from digital to analog type network resistors accuracy and provisions. - Conversion from peer to digital using the simultaneous method and the comparative method - ADC using the ascending counter.	Twenty-seventh - Twenty-eighth

-ADC using the counter ascending – descending.	**-Twenty-ninth – Thirtieth**

C. Central Ideas (1)

First: Conversion from decimal to binary system and vice versa.
Second: Conversion from the decimal system to the octagonal system and vice versa.
Third: Conversion from the decimal system to the hexadecimal system and vice versa.
Fourth: Conversion from the binary system to the eight-member system.
Fifth: Conversion from binary to hexadecimal system.
Sixth: Conversion from the eightfold system to the binary system.
Seventh: Conversion from the hexagonal system to the binary system.
Eighth: Adding numbers in the binary system
Ninth: Subtracting numbers in the binary system
Tenth: Subtracting numbers in the binary system
Eleven: Complement (1) and Complement of (2).
Twelve: Subtracting Numbers in Binary System Using the Complement of (1)
Thirteen: Subtracting Numbers in the binary system using the complement of (2)

د- Objectives of the lecture

After studying this lecture, the student will be able to:

- Identify the method of numbering indifferent numerical systems .

- Determines the types of transformations betweennumerical systems and find the value of the number for any numerical system.

- Recognize the method of addition and subtraction in the binary system.

- Extracts the complement of (1) and the complement of (2) in the binary system.

-Performs subtraction in binary system using the complement method of (1) and using the complement method of (2)

Digital Circuits – Integrated Assessment Section

Unit 1 – Number Systems

Pre-Test

1. The binary equivalent of decimal 79 is: A) 1101111■ B) 1101011■ C) 1001111■ D) 1001110■

2. The octal equivalent of decimal 496 is: A) 670■ B) 760■ C) 770■ D) 780■

3. The hexadecimal equivalent of decimal 6745 is: A) 1A59■■ B) 1A95■■ C) 1B95■■ D) 1B59■■

4. The decimal equivalent of binary 010111010.1■ is: A) 29.66525 B) 28.65252 C) 27.56562 D) 26.65625

5. The 1’s complement of 1101111■ is: A) 0010000■ B) 1101010■ C) 1001111■ D) 1001110■

6. The 2’s complement of 11011011■ is: A) 00100100■ B) 11101010■ C) 00100101■ D) 10101110■

7. The sum of 1101101■ + 1101001■ is: A) 00100100■ B) 11010110■ C) 00100101■ D) 10101110■

8. Subtract 110101■ from 1101001■ using the 2’s-complement method: A) 00100100■ B) 11010110■ C) 00100101■ D) 110100■

Lecture One

Number systems and operations

The study of number systems is important from the viewpoint of understanding how data are represented before they can be processed by any digital system including a digital computer. It is one of the most basic topics in digital electronics. In this chapter, we will discuss different number systems commonly used to represent data. We will begin the discussion with the decimal number system. Although it is not important from the viewpoint of digital electronics, a brief outline of this will be given to explain some of the underlying concepts used in other number systems. This will then be followed by the more commonly used number systems such as the binary, octal and hexadecimal number systems.

1. Decimal numbers:

The decimal number system is a radix-10 number system and therefore has 10 different digits or symbols. These are 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9. All higher numbers after '9' are represented in terms of these 10 digits only. The process of writing higher-order numbers after '9' consists in writing the second digit (i.e. '1') first, followed by the other digits, one by one, to obtain the next 10 numbers from '10' to '19'. The next 10 numbers from '20' to '29' are obtained by writing the third digit (i.e. '2') first, followed by digits '0' to '9', one by one.

The place values of different digits in a mixed decimal number, starting from the decimal point, are 10^0, 10^1, 10^2 and so on (for the integer part) and 10^{-1}, 10^{-2}, 10^{-3} and so on (for the fractional part).

As an illustration, in the case of the decimal number 3586.265, the integer part 3586 can be expressed as

$3586 = 6\times10^{0} +8\times10^{1}+5\times10^{2} +3\times10^{3} = 6+80+500+3000 = 3586$

and the fractional part 265 can be expressed as

$265 = 2\times10^{-1}+6\times10^{-2} +5\times10^{-3} = 0.2+0.06+0.005 = 0.265$

2. Binary Numbers:

The binary number system its two digits a base-two system. The two binary digits are 1 and 0 (1,0).

Binary weight	2^3	2^2	2^1	2^0
Weight value	8	4	2	1

A binary digit, called a bit, has two values 0 & 1. Each coefficient $\mathbf{a^j}$ **is multiplied by** $\mathbf{2^j}$, and the results are added to obtain the decimal equivalent of the number. For example,

11010.11 is equal to 26.75 as follows:

$$1\times2^4 + 1\times2^3 + 0\times2^2 + 1\times 2^1 + 0\times2^0 + 1\times 2^{-1} + 1\times 2^{-2}$$

$$\mathbf{16 + 8 + 0 + 2 + 0 + 0.5 + 0.25 = (26.75)_{10}}$$

In general, a number expressed in a base-r system has coefficients multiplied by powers of **r**.

$$\mathbf{r^n\times a_n + r^{n-1}\times a_{n-1} + \ldots.. + r^2\times a_2 + r^1\times a_1 + 1\times a_0 + r^{-1}\times a_{-1} + r^{-2}\times a_{-2} + \ldots.. + r^{-m}\times a_{-m}}$$

Table 1

Decimal	***Binary (0,1)***			
	8	***4***	***2***	***1***
	2^3	2^2	2^1	2^0
0	***0***	***0***	***0***	***0***
1	***0***	***0***	***0***	***1***
2	***0***	***0***	***1***	***0***
3	***0***	***0***	***1***	***1***
4	***0***	***1***	***0***	***0***
5	***0***	***1***	***0***	***1***
6	***0***	***1***	***1***	***0***
7	***0***	***1***	***1***	***1***
8	***1***	***0***	***0***	***0***
9	***1***	***0***	***0***	***1***
10	***1***	***0***	***1***	***0***
11	***1***	***0***	***1***	***1***
12	***1***	***1***	***0***	***0***
13	***1***	***1***	***0***	***1***
14	***1***	***1***	***1***	***0***
15	***1***	***1***	***1***	***1***

LSB (right-most bit) has a weight of $2^0 = 1$.

MSB (left- most bit) has a weight of $2^3 = 8$.

3. Octal numbers:

The octal number system has a radix of 8 and therefore has eight distinct digits. All higher-order numbers are expressed as a combination of these on the same pattern as the one followed in the case of the binary and decimal number systems described above.

Table 2

Decimal	*Octal*
0	*0*
1	*1*
2	*2*
3	*3*
4	*4*
5	*5*
6	*6*
7	*7*
8	*10*
9	*11*
10	*12*
11	*13*
12	*14*
13	*15*
14	*16*
15	*17*

3. Hexadecimal Numbers:

The hexadecimal number system is a radix-16 number system and its 16 basic digits are shown below.

Table 3

Decimal	***Hexadecimal***
0	*0*
1	*1*
2	*2*
3	*3*
4	*4*
5	*5*
6	*6*
7	*7*
8	*8*
9	*9*
10	*A*
11	*B*
12	*C*
13	*D*
14	*E*
15	*F*

Number Systems Conversions:

1- Binary - to - Decimal Conversion:

The decimal value of any binary number can be found by adding the weights of all bits that are 1 and discarding the weights of all bits that are 0.

EX1. The decimal equivalent of the binary number $(1001.0101)_2$ is determined as follows:

- **The integer part = 1001**
- **The decimal equivalent=**

$$1 \times 2^0 + 0 \times 2^1 + 0 \times 2^2 + 1 \times 2^3 = 1 + 0 + 0 + 8 = 9$$

- **The fractional part =0 .0101**
- **Therefore, the decimal equivalent = $0 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 2^{-4}$ = 0 + 0.25 + 0+ 0.0625 = 0.3125**
- **Therefore, the decimal equivalent of $(1001.0101)_2 = (9.3125)_{10}$**

2- Octal - to - Decimal Conversion:

The decimal equivalent of the octal number $(137.21)_8$ is determined as follows:

- **The integer part = 137**
- **The decimal equivalent = $7 \times 8^0 + 3 \times 8^1 + 1 \times 8^2 = 7 + 24 + 64 = 95$**
- **The decimal equivalent = $2 \times 8^{-1} + 1 \times 8^{-2} = 0.265$**
- **Therefore, the decimal equivalent of $(137.21)_8 = (95.265)_{10}$**

3- Hexadecimal - to - Decimal Conversion:

The decimal equivalent of the hexadecimal number $(1E0.2A)_{16}$ is determined as follows:

- **The integer part = 1E0**
- **The decimal equivalent = $0 \times 16^0 + 14 \times 16^1 + 1 \times 16^2 = 0 + 224 + 256 = 480$**
- **The fractional part = 2A**
- **The decimal equivalent = $2 \times 16^{-1} + 10 \times 16^{-2} = 0.164$**
- **Therefore, the decimal equivalent of $(1E0.2A)_{16} = (480.164)_{10}$**

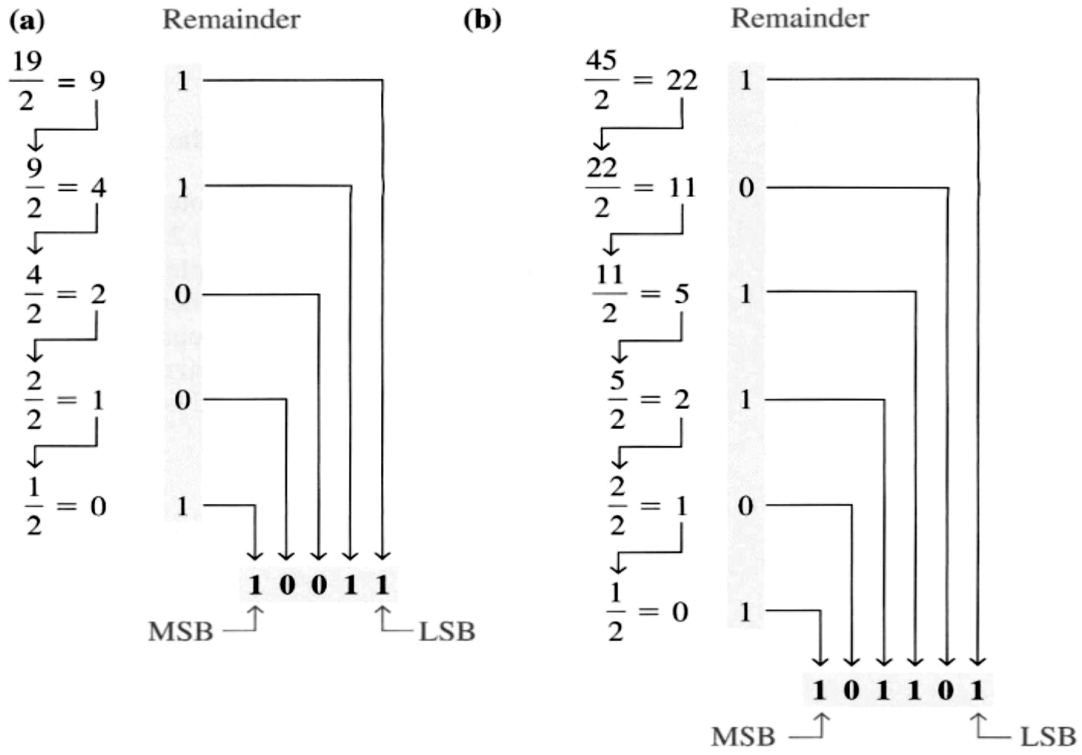
4- Decimal-to-Binary Conversion:

As outlined earlier, the integer and fractional parts are worked on separately. For the integer part, the binary equivalent can be found by successively dividing the integer part of the number by 2 and recording the remainders until the quotient becomes '0'. The remainders written in reverse order constitute the binary equivalent. For the fractional part, it is found by successively multiplying the fractional part of the decimal number by 2 and recording the carry until the result of multiplication is '0'. The carry sequence written in forward order constitutes the binary equivalent of the fractional part of the decimal number. If the result of multiplication does not seem to be heading towards zero in the case of the fractional part, the process may be continued only until the requisite number of equivalent bits has been obtained.

This method of decimal–binary conversion is popularly known as the double-dabble method. The process can be best illustrated with the help of a division process as explained in the following example:

EX2. Convert the following numbers from decimal to binary:

(a) 19 (b) 45



EX3. Convert $(0.6875)_{10}$ to binary:

	Integer	Fraction	Coefficient
0.6875*2	1+	0.375	1
0.375*2	0+	0.75	0
0.75*2	1+	0.5	1
0.5*2	1+	0	1

$(0.6875)_{10} = (0.1011)_2$

5- Decimal - to - Octal Conversion:

The process of decimal-to-octal conversion is similar to that of decimal-to-binary conversion. The progressive division in the case of the integer part and the progressive multiplication while working on the fractional part here are by '8' which is the radix of the octal number system. Again, the integer and fractional parts of the decimal number are treated separately. The process can be best illustrated with the help of an example.

<u>EX4.</u> Convert $(73.75)_{10}$ to octal

1. Integer part

Dividend	Remainder
73/8	1
9/8	1
1/8	1

$(73)_{10} = (111)_8$

2. Fractional

	Integer	Fraction	Coefficient
0.75*8	6+	0	6

$(73.75)_{10} = (111.6)_8$

6- Decimal - to - Hexadecimal Conversion:

The process of decimal-to-hexadecimal conversion is also similar. Since the hexadecimal number system has a base of 16, the progressive division and multiplication factor in this case is 16. The process is illustrated further with the help of an example.

EX4. Let us determine the hexadecimal equivalent of $(82.25)_{10}$

1. **Integer part**

Dividend	Remainder
82/16	**2**
5/16	5

2. **Fractional part**

	Integer	Fraction	Coefficient
0.25*16	4+	0	4

Therefore, the hexadecimal equivalent of $(82.25)_{10} = (52.4)_{16}$

Binary–Octal and Octal–Binary Conversions:

An octal number can be converted into its binary equivalent by replacing each octal digit with its three-bit binary equivalent. We take the three-bit equivalent because the base of the octal number system is 8 and it is the third power of the base of the binary number system, i.e. 2. All we have then to remember is the three-bit binary equivalents of the basic digits of the octal number system. A binary number can be converted into an equivalent octal number by splitting the integer and fractional parts into groups of three

bits, starting from the binary point on both sides. The 0s can be added to complete the outside groups if needed.

EX5. Let us find the binary equivalent of $(374.26)_8$ and the octal equivalent of $(1110100.0100111)_2$

1- $(374.26)_8 = (?)_2$

Octal	3	7	4	2	6
Binary	011	111	100	010	110

$(374.26)_8 = (011111100.010110)_2$

2- $(1110100.0100111)_2 = (?)_8$

$(001 \quad 110 \quad 100 . 010 \quad 011 \quad 100)_2$

Binary	001	110	100	010	011	100
Octal	1	6	4	2	3	4

$(1110100.0100111)_2 = (164.234)_8$

Hex–Binary and Binary–Hex Conversions:

A hexadecimal number can be converted into its binary equivalent by replacing each hex digit with its four-bit binary equivalent. We take the four-bit equivalent because the base of the hexadecimal number system is 16 and it is the fourth power of the base of the binary number system. All we have then

to remember is the four-bit binary equivalents of the basic digits of the hexadecimal number system. A given binary number can be converted into an equivalent hexadecimal number by splitting the integer and fractional parts into groups of four bits, starting from the binary point on both sides. The 0s can be added to complete the outside groups if needed.

<u>EX6.</u> Let us find the binary equivalent of $(17E.F6)_{16}$ and the hex equivalent of $(1011001110.011011101)_2$.

1- $(17E.F6)_{16} = (?)_2$

Hex	1	7	E	F	6
Binary	0001	0111	1110	1111	0110

$(17E.F6)_{16} = (101111110.1111011)_2$

2- $(1011001110.011011101)_2 = (?)_{16}$

$(0010\ 1100\ 1110 . 0110\ 1110\ 1000)_2$

Binary	0010	1100	1110	0110	1110	1000
Hex	2	C	E	6	E	8

$(1011001110.011011101)_2 = (2CE.6E8)_{16}$

<u>H.W.</u> Find the octal equivalent of $(2F.C4)_{16}$ and the hex equivalent of $(762.013)_8$?

Table 4

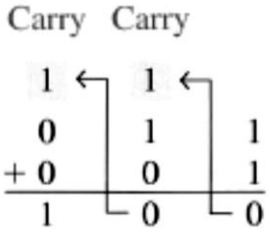
Decimal	Binary	Octal	Hexadecimal
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Binary Arithmetic:

1- Binary Addition

0 + 0 = 0	Sum of 0 with a carry of 0
0 + 1 = 1	Sum of 1 with a carry of 0
1 + 0 = 1	Sum of 1 with a carry of 0
1 + 1 = 10	Sum of 0 with a carry of 1

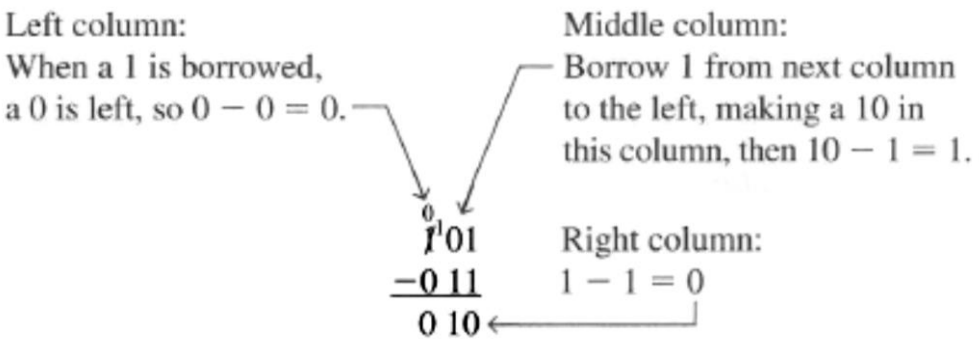
Ex// Find $(11+01)_2$?



2- Binary Subtraction

0 − 0 = 0	
1 − 1 = 0	
1 − 0 = 1	
10 − 1 = 1	0 − 1 with a borrow of 1

EX// Find $(101-11)_2$?



3- Binary Multiplication

$$0 \times 0 = 0$$
$$0 \times 1 = 0$$
$$1 \times 0 = 0$$
$$1 \times 1 = 1$$

EX// Find $(11x11)_2$?

$$\begin{array}{rl} & 11 \\ & \times\ 11 \\ \text{Partial products} & \begin{array}{l} 11 \\ +11 \end{array} \\ & \mathbf{1001} \end{array}$$

4-Binary Division

This operation follows the same procedure as division in decimal number system.

EX//

$$110 \div 11 = 10$$

$$\begin{array}{r|l} & 10 \\ 11 & 110 \\ & -11 \\ & 000 \end{array}$$

الى هنا ملزمة العمليات الحسابية

1’s and 2’s Complement of Binary Numbers

The l's complement and the 2's complement of a binary number are important because they permit the representation of negative numbers. The method of 2's complement arithmetic is commonly used in computers to handle negative numbers.

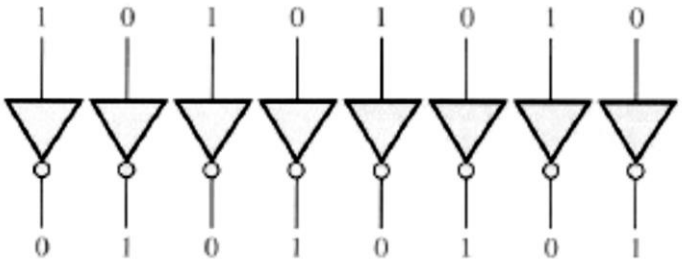
Finding the 1's Complement

The l's complement of a binary number is found by changing all 1s to 0s and all

0s to 1s, as illustrated below:

```
1 0 1 1 0 0 1 0      Binary number
↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓
0 1 0 0 1 1 0 1      1’s complement
```

The simplest way to obtain the l's complement of a binary number with a digital circuit is to use parallel inverters (NOT circuits), as shown in Fig. below for an 8-bit binary number



Finding the 2' s Complement

The 2's complement of a binary number is found by adding 1 to the LSB of the l's complement.

2's complement = (l's complement) + 1

EX// Find the 2's complement of 10110010.

	10110010	Binary number
	01001101	l's complement
+	1	add 1
	01001110	2's complement

Signed Numbers

Digital systems, such as the computer, must be able to handle both positive and negative numbers. A signed binary number consists of both sign and magnitude information. The sign indicates whether a number is positive or negative, and the magnitude is the value of the number. There are three forms in which signed integer (whole) numbers can be represented in binary: sign magnitude, l's complement, and 2' complement. Of these, the 2's complement is the most important and the sign-magnitude is the least used

أرقام موقعة

يجب أن تكون الأنظمة الرقمية ، مثل الكمبيوتر ، قادرة على التعامل مع الأرقام الموجبة والسالبة.

يتكون الرقم الثنائي الموقّع من معلومات الإشارة والحجم. تشير العلامة إلى ما إذا كان الرقم موجبًا أم سالبًا ، والحجم هو قيمة الرقم. هناك ثلاثة أشكال يمكن من خلالها تمثيل الأرقام الصحيحة (الصحيحة) الموقعة في ، ومكمل ٢. من بين هؤلاء ، المكمل ٢ هو الأكثر أهمية وحجم الإشارة هو l ثنائي: حجم الإشارة ، ومكمل الأقل استخدامًا

(ملاحظة الارقام الذي فيها الاشارات الموجبة والسالبة هناك ثلاث طرق لتحويل الاشارة من موجب الى السالب او العكس)

The Sign Bit

The left-most bit in a signed binary number is the sign bit, which tells you whether the number is positive or negative. A 0 sign bit indicates a positive number, and a 1 sign bit indicates a negative number.

A 0 sign bit indicates a positive number, and a 1 sign bit indicates a negative number

بت الاشارة

أقصى بت يسار رقم ثنائي بعلامة هو بت الإشارة ، والذي يخبرك ما إذا كان الرقم موجبًا أم سالبًا. تشير بت الإشارة ٠ إلى رقم موجب ، وتشير بت الإشارة ١ إلى رقم سالب.

تشير بت الإشارة ٠ إلى رقم موجب ، وتشير بت الإشارة ١ إلى رقم سالب

Sign-Magnitude Form

When a signed binary number is represented in sign-magnitude, the leftmost bit is the sign bit and the remaining bits are the magnitude bits. The magnitude bits are in true (un-complemented) binary for both positive and negative numbers. For example, the decimal number + 25 is expressed as an 8-bit signed binary number using the sign-magnitude form as 00011001. The decimal number -25 is expressed as 1001100l. Notice that the only difference between + 25 and - 25 is the sign bit because the magnitude bits are in true binary for both positive and negative numbers.

نموذج التوقيع الحجم

عندما يتم تمثيل رقم ثنائي بعلامة بحجم الإشارة ، فإن البت الموجود في أقصى اليسار هو بت الإشارة والبتات المتبقية هي بتات الحجم. بتات المقدار في ثنائي صحيح (غير مكمل) للأرقام الموجبة والسالبة.

على سبيل المثال ، يتم التعبير عن الرقم العشري + ٢٥ كرقم ثنائي موقع من ٨ بت باستخدام صيغة مقدار الإشارة كـ ٠٠٠١١٠٠١. يتم التعبير عن الرقم العشري -٢٥ كـ ١٠٠١١٠٠ .lلاحظ أن الاختلاف الوحيد بين + ٢٥ و - ٢٥ هو بت الإشارة لأن بتات المقدار تكون في ثنائي صحيح للأرقام الموجبة والسالبة.

In the sign-magnitude form, a negative number has the same magnitude bits as the corresponding positive number but the sign bit is a 1 rather than a zero.

في صيغة حجم الإشارة ، يكون للرقم السالب نفس بتات المقدار مثل الرقم الموجب المقابل لكن بتة الإشارة هي ١ بدلاً من صفر.

EX// Express the decimal number - 39 as an 8-bit number in the sign-magnitude, 1's complement, and 2's complement forms?

SOL// First, write the 8-bit number for + 39.

00100111

In the sign-magnitude form, - 39 is produced by changing the sign bit to a 1 and leaving the magnitude bits as they are. The number is

10100111

In the 1's complement form, -39 is produced by taking the l's complement of +39 (00100111).

11011000

In the 2's complement form, - 39 is produced by taking the 2's complement of +39 (00100111) as follows:

```
  11011000      1's complement
+       1
--------
  11011001      2's complement
```

The Decimal Value of Signed Numbers

Sign-magnitude: Decimal values of positive and negative numbers in the sign-magnitude form are determined by summing the weights in all the magnitude bit positions where there are 1s and ignoring those positions where there are zeros. The sign is determined by examination of the sign bit.

القيمة العشرية للأرقام الموقّعة

مقدار الإشارة: يتم تحديد القيم العشرية للأرقام الموجبة والسالبة في شكل حجم الإشارة عن طريق جمع وتجاهل تلك المواضع التي توجد بها أصفار. يتم تحديد sالأوزان في جميع مواضع بت الحجم حيث يوجد ١ العلامة من خلال فحص بت التوقيع

EX//

Determine the decimal value of this signed binary number expressed in sign-magnitude: 10010101

حدد القيمة العشرية لهذا الرقم الثنائي الذي يحمل علامة معبرًا عنه بحجم الإشارة

SOL//

The seven magnitude bits and their powers-of-two weights are as follows:

$$2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$$

$$0\ 0\ 1\ 0\ 1\ 0\ 1$$

Summing the weights where there are 1s,

$$16 + 4 + 1 = 21$$

The sign bit is 1; therefore, the decimal number is - 21.

بت الاشارة هو ١ ؛ لذلك ، فإن الرقم العشري هو - ٢١.

1's Complement:

Decimal values of positive numbers in the l's complement form are determined by summing the weights in all bit positions where there are 1s and ignoring those positions where there are zeros. Decimal values of negative numbers are determined by assigning a negative value to the weight of the sign bit, summing all the weights where there are 1s, and adding 1 to the result.

المتمم الاول (1s)

من خلال جمع الأوزان في جميع مواضع البت l يتم تحديد القيم العشرية للأرقام الموجبة في شكل المكمل وتجاهل تلك المواضع التي توجد بها أصفار. يتم تحديد القيم العشرية للأرقام السالبة عن sحيث يوجد ١ طريق تعيين قيمة سالبة لوزن بت الإشارة ، وجمع كل الأوزان حيث يوجد ١ ، وإضافة ١ إلى النتيجة.

Example:

Determine the decimal values of the signed binary numbers expressed in l's complement: (a) 00 010 111 (b) 11 101 000

Solution:

(a) The bits and their powers-of-two weights for the positive number are as follows:

-2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0

0 0 0 1 0 1 1 1

Summing the weights where there are 1s, 16 + 4 + 2 + 1 = +23

(b) The bits and their powers-of-two weights for the negative number are as follows. -2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0

1 1 1 0 1 0 0 0

Notice that the negative sign bit has a weight of -2^7 or -128.

Summing the weights where there are 1s, -128 + 64 + 32 + 8 = -24

Adding 1 to the result, the final decimal number is -24 + 1 = -23.

الى هنا

2's Complement:

Decimal values of positive and negative numbers in the 2's complement form are determined by summing the weights in all bit positions where there are 1's and ignoring those positions where there are zeros. The weight of the sign bit in a negative number is given a negative value.

s'٢ المتمم الثاني

يتم تحديد القيم العشرية للأرقام الموجبة والسالبة في نموذج المتمم الثاني عن طريق جمع الأوزان في جميع مواضع البت حيث يوجد ١ وتجاهل تلك المواضع التي توجد بها أصفار. يتم إعطاء وزن بت الإشارة في رقم سالب قيمة سالبة.

Example:

Determine the decimal values of the signed binary numbers expressed in 2's complement:

(a) 01010110 (b) 10101010

(a) The bits and their powers-of-two weights for the positive number are as follows:

-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	1	0	1	0	1	1	0

Summing the weights where there are 1s,

$64 + 16 + 4 + 2 = +86$

(b) The bits and their powers-of-two weights for the negative number are as follows. Notice that the negative sign bit has a weight of $-2^7 = -128$.

-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	0	1	0	1	0	1	0

Summing the weights where there are 1s, $-128 + 32 + 8 + 2 = -86$

Binary Coded Decimal (BCD)

Binary coded decimal (BCD) is a way to express each of the decimal digits with a binary code. There are only ten code groups in BCD system, so it is very easy to convert between decimal and BCD. Because we like to read and write in decimal, the interfaces are keypad input and digital readout.

ثنائي عشري مشفر (BCD)

طريقة للتعبير عن كل رقم من الأرقام العشرية برمز ثنائي. لا يوجد (BCD) يعد النظام العشري الثنائي BCD. ، لذلك من السهل جدًا التحويل بين النظام العشري و BCD سوى عشر مجموعات رموز في نظام نظرًا لأننا نحب القراءة والكتابة بالنظام العشري ، فإن الواجهات هي إدخال لوحة المفاتيح والقراءة الرقمية

The 8421code:

The 8421 code is a type of BCD (binary coded decimal) code. Binary coded decimal digit, 0 thought 9, is represented by a binary code of four bits. The designation 8421 indicated the binary weights of the four bits **(2^3,2^2,2^1,2^0)**. The ease of conversion between 8421 code number and the familiar decimal numbers is the main advantage of this code. All you have to remember are the ten binary combinations that represent the ten decimal digits as shown in table below. The 8421 code is predominant BCD code, and when we refer to BCD, we always mean the 8421 code unless otherwise stated.

Binary decimal code (BCD)	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	9
	0000	**0001**	**0010**	**0011**	**0100**	**0101**	**0110**	**0111**	**1000**	**1001**

رمز ٨٤٢١:

الكود ٨٤٢١ هو نوع من كود BCD (عشري مشفر ثنائي). الرقم العشري الثنائي ، ٠ الفكر ٩ ، يتم تمثيله برمز ثنائي مكون من أربعة بتات. أشار التصنيف ٨٤٢١ إلى الأوزان الثنائية للبتات الأربع (٢٣ ، ٢٢ ، ٢١ ، ٢٠). سهولة التحويل بين رقم الكود ٨٤٢١ والأرقام العشرية المألوفة هي الميزة الرئيسية لهذا الكود. كل ما عليك تذكره هو المجموعات الثنائية العشر التي تمثل الأرقام العشرية العشرة كما هو موضح في

الجدول أدناه. رمز ٨٤٢١ هو رمز BCD السائد ، وعندما نشير إلى BCD ، فإننا نعني دائمًا رمز ٨٤٢١ ما لم يذكر خلاف ذلك رموز غير صالحة

Invalid Codes:

It may be realized that, with four bits, sixteen numbers (0000 through 1111) can be represented but that, in the 8421 code, only ten of these are used. The six code combinations that are not used (**1010, 1011, 1100, 1101, 1110, 1111)** are invalid in 8421 BCD code. To express any decimal number in BCD, simply replace each decimal digit with the appropriate 4-bit code as shown by Example

يمكن إدراك أنه من خلال أربع بتات ، يمكن تمثيل ستة عشر رقمًا (من ٠٠٠٠ إلى ١١١١) ولكن في رمز ٨٤٢١ ، يتم استخدام عشرة فقط من هذه الأرقام. مجموعات الأكواد الستة غير المستخدمة (١٠١٠ ، ١٠١١ ، ١١٠٠ ، ١١٠١ ، ١١١٠ ، ١١١١) غير صالحة في كود ٨٤٢١ BCD. ، ما عليك سوى استبدال كل BCD للتعبير عن أي رقم عشري في رقم عشري برمز ٤ بت المناسب كما هو موضح في المثال

EX// Convert each of the following decimal numbers to BCD codes:

a) 35 b) 98 c) 170

Solution:

a) 35 b) 98 c) 170

3	5	9	8	1	7	0
↓	↓	↓	↓	↓	↓	↓
0011	0101	1001	1000	0001	0111	0000

EX//

Convert each of the following BCD codes to decimal:

(a) 10000110 **(b)** 001101010001 **(c)** 1001010001110000

(a) 1000 0110 **(b)** 0011 0101 0001 **(c)** 1001 0100 0111 0000

↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓

8 6 **3 5 1** **9 4 7 0**

BCD Addition

BCD is a numerical code and can be used in arithmetic operation. Addition is the most important operation because the other three operations (subtraction, multiplication, and deviation) can be accomplished by use of the addition. Here, is how to add two BCD numbers

1. Add the two BCD numbers, using the rules for binary addition.

2. If a 4-bit sum is equal to or less than 9, it is a valid BCD number.

3. If a 4-bit sum is greater than 9, or if a carry out of the four-bit group is generated, it is an invalid result. **Add 6(0110)** to 4-bit sum in order to skip the six invalid states and return the code to 8421. If a carry results when 6 is added, simply add the carry to the next 4-bit group.

Ex//

Add the following BCD numbers:

(a) 0011 + 0100 **(b)** 00100011 + 00010101

(c) 10000110 + 00010011 **(d)** 010001010000 + 010000010111

The decimal number additions are shown for comparison.

(a)

0011	3
+ 0100	+ 4
0111	7

(b)

0010	0011	23
+ 0001	0101	+ 15
0011	**1000**	38

(c)

1000	0110	86
+ 0001	0011	+ 13
1001	**1001**	99

(d)

0100	0101	0000	450
+ 0100	0001	0111	+ 417
1000	**0110**	**0111**	867

Note that in each case the sum in any 4-bit column does not exceed 9, and the results are valid BCD numbers.

Ex//

Add the following BCD numbers

(a) 1001 + 0100 **(b)** 1001 + 1001

(c) 00010110 + 00010101 **(d)** 01100111 + 01010011

The decimal number additions are shown for comparison.

(a)

1001			9
+ 0100			+4
1101		Invalid BCD number (>9)	13
+ 0110		Add 6	
0001	**0011**	Valid BCD number	
↓	↓		
1	3		

(b)

1001			9
+ 1001			+ 9
1 0010		Invalid because of carry	18
+ 0110		Add 6	
0001	**1000**	Valid BCD number	
↓	↓		
1	8		

(c)

0001	0110		16
+ 0001	0101		+ 15
0010	1011	Right group is invalid (>9), left group is valid.	31
	+ 0110	Add 6 to invalid code. Add carry, 0001, to next group.	
0011	**0001**	Valid BCD number	
↓	↓		
3	1		

(d)

	0110	0111		67
	+ 0101	0011		+ 53
	1011	1010	Both groups are invalid (>9)	120
	+ 0110	+ 0110	Add 6 to both groups	
0001	**0010**	**0000**	Valid BCD number	
↓	↓	↓		
1	2	0		

The Gray code

The Gray code is un-weighted and is not an arithmetic code; that is, there are no specific weight assigned to the bit positions. The important feature of the **Gray code is that it exhibits only a single bit change from one code word to the next in sequence.** This property is important in many applications, such as shaft position encoder, where error susceptibility increases with the number of bit change between adjacent numbers in sequence.

Binary to Gray code conversion:

Conversion between binary code and Gray code is sometimes useful. The following rules explain how to convert from a binary number to a Gray code word:

1. The most significant bit (left most) in the Gray code is the same as the corresponding MSB in the binary number.
2. Going from left to right, add each adjacent pair of binary code bits to get the next Gray code bit .Discard carries.

For example, the conversion of the binary number 10110 to Gray code is as the follows:

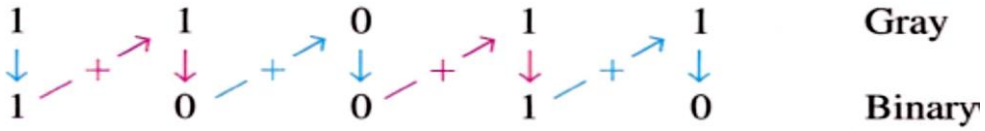
The Gray code is (11101)

Gray to Binary conversion

To convert Gray code to binary code use similar method; however, there are some differences. The following rules apply:

1. The most significant bit (left-most) in the binary code is the same as the corresponding in the gray code
2. Add each binary code bit generated to Gray code bit in the next adjacent position. Discard carries.

For example, the conversion of the Gray code word 11011 to binary is as follows:



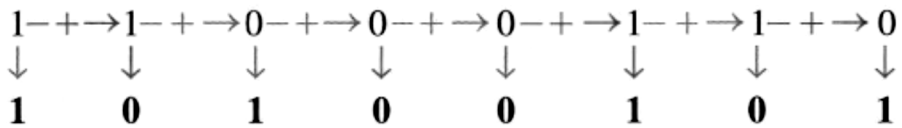
The binary number is (10010)

EX//

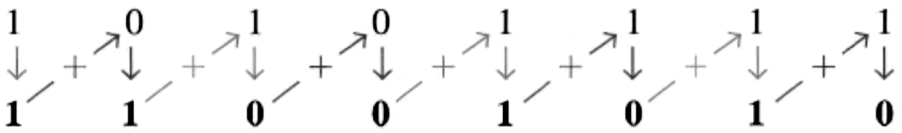
(a) Convert the binary number 11000110 to Gray code.

(b) Convert the Gray code 10101111 to binary.

(a) Binary to Gray code:



(b) Gray code to binary:



H.W.

(a) Convert binary 101101 to Gray code. **(b)** Convert Gray code 100111 to binary.

Sheet of Problems

1-1 List the first 16 numbers in base 12. Use the letters A and B to represent the last two digits.

1-2 What is the largest binary number that can be obtained with 16 bits? What is its decimal equivalent?

1-3 Convert the following binary numbers to decimal: 101110; 1110101.11; and 110110100.

1-4 Convert the following numbers with the indicated bases to decimal: $(12121)_3$; $(4310)_5$; $(50)_7$; and $(198)_{12}$.

1-5 Convert the following decimal numbers to binary: 1231; 673.23; 10^4; and 1998.

1-6 Convert the following decimal numbers to the indicated bases:
(a) 7562.45 to octal.
(b) 1938.257 to hexadecimal.
(c) 175.175 to binary.

1-7 Convert the hexadecimal number F3A7C2 to binary and octal.

1-8 Convert the following numbers from the given base to the other three bases indicated.
(a) Decimal 225 to binary, octal, and hexadecimal.
(b) Binary 11010111 to decimal, octal, and hexadecimal.
(c) Octal 623 to decimal, binary, and hexadecimal.
(d) Hexadecimal 2AC5 to decimal, octal, and binary.

1-9 Add and multiply the following numbers without converting to decimal.
(a) $(367)_8$ and $(715)_8$.
(b) $(15F)_{16}$ and $(A7)_{16}$.
(c) $(110110)_2$ and $(110101)_2$.

1-10 Perform the following division in binary: 11111111/101.

1-15 Find the 1's and 2's complements of the following 8-digit binary numbers: 10101110; 10000001; 10000000; 00000001; and 00000000.

(a) 11010 − 10000
(b) 11010 − 1101
(c) 100 − 110000
(d) 1010100 − 1010100

1-18 Perform the arithmetic operations (+42) + (−13) and (−42) − (−13) in binary using the signed-2's-complement representation for negative numbers.

1-19 The binary numbers listed have a sign in the leftmost position and, if negative, are in 2's-complement form. Perform the arithmetic operations indicated and verify the answers.
(a) 101011 + 111000
(b) 001110 + 110010
(c) 111001 − 001010
(d) 101011 − 100110

1-17 Perform the subtraction with the following unsigned binary numbers by taking the 2's complement of the subtrahend.

1-20 Represent the following decimal numbers in BCD: 13597; 93286; and 99880.

Post-Test

1. Find x: (110110.11001)■ = x■■.
2. Convert (10110111011.01010)■ to octal.
3. Convert (749.586)■■ to binary.
4. Convert (11011100110.110101011)■ to hexadecimal.
5. Convert (7605.4203)■ to binary.
6. Using the 2’s complement, subtract (101001)■ from (1010110)■.
7. Using the 1’s complement, subtract (1100101)■ from (10110)■.
8. Add (100110)■ and (111011)■.
9. Using the 2’s complement, subtract (1011011)■ from (101101)■.
10. Add (10110)■ and (111011)■.

Digital Circuits – Integrated Assessment Section

Unit 2 – Logic Gates

Pre-Test

1. Logic gates are electronic circuits that: A) Make logical decisions B) Accelerate electrons in one direction C) Operate using the binary number system D) Have an output between 1 and 5 V.

2. In logic circuits, logic 1 represents: A) Positive voltage B) High voltage C) Negative voltage D) Low voltage.

3. A two-input OR gate has output zero when: A) Both inputs are 0 B) One input is 1 C) Both inputs are 1 D) One input is 0.

4. A two-input NAND gate has output zero when: A) Both inputs are 1 B) Both inputs are 0 C) One input is 0 D) Input is 1.

- Central Ideas

First: Representing logic gates (AND , OR , NOT) using keys and extracting their realistic tables.
Second: Representation of logic gates (AND , OR , NOT) using diode, resistance and transistor.
Third: Representation of logic gates (XOR , XNOR) using different gates and extracting their realistic tables.
Fourth: Representing the different gates using the NAND gate once and the NOR gate again.
Fifth: Writing the equation for taking out logical gates.

Objectives of the lecture

After studying this lecture, the student will be able to:

- Learn how to build different logic gates using keys.
- Writes the equation for taking out different logic gates.
- Draws the logical code for each gate.
- Learn how to represent different gates using NAND gate once and NOR gate again.

Lecture Two

Logic Gates

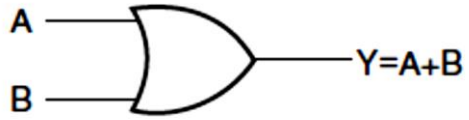
Logic gates are electronic circuits that can be used to implement the most elementary logic expressions, also known as Boolean expressions. The logic gate is the most basic building block of combinational logic. There are three basic logic gates, namely the OR gate, the AND gate and the NOT gate. Other logic gates that are derived from these basic gates are the NAND gate, the NOR gate, the EXCLUSIVEOR gate and the EXCLUSIVE-NOR gate.

1. OR Gate:

An OR gate performs an ORing operation on two or more than two logic variables. The OR operation on two independent logic variables A and B is written as Y = A+B and reads as Y equals A OR B and not as A plus B. An OR gate is a logic circuit with two or more inputs and one output. The output of an OR gate is LOW only when all of its inputs are LOW. For all other possible input combinations, the output is HIGH. This statement when interpreted for a positive logic system means the following. The output of an OR gate is a logic ‘0’ only when all of its inputs are at logic ‘0’. For all other possible input combinations, the output is a logic ‘1’. Figure (1) shows the circuit symbol and the truth table of a two-input OR gate.

The operation of a two-input OR gate is explained by the logic expression:

$$Y = A+B$$



A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

Fig. (1): the circuit symbol and the truth table of a two-input OR gate.

Now let us look at the operation of an OR gate with pulse waveform inputs, keeping in mind its logical operation. Again, the important thing in the analysis of gate operation with pulse waveforms is the time relationship of all the waveforms involved. For example, in Fig.(2), inputs A and B are both HIGH (1) during time interval t1 making output X HIGH (1). During time interval t2, input A is LOW (0), but because input B is HIGH (1), the output is HIGH (1). Both inputs are LOW (0) during time interval t3 , so there is a LOW (0) output during this time. During time interval t4 , the output is HIGH (1) because input A is HIGH (1).

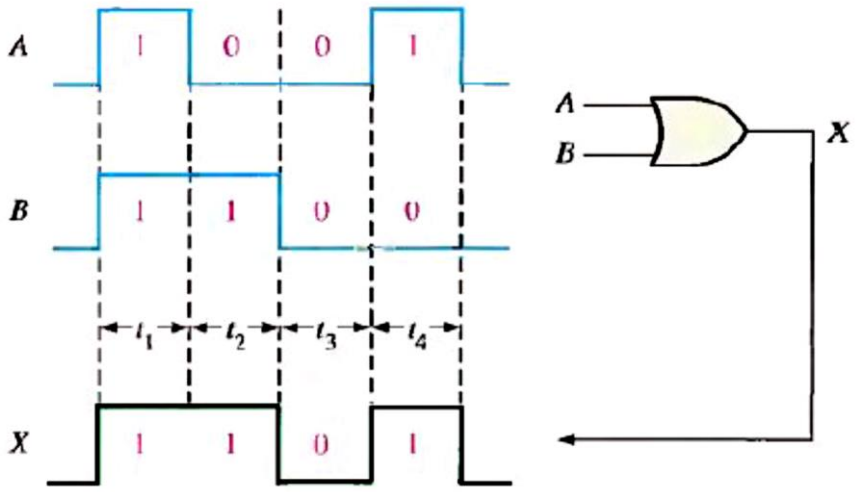


Fig. (2): Example of OR gate operation with a timing diagram showing input and output relationships.

As an illustration, if we have four logic variables and we want to know the logical output of (A+B+C +D), then it would be the output of a four-input OR gate with A, B, C and D as its inputs.

Figures 3(a) and (b) show the circuit symbol of three-input and four-input OR gates. Figure 3(c) shows the truth table of a three-input OR gate. Logic expressions explaining the functioning of three input and four-input OR gates are **Y = A+B+C** and **Y = A+B+C +D**

The total number of possible combinations of binary inputs to a gate is determined by the following formula:

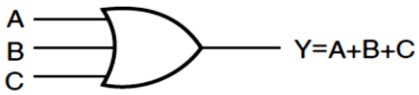
$$N=2^n$$

where N is the number of possible input combinations and n is the number of input variables. To illustrate:

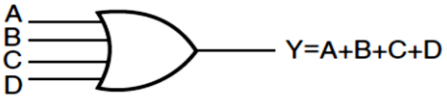
For two input variables: $N = 2^2 = 4$ combinations.

For three input variables: $N = 2^3 = 8$ combinations.

For four input variables: $N = 2^4 = 16$ combinations.



(a)



(b)

A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

(c)

Fig. (3)

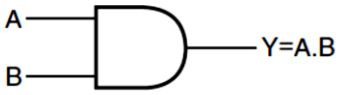
2. AND Gate:

An AND gate is a logic circuit having two or more inputs and one output. The output of an AND gate is HIGH only when all of its inputs are in the HIGH state. In all other cases, the output is LOW. When interpreted for a positive logic system, this means that the output of the AND gate is a logic ‘1’ only when all of its inputs are in logic ‘1’ state. In all other cases, the output is logic ‘0’.

The AND operation on two independent logic variables A and B is written as:

$$\mathbf{Y = A.B}$$

and reads as **Y** equals **A AND B** and not as **A multiplied by B**. Here, A and B are input logic variables and Y is the output. An AND gate performs an ANDing operation



(a)

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

(b)

Fig. (4): AND gate.

Let's examine the waveform operation of an AND gate by looking at the inputs with respect to each other in order to determine the output level at any given time. In Fig.(5), inputs A and B are both HIGH (1) during the time interval, t1 making output X HIGH (1) during this interval. During time interval t2 input A is LOW (0) and input B is HIGH (1), so the output is LOW (0). During time interval t3 , both inputs are HIGH (1), and therefore the output is HIGH (1). During time interval t4 , input A is HIGH 0) and input B is LOW (0), resulting in a LOW (0) output. Finally, during time interval t5 , input A is LOW (0), input B is LOW (0), and the output is therefore LOW (0). As you know, a diagram of input and output waveforms showing time relationships is called a timing diagram.

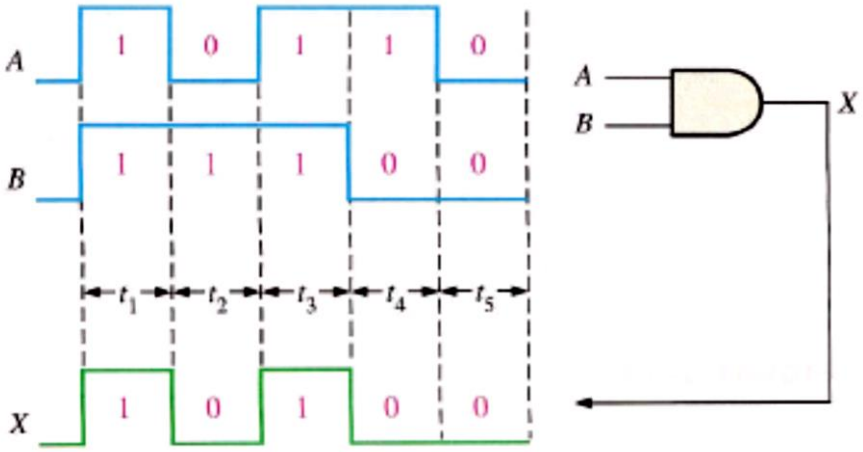


Fig. (5)

3. NOT Gate:

A NOT gate is a one-input, one-output logic circuit whose output is always the complement of the input. That is, a LOW input produces a HIGH output, and vice versa. When interpreted for a positive logic system, a logic '0' at the input produces a logic '1' at the output, and vice versa. It is also known as a 'complementing circuit' or an 'inverting circuit'. Figure 6 shows the circuit symbol and the truth table.

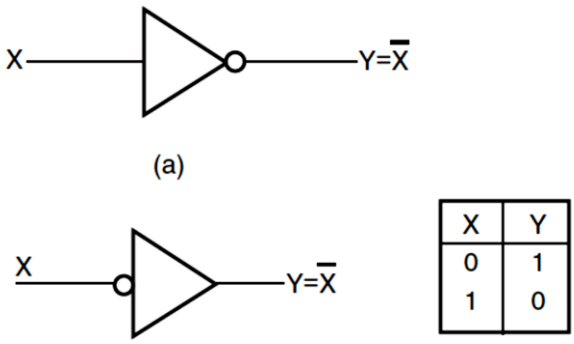


Fig. (6): Circuit symbol and the truth table of a NOT circuit.

The operation of an inverter (NOT circuit) can be expressed as follows:

If the input variable is called A and the output variable is called X, then

$$X = \overline{A}$$

This expression states that the output is the complement of the input, so if A= 0, then X = 1, and if A = 1, then X = 0.

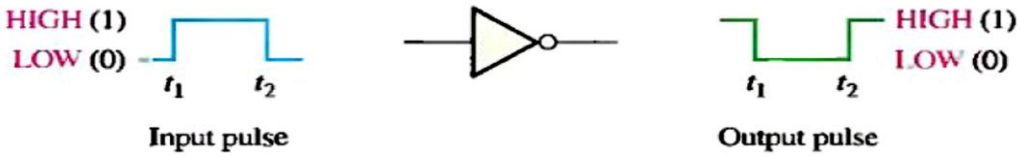


Fig. (7): Inverter operation.

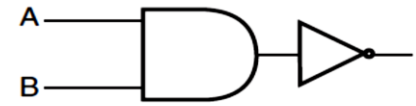
4. NAND Gate:

NAND stands for NOT AND. An AND gate followed by a NOT circuit makes it a NAND gate [Fig.8 (a)]. Fig.8 (b) shows the circuit symbol of a two-input NAND gate. The truth table of a NAND gate is obtained from the truth table of an AND gate by complementing the output entries [Fig.8(c)]. The output of a NAND gate is a logic '0' when all its inputs are a logic '1'. For all other input combinations, the output is a logic '1'. NAND gate operation is logically expressed as:

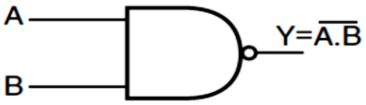
$$Y = \overline{A.B}$$

In general, the Boolean expression for a NAND gate with more than two inputs can be written as:

$$Y = \overline{(A.B.C.D...)}$$



(a)



(b)

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

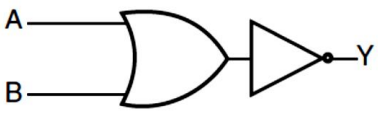
(c)

Fig. (8): (a) Two-input NAND implementation using an AND gate and a NOT circuit, (b) the circuit symbol of a two-input NAND gate, and (c) the truth table of a two-input NAND gate.

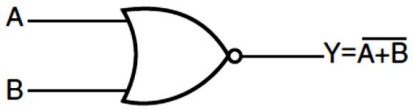
5. NOR Gate:

NOR stands for NOT OR. An OR gate followed by a NOT circuit makes it a NOR gate [Fig. 9(a)]. The truth table of a NOR gate is obtained from the truth table of an OR gate by complementing the output entries. The output of a NOR gate is a logic '1' when all its inputs are logic '0'. For all other input combinations, the output is a logic '0'. The output of a two-input NOR gate is logically expressed as:

$$Y = \overline{(A + B)}$$



(a)



(b)

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

(c)

Fig. (9): (a) Two-input NOR implementation using an OR gate and a NOT circuit, (b) the circuit symbol of a two-input NOR gate and (c) the truth table of a two-input NOR gate.

In general, the Boolean expression for a NOR gate with more than two inputs can be written as:

$$Y = \overline{(A + B + C + D...)}$$

THE EXCLUSIVE-OR AND EXCLUSIVE-NOR:

Exclusive-OR and exclusive-NOR gates are formed by a combination of other gates already discussed. However, because of their fundamental importance in many applications, these gates are often treated as basic logic elements with their own unique symbols.

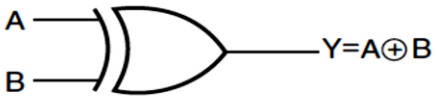
6. The Exclusive-OR Gate:

Standard symbol for an exclusive-OR (XOR for short) gate is shown in Fig. (10). The XOR gate has only two inputs.

For an exclusive-OR gate, output X is HIGH when input A is LOW and input B is HIGH, or when input A is HIGH and input B is LOW: X is LOW when A and B are both HIGH or both LOW.

The output of a two-input EX-OR gate is expressed as:

$$Y = (A \oplus B) = \overline{A}B + A\overline{B}$$



(a)

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

(b)

Fig. (10): X-OR gate.

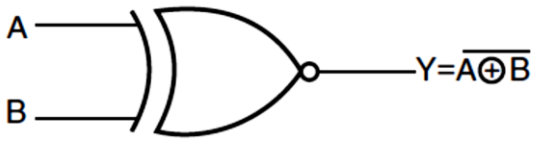
7. The Exclusive-NOR Gate:

Standard symbols for an exclusive-NOR (XNOR) gate are shown in Fig.(11). Like the XOR gate, an XNOR has only two inputs. The bubble on the output of the XNOR symbol indicates that its output is opposite that of the XOR gate. When the two input logic levels are opposite, the output of the exclusive-NOR gate is LOW. The operation can be stated as follows (A and B are inputs, X is the output):

For an exclusive-NOR gate, output X is LOW when input A is LOW and input B is HIGH, or when A is HIGH and B is LOW; X is HIGH when A and B are both HIGH or both LOW.

The truth table of an EX-NOR gate is obtained from the truth table of an EX-OR gate by complementing the output entries. Logically,

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



(a)

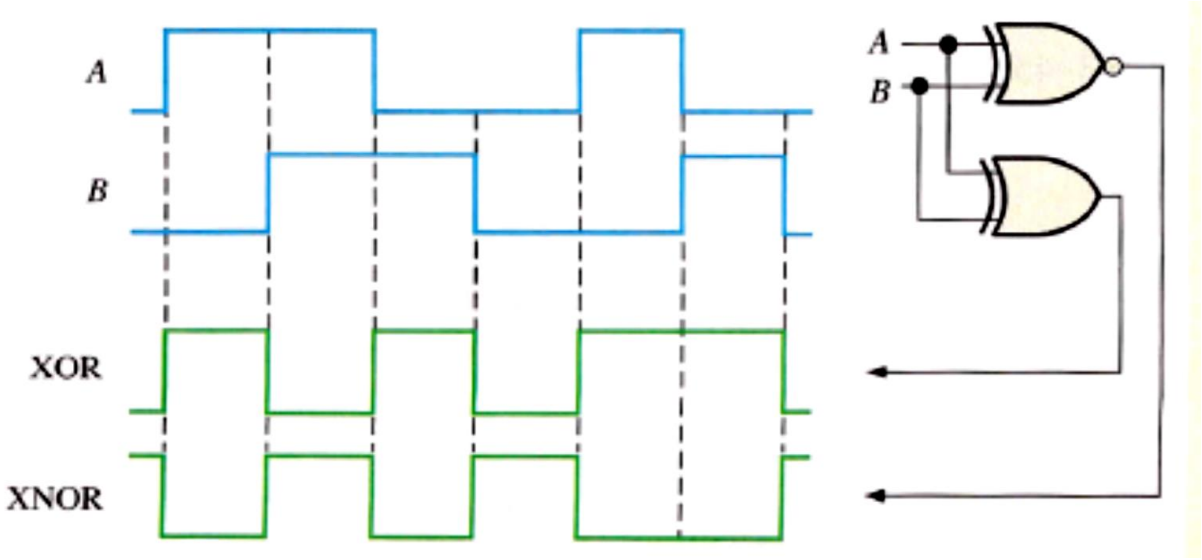
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

(b)

Fig. (11): XNOR gate.

Example:

Determine the output waveforms for the XOR gate and for the XNOR gate, given the input waveforms, A and B, in Figure below:



Post-Test

1. From a truth table, if Y=0 only when A=B=0, identify the gate.
2. If Y=1 when A=B=0 and A=B=1, identify the gate.
3. If Y=1 only when A=B=0, identify the gate.
4. If Y=1 only when A=B=1, identify the gate.

Digital Circuits – Integrated Assessment Section

Unit 3 – Boolean Algebra and Logic Simplification

Pre-Test

1. Write the output equation of the logic circuit shown in the original source figure.
2. Write the truth table for the Boolean function given in the source package: Y = ABC + ABC + ABC.
3. Draw the equivalent logic circuit for: Y = ABC + ABC.
4. Simplify using Boolean algebra: Y = AB + A(A + C) + B(A + C).
5. Draw the logic circuit equivalent to the simplified expression in Question 4.

Central Ideas

First: Boolean algebraic relations and de Morcan's theories.
Second: Simplifying logical functions using the laws and theorems of Boolean algebra.
Third: Finding a table of truth for circles using different gates.
Fourth: Writing the logical equation from the reality table - either using the product of the sum or the sum of the product.

D. Objectives of the lecture

After studying this lecture, the student will be able to:

- Learn how to use the laws and theorems of Boolean algebra in simplifying different logical functions.

- Deduces output equations for different logical circuits

- Learns how to implement different gates using one type of gateway (NOR or NAND)

Lecture Three

BOOLEAN ALGEBRA and Logic Simplification

BOOLEAN Operations and Expressions:

Variable, complement, and literal are terms used in Boolean algebra. A variable is a symbol used to represent a logical quantity. Any single variable can have a **1** or a **0** value. The complement is the inverse of a variable and is indicated by a bar over variable (overbar). For example, the complement of the variable **A** is **A.** If **A = 1,** then **A = 0.** If **A = 0**, then **A = 1**. The complement of the variable A is read as "not **A**" or "**A** bar."

Sometimes a prime symbol rather than an overbar is used to denote the complement of a variable; for example, B' indicates the complement of B.

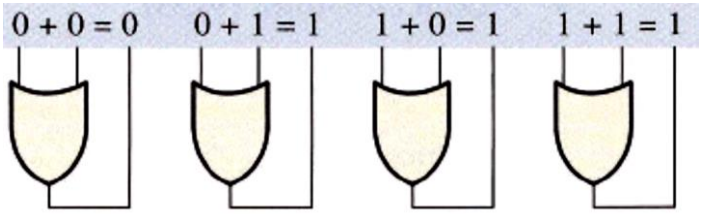


Fig. (1): OR Gate.

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

Boolean Addition:

sum term is a sum of literals. In logic circuits, a sum term is produced by an OR operation with no AND operations involved. Some examples of sum terms are

A + B

A + $\overline{B}$,

A +B + $\overline{C}$, and

$\overline{A}$ + B + C + $\overline{D}$.

EX// Determine the values of A, B, C, and D that make the sum term:

A + $\overline{B}$ + C + $\overline{D}$ equal to 0.

For the solution must the variables equal (0) therefore,

A=0, B=1 so that $\overline{B}$ =0, C=0 and D=1 so that $\overline{D}$=0

A+$\overline{B}$+C+$\overline{D}$= 0+$\overline{1}$+0+$\overline{1}$ = 0+ 0+0+0= 0

Boolean Multiplication:

In Boolean algebra, a product term is the product of literals. In logic circuits, a product term is produced by an AND operation with no OR operations involved. Some examples of product terms are:

AB, A$\overline{B}$, ABC, and $\overline{A}$BC$\overline{D}$.

A product term is equal to **1** only if each of the literals in the term is **1**. A product term is equal to **0** when one or more of the literals are **0**.

EX// Determine the values of A, B, C, and D that make the product term A$\overline{B}$C$\overline{D}$ equal to 1?

For the solution to be 1, therefore A=1,B=0 so that $\overline{B}$=1,C=1,and D=0 so that $\overline{D}$=1.

A$\overline{B}$C$\overline{D}$=1.$\overline{0}$.1.$\overline{0}$ =1.1.1.1=1

Laws and Rules of Boolean algebra:

Laws of Boolean algebra:

The basic laws of Boolean algebra-the commutative laws for addition and multiplication, the associative laws for addition and multiplication, and the distributive law-are the same as in ordinary algebra.

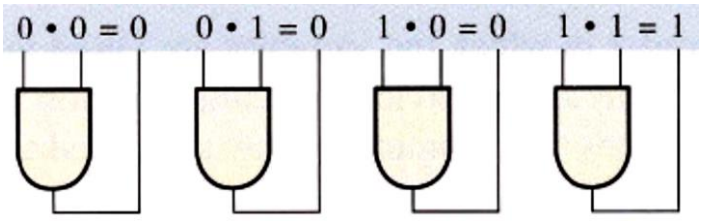


Fig. (2): And Gate.

Commutative Laws:

The commutative law of addition for two variables is written as:

$$A+B = B+A$$

This law states that the order in which the variables are ORed makes no difference. Remember, in Boolean algebra as applied to logic circuits, addition and the OR operation are the same. **(The symbol ≡ means "equivalent to.")**.

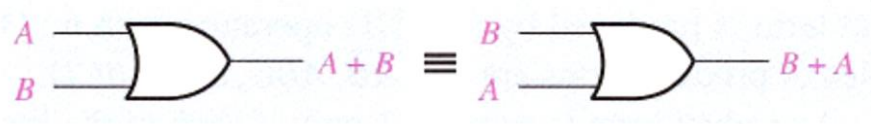


Fig. (3): Application of commutative law of addition.

The commutative law of multiplication for two variables is:

$$A.B = B.A$$

This law states that the order in which the variables are ANDed makes no difference. Fig.(4), il1ustrates this law as applied to the AND gate.

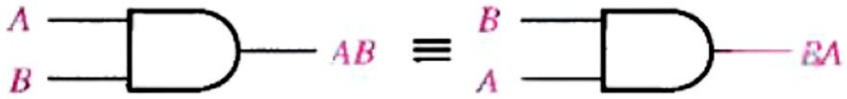


Fig. (4): Application of commutative law of multiplication.

Associative Laws:

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

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

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

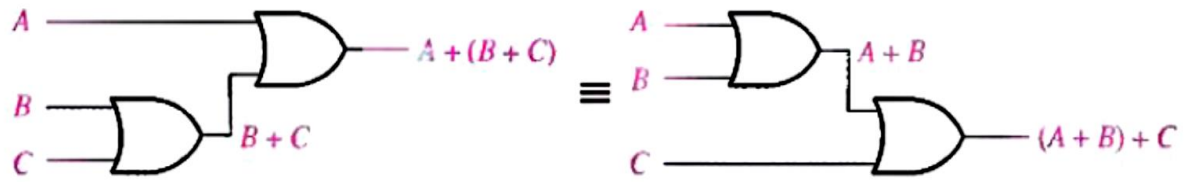


Fig. (5): Application of associative law of addition.

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

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

This law states that it makes no difference in what order the variables are grouped when ANDing more than two variables. Fig.(6) illustrates this law as applied to 2 -input AND gates.

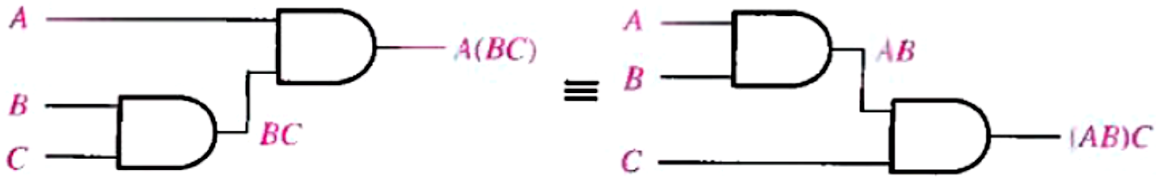


Fig. (6): Application of associative law of multiplication.

Distributive Law:

The distributive law is written for three variables as follows:

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

This law states that ORing two or more variables and then ANDing the result with a single variable is equivalent to ANDing the single variable with each of the two or more variables and then ORing the products. The distributive law also expresses the process of factoring in which the common variable A is factored out of the product terms, for example,

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

Fig.(7) illustrates the distributive law in terms of gates.

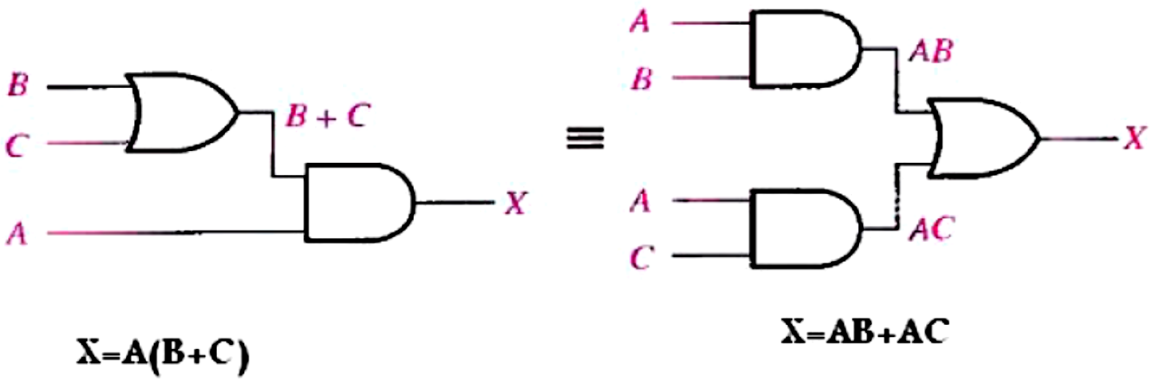


Fig. (7): Application of distributive law.

Rules of Boolean Algebra:

Table 1 lists 12 basic rules that are useful in manipulating and simplifying Boolean expressions. Rules 1 through 9 will be viewed in terms of their application to logic gates. Rules 10 through 12 will be derived in terms of the simpler rules and the laws previously discussed.

Table 1 Basic rules of Boolean algebra

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

1. **A+0=A**

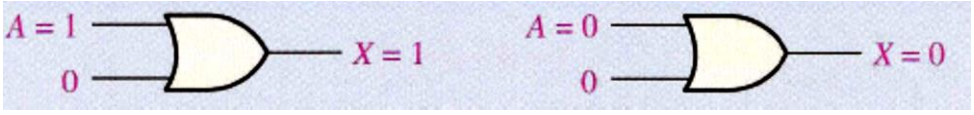


Fig. (8)

X = A+0 = A

2. **A+1=1**

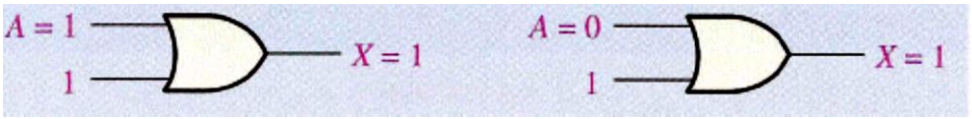


Fig. (9)

X = A+1 = 1

3. **A.0=0**

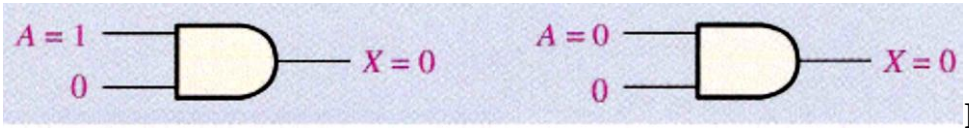


Fig. (10)

X = A.0 = 0

4. **A.1=A**

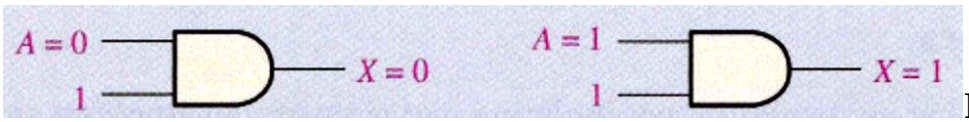


Fig. (11)

X = A.1 = A

5. **A+A=A**

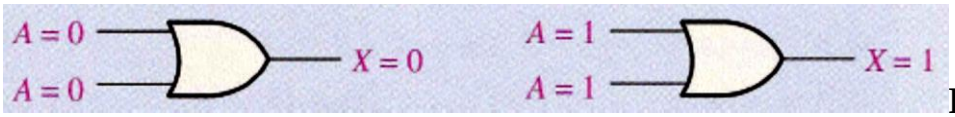


Fig. (12)

X = A+A = A

6. **A+$\overline{A}$=1**

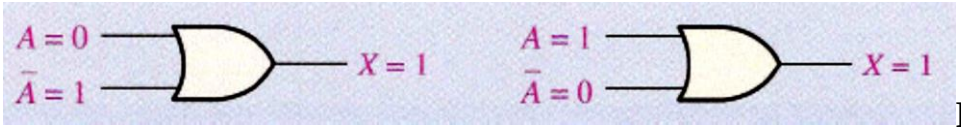


Fig. (13)

X = A+$\overline{A}$ = 1

7. **A. A=A**

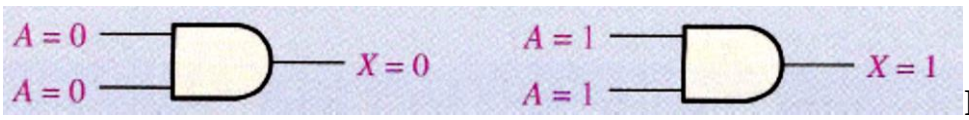


Fig. (14)

X = A. A = A

8. $A.\overline{A}=0$

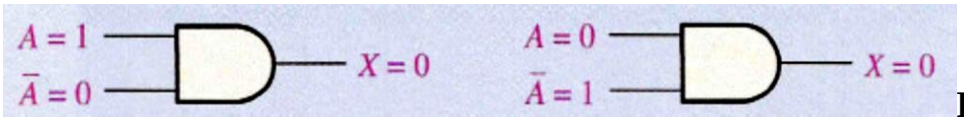


Fig. (15)

9. $\overline{\overline{A}}=A$

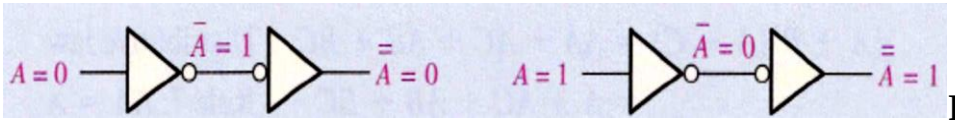


Fig. (16)

$X=\overline{\overline{A}}=A$

10. A+AB=A

A+AB = A (1+B)	**Factoring (distributive law)**
= A . 1	**Rule 2: (1 + B) = 1**
= A	**Rule 4: A . 1 = A**

The proof is shown in Table 2, which shows the truth table and the resulting logic circuit simplification:

A	B	AB	A + AB
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

equal

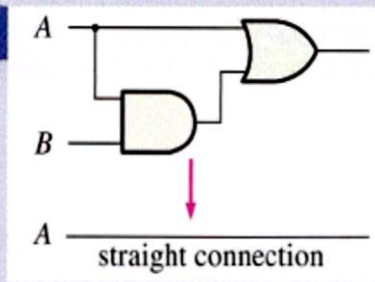


Fig. (17)

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

This rule can be proved as follows:

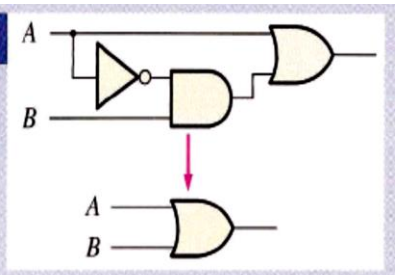
$A + \overline{A}B = (A + AB) + \overline{A}B$	**Rule 10: $A = A + AB$**
$= (AA + AB) + \overline{A}B$	**Rule 7: $A = AA$**
$=AA + AB + A\overline{A} + \overline{A}B$	**Rule 8: adding $A\overline{A} = 0$**
$= (A + \overline{A})(A + B)$	**Factoring**
$= 1. (A + B)$	**Rule 6: $A + \overline{A} = 1$**
$=A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

The proof is shown in Table 3, which shows the truth table and the resulting logic circuit simplification.

Table 3

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

equal



12. $(A + B)(A + C) = A + BC$

This rule can be proved as follows:

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

= A. 1 + BC **Rule 2: 1 + B = 1**

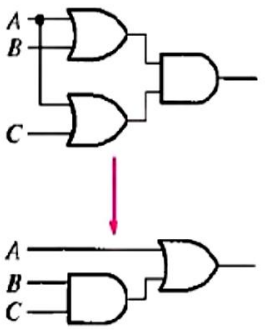
= A + BC **Rule 4: A . 1 = A**

The proof is shown in Table 4, which shows the truth table and the resulting logic circuit simplification.

Table 4

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

equal



DEMORGAN'S Theorems:

DeMorgan, a mathematician who knew Boole, proposed two theorems that are an important part of Boolean algebra. In practical terms. DeMorgan's theorems provide mathematical verification of the equivalency of the NAND and negative-OR gates and the equivalency of the NOR and negative-AND gates.

One of DeMorgan's theorems is stated as follows:

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

Stated another way,

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

The formula for expressing this theorem for two variables is:

$$\overline{XY} = \overline{X} + \overline{Y}$$

DeMorgan's second theorem is stated as follows:

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

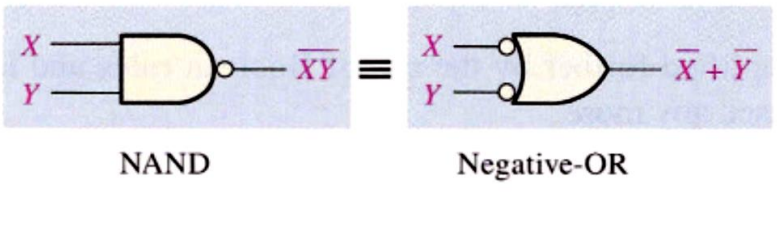
Stated another way,

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

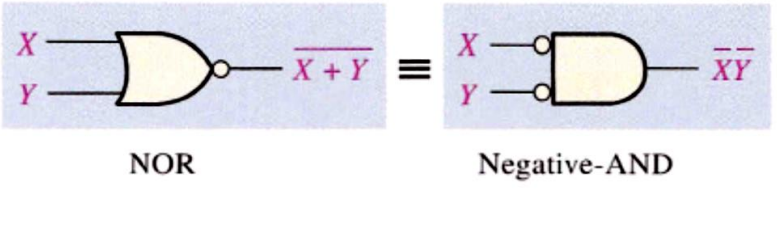
The formula for expressing this theorem for two variables is:

$$\overline{X + Y} = \overline{X}\,\overline{Y}$$

Fig.(18) shows the gate equivalencies and truth tables for the two equations above.



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



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

Fig. (18): Gate equivalencies and the corresponding truth tables that illustrate DeMorgan's theorems.

As stated, DeMorgan's theorems also apply to expressions in which there are more than two variables. The following examples illustrate the application of DeMorgan's theorems to 3-variable and 4-variable expressions.

EX// Apply DeMorgan's theorems to the expressions $\overline{XYZ}$ and $\overline{X + Y + Z}$.

$$\overline{XYZ} = \overline{X} + \overline{Y} + \overline{Z}$$
$$\overline{X + y + Z} = \overline{X}\,\overline{Y}\,\overline{Z}$$

Applying DeMorgan's Theorems:

The following procedure illustrates the application of DeMorgan's theorems and Boolean algebra to the specific expression:

$$\overline{\overline{A+B\overline{C}}+D(\overline{E+\overline{F}})}$$

Step 1. Identify the terms to which you can apply DeMorgan's theorems, and think of each term as a single variable.

Let:

$\overline{A+B\overline{C}} = X$ and $D(\overline{E+\overline{F}}) = Y$

Step 2. Since $\overline{X + Y} = \overline{X}\,\overline{Y}$,

$$\overline{\overline{A+B\overline{C}}+D(\overline{E+\overline{F}})} = \overline{\overline{A+B\overline{C}}}\;\overline{D(\overline{E+\overline{F}})}$$

Step 3. Use rule 9 ($A = \overline{\overline{A}}$) to cancel the double bars over the left term (this is not part of DeMorgan's theorem).

$$\overline{\overline{A+B\overline{C}}}\;\overline{D(\overline{E+\overline{F}})} = (A+B\overline{C})\,\overline{D(\overline{E+\overline{F}})}$$

Step 4. Applying DeMorgan's theorem to the second term**,**

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

Step 5. Use rule 9 ($A = \overline{\overline{A}}$) to cancel the double bars over the $E + \overline{F}$ part of the term.

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

EX// Apply DeMorgan's theorems to each of the following expressions:

(a) $\overline{(A + B + C)\,D}$ **(b)** $\overline{ABC + DEF}$ **(c)** $\overline{A\overline{B} + \overline{C}D + EF}$

SOL//

Post-Test

1. Write the truth table for: Y = ABC + ABC + ABC + ABC.
2. Write the truth table for: Y=(A+B+C)(A+B+C)(A+B+C).
3. Simplify using Boolean algebra: Y = ABCD + ABCD + ABCD + ABCD + AB + BD.
4. Simplify using Boolean algebra: Y = ABCD + ABCD + ABC + BCD + AC + BC.
5. Write the output equation of the logic circuit represented by the truth-table figure in the original source.

Digital Circuits – Integrated Assessment Section

Unit 4 – Boolean Analysis / Karnaugh-Map Preparation

Pre-Test

1. Simplify the given Boolean expression using a Karnaugh map.
2. Draw the logic circuit equivalent to the simplified expression.
3. Simplify the Boolean expression shown in the source using a Karnaugh map.
4. Simplify the next Boolean expression shown in the source using a Karnaugh map.
5. Simplify the diagram-based Boolean expression shown in the source using a Karnaugh map.

(a) $\overline{(A + B + C)\,D}$ **(b)** $\overline{ABC + DEF}$ **(c)** $\overline{A\overline{B} + \overline{C}D + EF}$

(a) Let $A + B + C = X$ and $D = Y$. The expression $\overline{(A + B + C)D}$ is of the form $\overline{XY} = \overline{X} + \overline{Y}$ and can be rewritten as

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

Next, apply DeMorgan's theorem to the term $\overline{A + B + C}$.

$$\overline{A + B + C} + \overline{D} = \overline{A}\,\overline{B}\,\overline{C} + \overline{D}$$

(b) Let $ABC = X$ and $DEF = Y$. The expression $\overline{ABC + DEF}$ is of the form $\overline{X + Y} = \overline{X}\,\overline{Y}$ and can be rewritten as

$$\overline{ABC + DEF} = (\overline{ABC})(\overline{DEF})$$

Next, apply DeMorgan's theorem to each of the terms $\overline{ABC}$ and $\overline{DEF}$.

$$(\overline{ABC})(\overline{DEF}) = (\overline{A} + \overline{B} + \overline{C})(\overline{D} + \overline{E} + \overline{F})$$

(c) Let $A\overline{B} = X$, $\overline{C}D = Y$, and $EF = Z$. The expression $\overline{A\overline{B} + \overline{C}D + EF}$ is of the form $\overline{X + Y + Z} = \overline{X}\,\overline{Y}\,\overline{Z}$ and can be rewritten as

$$\overline{A\overline{B} + \overline{C}D + EF} = (\overline{A\overline{B}})(\overline{\overline{C}D})(\overline{EF})$$

Next, apply DeMorgan's theorem to each of the terms $\overline{A\overline{B}}$, $\overline{\overline{C}D}$, and $\overline{EF}$.

$$(\overline{A\overline{B}})(\overline{\overline{C}D})(\overline{EF}) = (\overline{A} + B)(C + \overline{D})(\overline{E} + \overline{F})$$

Lecture Four

Boolean Analysis of Logic Circuits

Boolean algebra provides a concise way to express the operation of a logic circuit formed by a combination of logic gates so that the output can be determined for various combinations of input values.

Boolean Expression for a Logic Circuit:

To derive the Boolean expression for a given logic circuit, begin at the leftmost inputs and work toward the final output, writing the expression for each gate. For the example circuit in Fig.(1), the Boolean expression is determined as follows:

1. The expression for the left-most AND gate with inputs C and D is CD.
2. The output of the left-most AND gate is one of the inputs to the OR gate and B is the other input. Therefore, the expression for the OR gate is B + CD.
3. The output of the OR gate is one of the inputs to the right-most AND gate and A is the other input. Therefore, the expression for this AND gate is A(B + CD), which is the final output expression for the entire circuit.

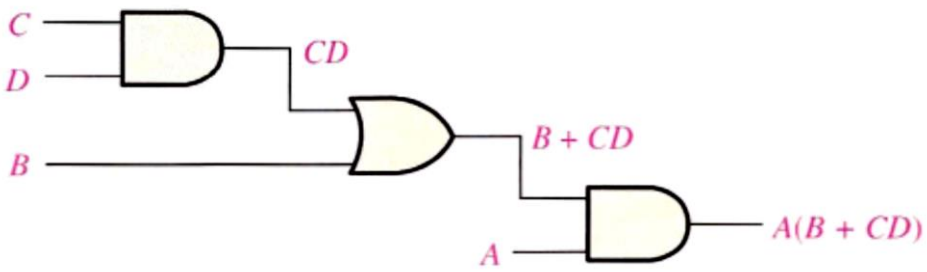


Fig. (1): A logic circuit showing the development of the Boolean expression for the output.

Constructing a Truth Table for a Logic Circuit:

Once the Boolean expression for a given logic circuit has been determined, a truth table that shows the output for all possible values of the input variables can be developed. The procedure requires that you evaluate the Boolean expression for all possible combinations of values for the input variables. In the case of the circuit in Fig.(2), there are four input variables (A, B, C, and D) and therefore sixteen (2^4 = 16) combinations of values are possible.

Evaluating the expression:

To evaluate the expression **(A(B+CD))**,first find the values of the variables that make the expression equal to 1, using the rules for Boolean addition and multiplication. In this case the expression equals 1 only if **A=1** and **B+CD=1**because **A(B+CD)=1.1=1**

Now determine when the **B+CD** term equals **1**. The term **B+CD =1** if either **B=1**or **CD=1** or if the both **B** and **CD** equals **1**because

B+CD = 1+0=1

B+CD = 0+1=1

B+CD = 1+1=1

The term **CD =1** only if **C=1**and **D=1.**

The summarize, the expression **A(B+CD)=1**when **A =1** and **B =1** regardless of the values of **C** and **D** or when **A=1** and **C = 1**and **D =1** regardless of the value of **B.** the expression **A(B+CD)=0** for all other value combinations of variables.

Putting the Results in Truth Table format

The first step is to list the sixteen input variable combinations of 1s and 0s in a binary sequence as shown in Table 4-5. Next, place a 1 in the output column for each combination of input variables that was determined in the evaluation. Finally,

place a 0 in the output column for all other combinations of input variables. These results are shown in the truth table in Table 1.

Table 1

INPUTS				OUTPUT
A	*B*	*C*	*D*	*A(B + CD)*
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Simplification Using Boolean Algebra:

A simplified Boolean expression uses the fewest gates possible to implement a given expression.

EX//

Using Boolean algebra techniques, simplify this expression:

AB + A(B + C) + B(B + C)

SOL//

Step 1: Apply the distributive law to the second and third terms in the expression, as follows:

AB + AB + AC + BB + BC

Step 2: Apply rule 7 **(BB = B)** to the fourth term

AB+AB+AC+B+BC

Step 3: Apply rule 5 **(AB + AB = AB)** to the first two terms.

AB + AC + B + BC

Step 4: Apply rule 10 **(B + BC = B)** to the last two terms.

AB+AC+B

Step 5: Apply rule 10 **(AB + B = B)** to the first and third terms.

B+AC

At this point the expression is simplified as much as possible.

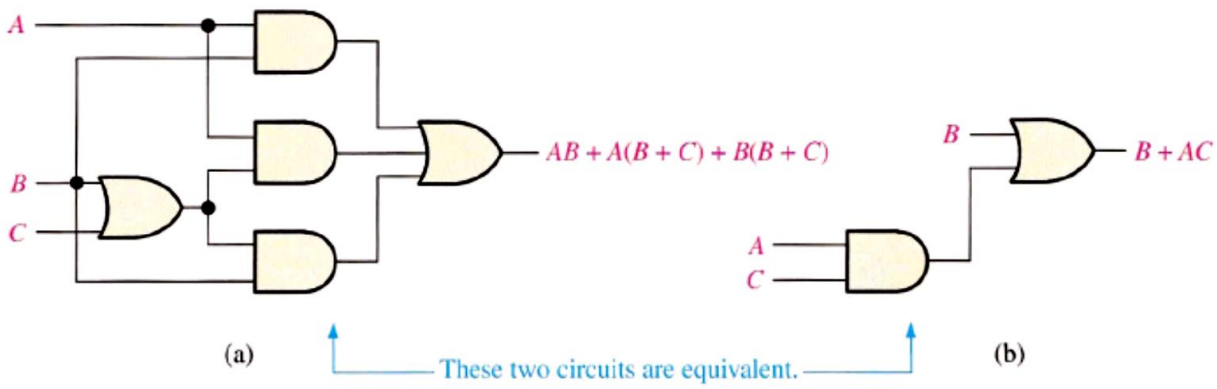


Fig. (2).

EX// Simplify the following Boolean expression:

$$[A\bar{B}(C+BD)+\bar{A}\bar{B}]C$$

Note that brackets and parentheses mean the same thing: the term inside is multiplied (ANDed) with the term outside.

SOL//

Step1: Apply the distributive law to the terms within the brackets.

$$(A\bar{B}C+AB\bar{B}D+\bar{A}\bar{B})C$$

Step2: apply rule 8 (**BB = 0)** to the second term within the parentheses

$$(A\bar{B}C+A.0.D+\bar{A}\bar{B})C$$

Step3: apply rule3 **(A.0.D =0)**

$$(A\bar{B}C+0+\bar{A}\bar{B})C$$

Step4: apply rule1 (drop the 0)

$$(A\bar{B}C+\bar{A}\bar{B})C$$

Step5: apply the distributive law

$$A\bar{B}CC+\bar{A}\bar{B}C$$

Step6: apply rule 7(**CC=C)**

$$A\bar{B}C+\bar{A}\bar{B}C$$

Step7: Factor out $\bar{B}C$

$$\bar{B}C(A+\bar{A})$$

Step8: apply rule 6 **($A+\bar{A}$ =1)**

$$\bar{B}C$$

Standard Forms of Boolean Expressions:

All Boolean expressions, regardless of their form, can be converted into either of two standard forms: the sum-of-products form or the product-of sums form. Standardization makes the evaluation, simplification, and implementation of Boolean expressions much more systematic and easier.

A binary variable may appear either in its normal form (x) or in its complement form ($\bar{x}$). Now consider two binary variables x and y combined with an **AND** operation. Since each variable may appear in either form, there are four possible combinations: $\bar{x}$ y , x $\bar{y}$, $\bar{x}$ $\bar{y}$, and x y. Each of these four AND terms is called a minterm, or a standard product. In a similar manner, **n** variables can be combined to form 2^n minterms. The 2^n different minterms may be determined by a method similar to the one shown in Table **2** for three variables.

Each minterm is obtained from an AND term of the n variables, with each variable being primed if the corresponding bit of the binary number is a = 0 and unprimed if a = 1. A symbol for each minterm is also shown in the table and is of the form m_j, where the subscript **j** denotes the decimal equivalent of the binary number of the minterm designated.

In a similar fashion, n variables forming an OR term, with each variable being primed or unprimed, provide 2^n possible combinations, called maxterms, or standard sums. The eight maxterms for three variables, together with their symbolic designations, are listed in Table 2. Any 2^n maxterms for **n** variables may be determined similarly. It is important to note that (1) each maxterm is obtained from an OR term of the n variables, with each variable being unprimed if the corresponding bit is a = 0 and primed if a = 1, and (2) each **maxterm is the complement of its corresponding minterm and vice versa.**

A Boolean function can be expressed algebraically from a given truth table by forming a minterm for each combination of the variables that produces a 1 in the function and then taking the OR of all those terms.

For example, the function f1 in Table 2 is determined by expressing the combinations 001, 100, and 111 as $\bar{x}\,\bar{y}\,z$, $x\,\bar{y}\,\bar{z}$, and xyz, respectively. Since each one of these minterms results in **f1 = 1**, we have

$f1 = \bar{x}\,\bar{y}\,z + x\,\bar{y}\,\bar{z} + xyz = m1 + m4 + m7$

Table 2, Minterms and Maxterms for Three Binary Variables

X	Y	Z	Minterms		Maxterms	
			Term	Designation	Term	Designation
0	0	0	$\bar{X}\,\bar{Y}\,\bar{Z}$	m_0	$X+Y+Z$	M_0
0	0	1	$\bar{X}\,\bar{Y}\,Z$	m_1	$X+Y+\bar{Z}$	M_1
0	1	0	$\bar{X}\,Y\,\bar{Z}$	m_2	$X+\bar{Y}+Z$	M_2
0	1	1	$\bar{X}\,Y\,Z$	m_3	$X+\bar{Y}+\bar{Z}$	M_3
1	0	0	$X\,\bar{Y}\,\bar{Z}$	m_4	$\bar{X}+Y+Z$	M_4
1	0	1	$X\,\bar{Y}\,Z$	m_5	$\bar{X}+Y+\bar{Z}$	M_5
1	1	0	$X\,Y\,\bar{Z}$	m_6	$\bar{X}+\bar{Y}+Z$	M_6
1	1	1	$X\,Y\,Z$	m_7	$\bar{X}+\bar{Y}+\bar{Z}$	M_7

Sum of Minterms:

The minterms whose sum defines the Boolean function are those which give the 1's of the function in a truth table.

EX// Express the Boolean function **$F = A + \bar{B}C$** as a sum of minterms. The function has three variables: A, B, and C.

The first term **A** is missing two variables; therefore,

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

This function is still missing one variable, so

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

$$= ABC + AB\overline{C} + A\overline{B}C + A\overline{B}\,\overline{C}$$

The second term $\overline{B}C$ is missing one variable; hence,

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

Combining all terms, we have

$$F = A + \overline{B}C$$

$$= ABC + AB\overline{C} + A\overline{B}C + A\overline{B}\,\overline{C} + A\overline{B}C + \overline{A}\overline{B}C$$

But $A\overline{B}C$ appears twice, and according to theorem 1 (x + x = x), it is possible to remove one of those occurrences. Rearranging the minterms in ascending order, we finally obtain

$$F = \overline{A}\overline{B}C + A\overline{B}\,\overline{C} + A\overline{B}C + AB\overline{C} + ABC$$

$$= m1 + m4 + m5 + m6 + m7$$

When a Boolean function is in its sum-of-minterms form, it is sometimes convenient to express the function in the following brief notation:

$$F(A, B, C) = \sum(1, 4, 5, 6, 7)$$

The summation symbol $\sum$ stands for the ORing of terms

An alternative procedure for deriving the minterms of a Boolean function is to obtain the truth table of the function directly from the algebraic expression and then read the minterms from the truth table.

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Product of Maxterms:

Each of the 2^{2n} functions of n binary variables can be also expressed as a product of maxterms.

To express a Boolean function as a product of maxterms, it must first be brought into a form of OR terms. This may be done by using the distributive law, **x + yz = (x + y)(x + z)**. Then any missing variable **x** in each OR term is ORed with **$x\bar{x}$** The procedure is clarified in the following example.

EX// Express the Boolean function F = xy + $\bar{x}$ z as a product of maxterms**.**

First, convert the function into OR terms by using the distributive law:

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

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

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

The function has three variables: x, y, and z. Each OR term is missing one variable; therefore,

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

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

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

Combining all the terms and removing those which appear more than once, we finally obtain

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

$$= M0 \quad M2 \quad M4 \quad M5$$

A convenient way to express this function is as follows:

$$F(x, y, z) = \Pi(0, 2, 4, 5)$$

The product symbol, **Π**, denotes the ANDing of maxterms; the numbers are the indices of the maxterms of the function

Conversion between Canonical Forms:

The complement of a function expressed as the sum of minterms equals the sum of minterms missing from the original function. This is because the original function is expressed by those minterms, which make the function equal to 1, whereas its complement is a 1 for those minterms for which the function is a 0. As an example, consider the function:

$$\mathbf{F(A, B, C) = \sum(1, 4, 5, 6, 7)}$$

This function has a complement that can be expressed as:

$$\mathbf{\overline{F}\ (A, B, C) = \sum(0, 2, 3) = m_0 + m_2 + m_3}$$

Now, if we take the complement of $\overline{F}$ by DeMorgan's theorem, we obtain F in a different form:

$$\mathbf{F = \overline{(m_0 + m_2 + m_3)} = \overline{m_0} \cdot \overline{m_2} \cdot \overline{m_3} = M_0M_2M_3 = \Pi(0, 2, 3)}$$

The last conversion follows from the definition of minterms and maxterms as shown in Table 2 . From the table, it is clear that the following relation holds:

$$\mathbf{\overline{m_j} = M_j}$$

That is, the **maxterm with subscript j is a complement of the minterm with the same subscript j and vice versa.**

1. **Rationale**

The laws of Boolean algebra are the basis for simplifying complex logical equations and circuits, but they become ineffective when the number of circuit inputs increases, so the Karnov map method is used.

This lecture is designed for the student to learn how to simplify different logical circles and equations using the Karnov map.

C. Central Ideas

First: Karnov map - Karnov map for two variables, Karnov map for three variables, Karnov map for four variables.

Second: How to transfer the table of realism to the map of Karnov.

Third: Simplifying logical functions and logical circuits using the Karnov map.

D. Objectives of the lecture

After studying this lecture, the student will be able to:

- **Learn how to simplify using Karnov map for two variables, Karnov map for three variables and Karnov map for four variables.**
- **Transfers the table of realism to the map of Karnov.**

Lecture Five

KARNAUGH MAPS

Karnaugh mapping is a method used to simplify a truth table using sum of products or product of sums along with simultaneous optimization of the output function. Karnaugh maps are the graphical equivalent of a truth table. In other words, Karnaugh maps are an easy way of designing and optimizing a circuit from a truth table.

Rules for K-Maps

1. Each cell with a 1 must be included in at least one group.

2. Try to form the largest possible groups.

3. Try to end up with as few groups as possible.

4. Groups may be in sizes that are powers of 2: $2^0 = 1$, $2^1 = 2$, $2^2 = 4$, $2^3 = 8$, $2^4 = 16$, ...

5. Groups may be square or rectangular only (including wraparound at the grid edges). No diagonals or zig-zags can be used to form a group.

6. The larger a group is, the more redundant inputs there are:

i. A group of 1 has no redundant inputs.

ii. A group of 2 has 1 redundant input.

iii. A group of 4 has 2 redundant inputs.

iv. A group of 8 has 3 redundant inputs.

v. A group of 16 has 4 redundant inputs

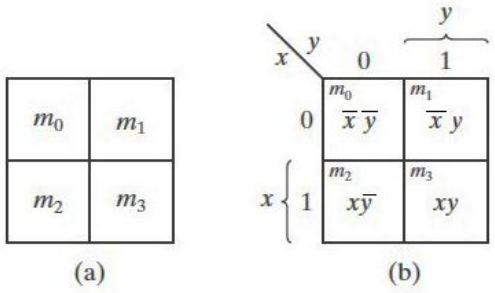
TWO-VARIABLE KARNAUGH MAP:

The two-variable map is shown in Fig. 1 (a). There are four minterms for two variables; hence, the map consists of four squares, one for each minterm. The map is redrawn in (b) to show the relationship between the squares and the two variables x and y . The 0 and 1 marked in each row and column designate the values of variables. Variable x appears primed in row 0 and unprimed in row 1. Similarly, y appears primed in column 0 and unprimed in column 1.

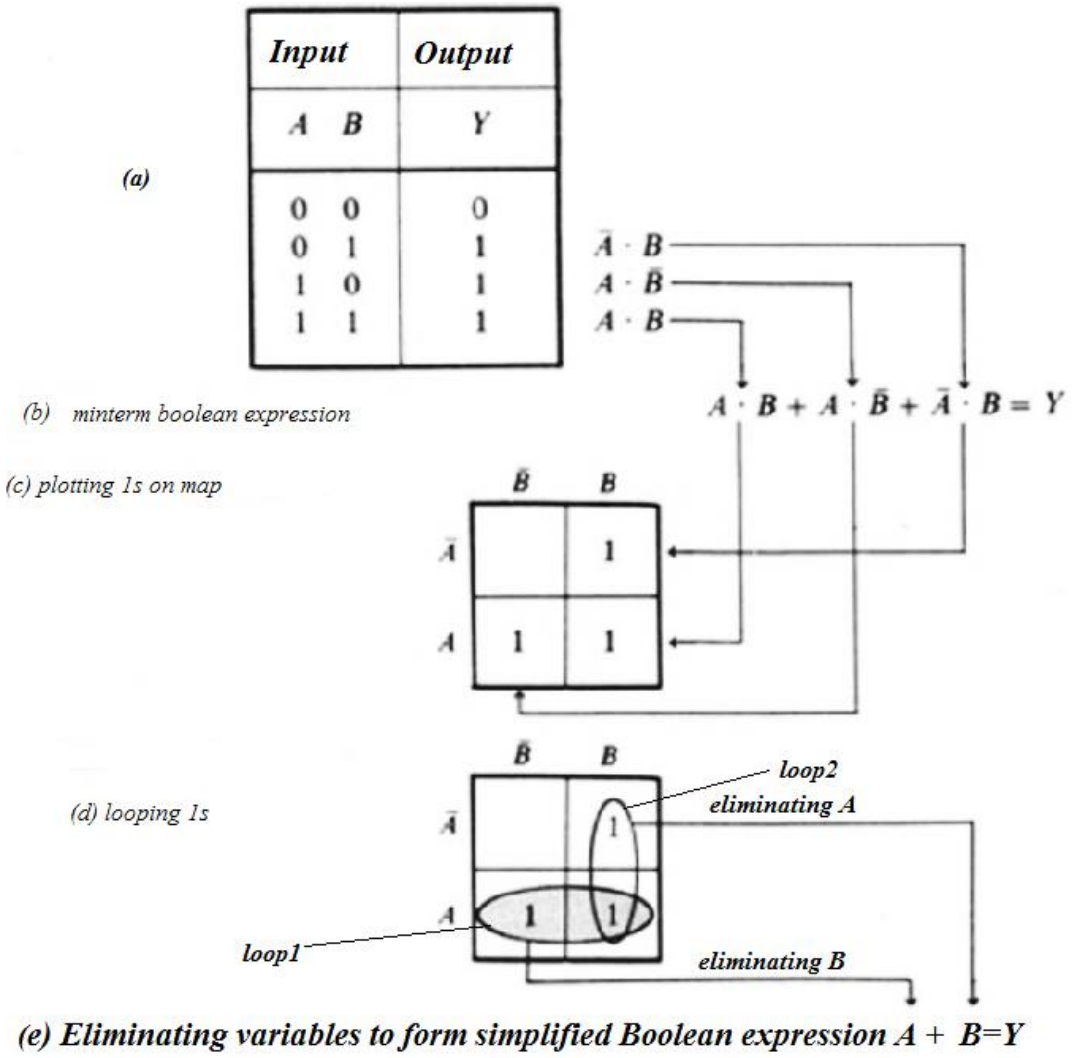
If we mark the squares whose minterms belong to a given function, the two-variable map becomes another useful way to represent any one of the **16** Boolean functions of two variables. As an example, the function xy is shown in Fig. 1 (a). Since xy is equal to $\mathbf{m_3}$ **,** a **1** is placed inside the square that belongs to m3. Similarly, the function **x + y** is represented in the map of Fig. 1 (b) by three squares marked with **1's**. These squares are found from the minterms of the function:

$$m1 + m2 + m3 = \bar{x}\,y + x\,\bar{y} + xy = x + y$$

A	B	f =1
0	0	0
0	1	1
1	0	1
1	1	1



Two-variable K-map.



(e) Eliminating variables to form simplified Boolean expression $A + B = Y$

Fig 2

The 3-Variable Karnaugh Map

The 3-variable Karnaugh map is an array of eight cells. as shown in Fig.(3). In this case, A, B, and C are used for the variables although other letters could be used.

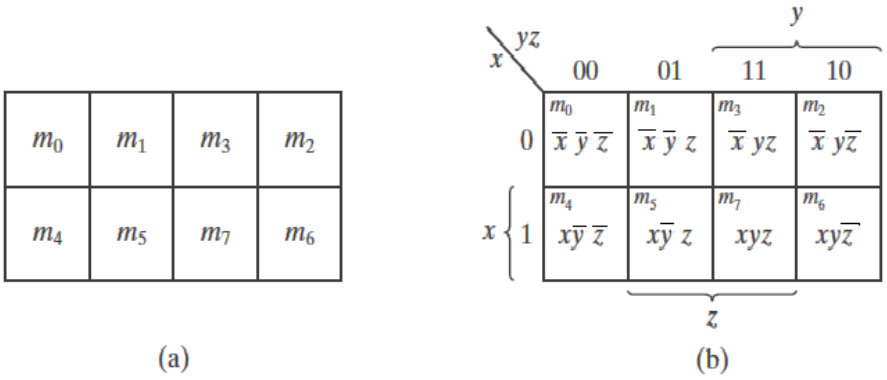


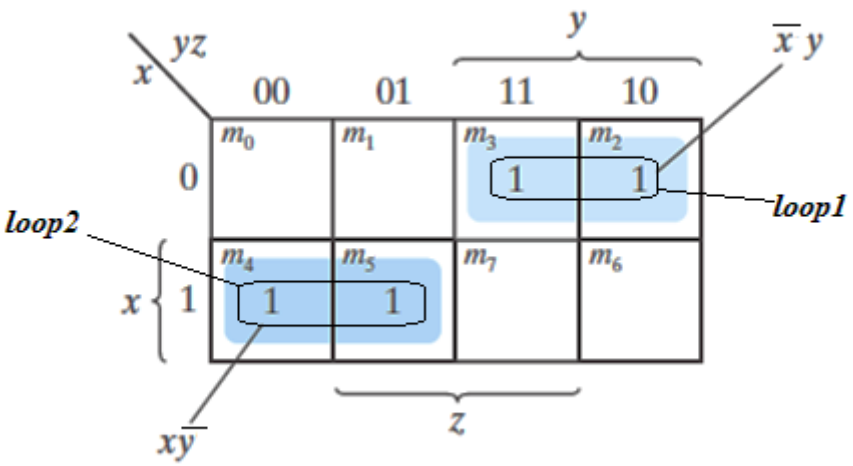
Fig3: three variable K-map

EX// Simplify the Boolean function

F (x, y, z) = ∑(2, 3, 4, 5)

loop1= $\bar{X}$ Y Z + $\bar{X}$ Y $\bar{Z}$= $\bar{x}$ y

loop2= x $\bar{y}$ $\bar{z}$ + x $\bar{y}$ z =x $\bar{y}$



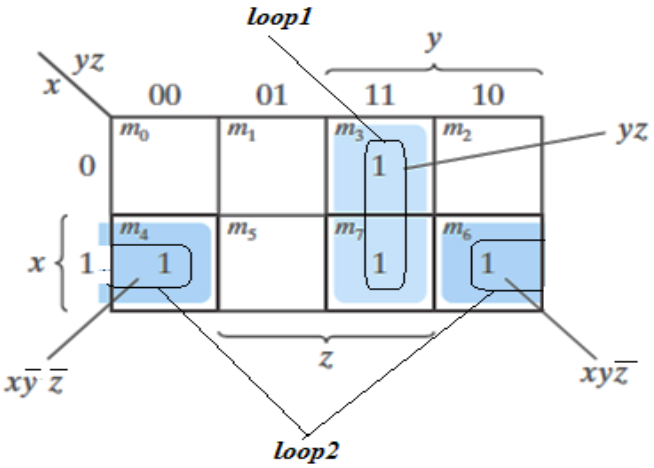
Map for Example , F (x, y, z) = ∑(2, 3, 4, 5) = $\bar{x}$ y + x $\bar{y}$

Ex// Simplify the Boolean function:

F (x, y, z) = ∑(3, 4, 6, 7)

loop1= $\bar{x}$ y z+ x y z = y z

loop2= x $\bar{y}$ $\bar{z}$+ x y $\bar{z}$ = x $\bar{z}$

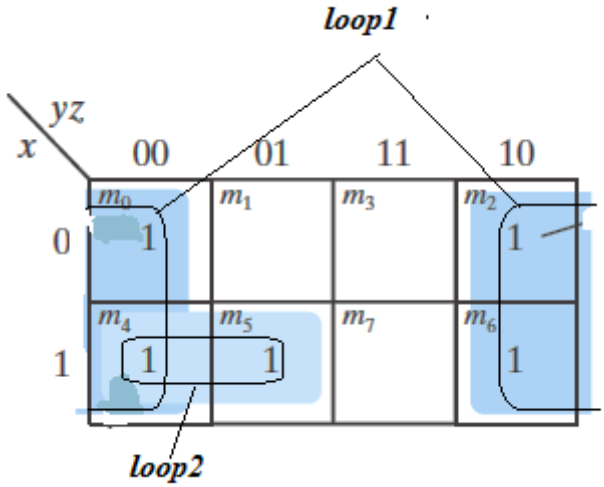


$$f = Y\,Z + X\,\overline{Z}$$

EX// **Simplify the Boolean function** $F(x, y, z) = \sum(0, 2, 4, 5, 6)$

loop1 $= \overline{x}\,\overline{y}\,\overline{z} + \overline{x}\,y\,\overline{z} + x\,\overline{y}\,\overline{z} + x\,y\,\overline{z} = \overline{z}$

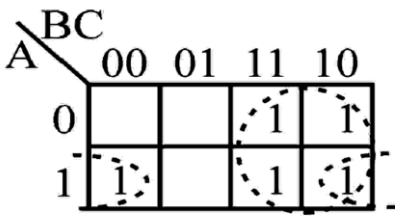
loop2 $= x\,\overline{y}\,\overline{z} + x\,\overline{y}\,z = x\,\overline{y}$



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

EX// Simplify by using K-map:

A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1



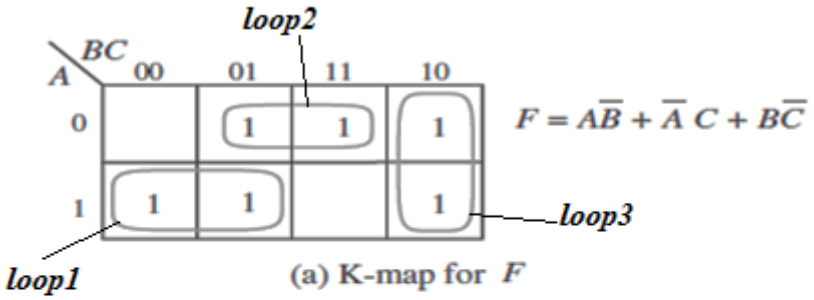
$Y = AC' + B$

EX// Simplify f =∑(1,2,3,4,5,6):

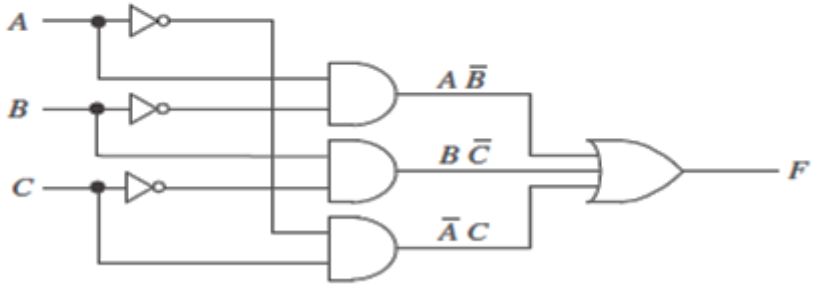
loop1 = $A\bar{B}C + A\bar{B}C = A\bar{B}$

loop2 = $\bar{A}\bar{B}C + A\bar{B}C = \bar{A}C$

loop3 = $\bar{A}B\bar{C} + AB\bar{C} = B\bar{C}$



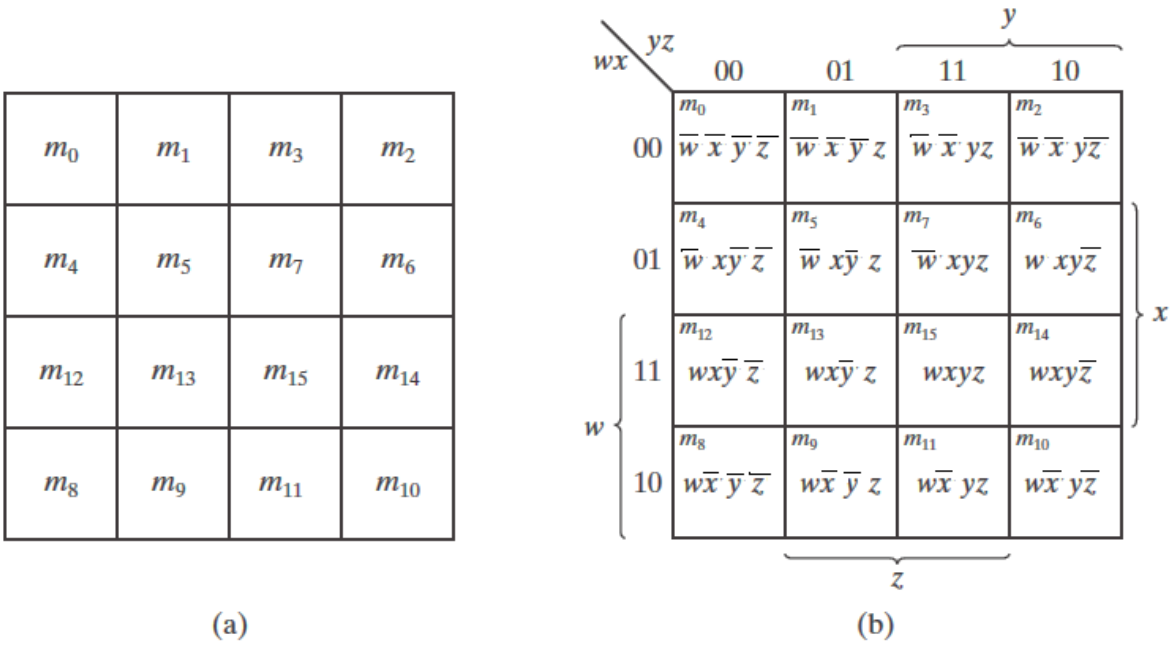
(a) K-map for F



(b) Logic Diagram for the output, F

FOUR-VARIABLE K-MAP

The map for Boolean functions of four binary variables (w, x, y, z) is shown in Fig. 8 .In Fig. 8(a) are listed the 16 minterms and the squares assigned to each. In Fig.8(b), the map is redrawn to show the relationship between the squares and the four variables.

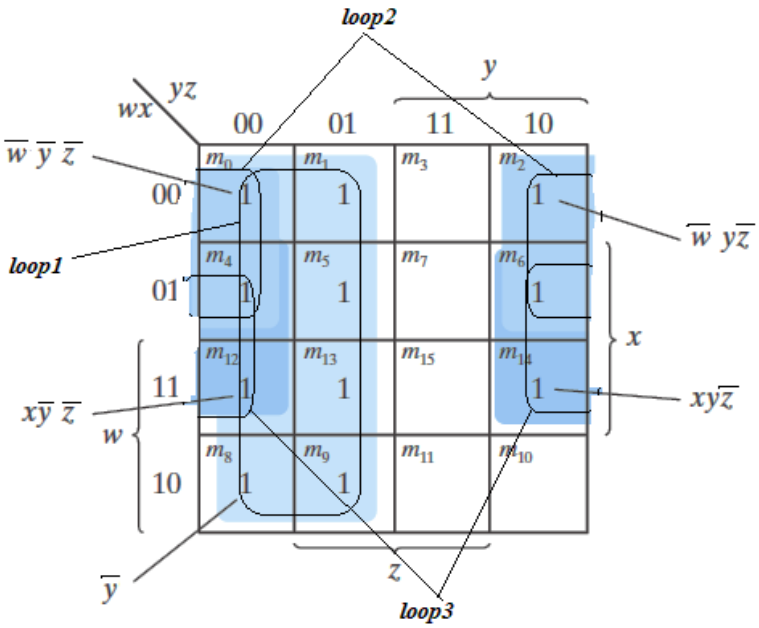


Fig(8) four variables

Ex //Simplify the Boolean function

F (w, x, y, z) = ∑(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)

w	x	y	z	F
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0



loop1= $\bar{w}\,\bar{x}\,\bar{y}\,z + \bar{w}\,x\,y\,\bar{z} + \bar{w}\,x\,\bar{y}\,\bar{z} + w\,x\,\bar{y}\,\bar{z} + w\,x\,\bar{y}\,z + \bar{w}\,\bar{x}\,y\,z + \bar{w}\,x\,\bar{y}\,z + w\,x\,y\,z = y$

loop2= $\bar{w}\,\bar{x}\,\bar{y}\,\bar{z} + \bar{w}\,x\,y\,\bar{z} + \bar{w}\,x\,\bar{y}\,z + w\,\bar{x}\,y\,z = w\,z$

loop3 = $\bar{w}\,\bar{x}\,y\,z + \bar{w}\,x\,y\,\bar{z} + \bar{w}\,x\,y\,z + w\,\bar{x}\,y\,z = x\,z$

f = $\bar{y} + \bar{w}\,z + \bar{x}\,z$

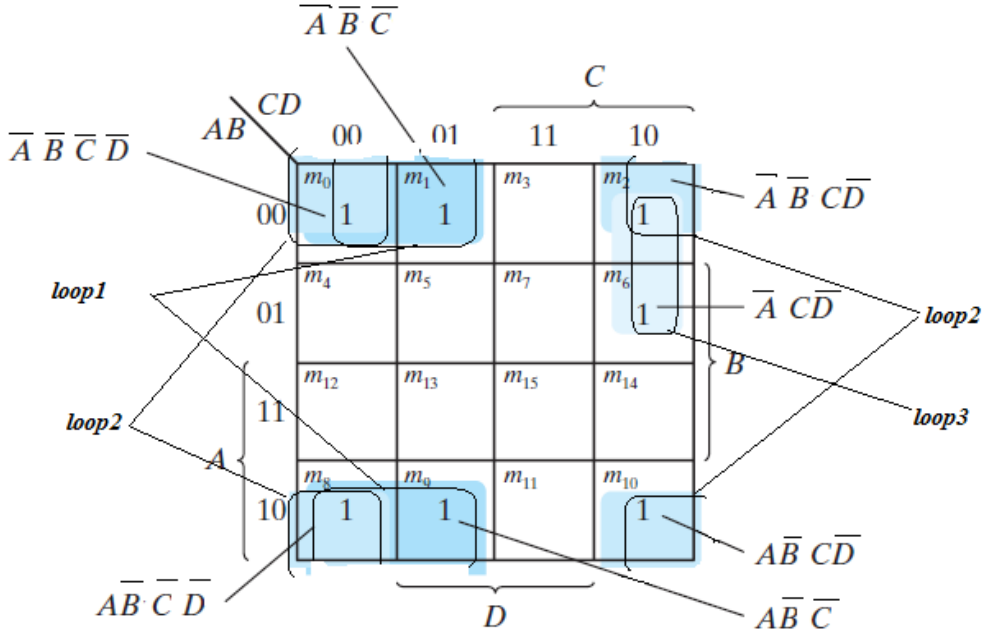
<u>Ex //</u> Simplify the Boolean function:

F = $\bar{A}\,\bar{B}\,\bar{C} + \bar{B}\,C\,\bar{D} + \bar{A}\,BC\bar{D} + A\,B\,\bar{C}$

f = $\bar{A}\,\bar{B}\,\bar{C}(D+\bar{D}) + \bar{B}\,C\,\bar{D}(\bar{A}+A) + \bar{A}\,B\,C\,\bar{D} + A\,\bar{B}\,\bar{C}\,(\bar{D}+D)$

f = $\bar{A}\,\bar{B}\,\bar{C}\,D + \bar{A}\,\bar{B}\,C\,\bar{D} + A\,\bar{B}\,C\,\bar{D} + \bar{A}\,B\,\bar{C}\,D + \bar{A}\,B\,\bar{C}\,\bar{D} + A\,\bar{B}\,\bar{C}\,D + A\,B\,C\,D$

w	x	y	z	F
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0



loop1=$\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$+ $\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$+$\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$ + $\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$ = $\overline{B}\,\overline{C}$

loop2= $\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$+ $\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$+$\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$ + $\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$= $\overline{B}\overline{D}$

loop3 = $\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$+ $\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$ =$\overline{A}\overline{C}\overline{D}$

f= $\overline{B}\,\overline{C}$+ $\overline{B}\,\overline{D}$ +$\overline{A}\,\overline{C}\,\overline{D}$

Don’t-care conditions:

In most applications, we simply don’t care what value is assumed by the function for the unspecified minterms. For this reason, it is customary to call the unspecified minterms of a function don’t-care conditions. These don’t-care conditions can be used on a map to provide further simplification of the Boolean expression.

A don’t-care minterm is a combination of variables whose logical value is not specified. Such a minterm cannot be marked with a 1 in the map, because it would require that the function always be a 1 for such a combination. Likewise, putting a 0 on the square requires the function to be 0. To distinguish the don’t-care condition from 1’s and 0’s, an X is used. Thus, an X inside a square in the map indicates that we don’t care whether the value of 0 or 1 is assigned to F for the particular minterm.

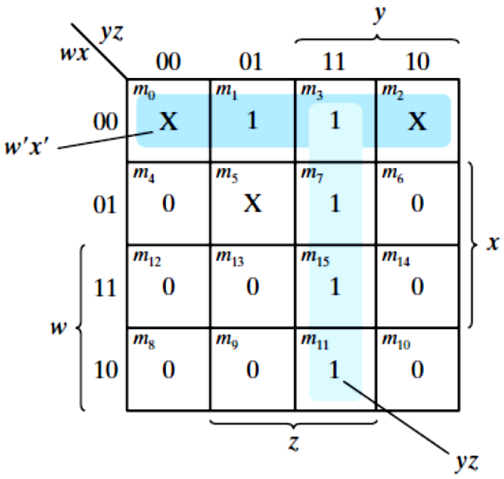
In choosing adjacent squares to simplify the function in a map, the don’t-care minterms may be assumed to be either 0 or 1. When simplifying the function, we can choose to include each don’t-care minterm with either the 1’s or the 0’s, depending on which combination gives the simplest expression.

Ex// Simplify the Boolean function

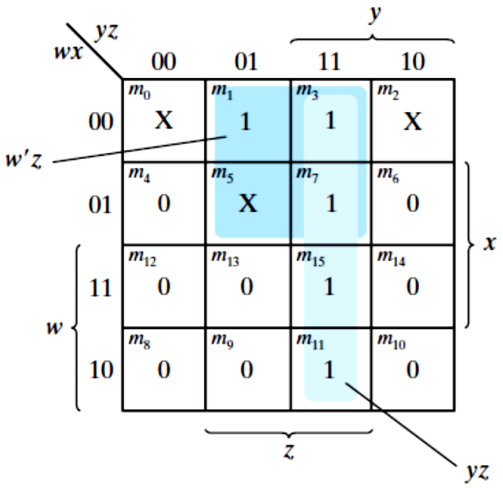
$F(w, x, y, z) = \sum(1, 3, 7, 11, 15)$

which has the don’t-care conditions

$d(w, x, y, z) = \sum(0, 2, 5)$



(a) $F = yz + w'x'$



(b) $F = yz + w'z$

Post-Test

1. Simplify the Boolean expressions shown in the source using a Karnaugh map.
2. Draw the equivalent logic circuit for the simplified result.
3. Complete the remaining K-map simplification exercises from the original figures.

Digital Circuits – Integrated Assessment Section

Unit 5 – Digital Comparator

Pre-Test

1. For a one-bit comparator, F1=1 when (A,B) is: A) (0,0) B) (0,1) C) (1,0) D) (1,1).

2. For a one-bit comparator, F2=1 when (A,B) is: A) (0,0) B) (0,1) C) (1,0) D) (1,1).

3. For a one-bit comparator, F3=1 when the ordered input pairs are: A) (0,0) and (0,1) B) (0,1) and (1,0) C) (1,0) and (1,1) D) source alternatives as shown in the original.

Digital Circuits – Integrated Assessment Section

Unit 6 – Decoders

Pre-Test

1. A decoder is a logic circuit that: A) Converts an input code from binary to other systems B) Combines inputs C) Converts an input code from other systems to binary D) Finds the 1’s complement.

2. To design a binary-to-octal decoder, we need: A) Eight inputs and three AND gates with inverters B) Three inputs and eight AND gates with inverters C) Three inputs and eight OR gates with inverters D) Seven inputs and three AND gates with inverters.

3. Draw the internal structure of a binary-to-octal decoder with an active-low enable input.

4. Connect the gate whose output corresponds to value 0 to the appropriate inputs.

Digital Circuits – Integrated Assessment Section

Unit 7 – Encoding

Pre-Test

1. An encoder is a logic circuit that: A) Inverts inputs at the output B) Combines inputs C) Converts an input code from one system to another D) Finds the 2’s complement.

2. To design an octal-to-binary encoder, we need: A) Eight inputs and three OR gates B) Eight inputs and four OR gates C) Eight inputs and two OR gates D) Seven inputs and three OR gates.

3. Draw the internal structure of an octal-to-binary encoder with an active-low enable input.

4. Connect switch number 8 to the appropriate OR gate.

Digital Circuits – Integrated Assessment Section

Unit 8 – Half and Full Adders

Pre-Test

1. A half adder has how many inputs and outputs? A) (3,2) B) (2,3) C) (2,2) D) (1,2).

2. A half adder is used to add: A) Two digits B) Two numbers C) A digit and a number D) Two sine waves.

3. A half adder consists of: A) NOR and AND B) XOR and OR C) AND and NAND D) XOR and AND.

4. A full adder consists of: A) NOT gate and a half adder; the remaining alternatives are as in the source.

Digital Circuits – Integrated Assessment Section

Unit 9 – Half and Full Subtractors

Pre-Test

1. A half subtractor has how many inputs and outputs? A) (3,2) B) (2,3) C) (2,2) D) (1,2).

2. A half subtractor is used to subtract: A) Two digits B) Two numbers C) A digit and a number D) Two sine waves.

3. A half subtractor consists of: A) NOR, AND and NOT B) XOR, OR and NOT C) AND, NAND and NOT D) XOR, AND and NOT.

4. A full subtractor consists of: A) NOT gate and a half subtractor; the remaining alternatives are as in the source.

Digital Circuits – Integrated Assessment Section

Unit 10 – Parallel Binary Addition/Subtraction

Pre-Test

1. A parallel binary adder is used to add _____ numbers: A) Decimal B) Octal C) Binary D) Hexadecimal.

2. A parallel binary adder consists of _____ plus a number of full adders: A) Two half adders B) One half adder C) Three half adders D) Four half adders.

3. The number of full adders _____ as the number of binary bits to be added increases: A) Decreases B) Increases C) Is reduced D) Remains equal.

4. To add two 5-bit binary numbers, we need one half adder plus _____ full adders: A) Two B) Three C) Four D) Five.

1. **Rationale**

One of the important circuits in the calculation and logic unit (ALU) of the central processing unit (CPU) in electronic computers are arithmetic circuits, which perform arithmetic operations such as addition, subtraction, and others.
This lecture was designed for the student to learn how to design the circle of half of the mosque - the circle of the complete mosque.

C. Central Ideas

First: Designing the circle of half of the mosque.
Second: The design of the circle of the complete mosque.

D. Objectives of the lecture

After studying this lecture, the student will be able to:
- Designs the circle of half of the mosque.
- Designs the circle of the full mosque.

1. **Rationale**

One of the important circuits in the calculation and logic unit (ALU) of the central processing unit (CPU) in electronic computers are arithmetic circuits, which perform arithmetic operations such as addition, subtraction, and others.

This lecture is designed for the student to learn how to design a parallel binary addition circle - using a parallel binary addition circle to subtract two binary numbers / the complement method of (1).

C. Central Ideas

First: Design of the parallel binary addition circle.

Second: How to use the parallel binary addition circle to subtract two binary numbers / the complementary method of (1).

D. Objectives of the lecture

After studying this lecture, the student will be able to:

- Designs the parallel binary addition circuit.

- Uses the parallel binary addition circuit to subtract two binary numbers / the complement method of (1).

Rationale

One of the important circuits in the calculation and logic unit (ALU) of the Central Processing Unit (CPU) in electronic computers is the digital comparator circuit, which compares the data entering it and gives outputs showing the data statuses whether they are equal, greater or smaller.
This lecture is designed for the student to learn how to design the digital comparison circle in a simplified way for one and two ranks.

C. Central Ideas

First: Design of the circuit of the digital comparator with one rank.
Second: Design of the digital comparator circle with rankers.

D. Objectives of the lecture

After studying this lecture, the student will be able to:
- Learn how to design a single-rank digital comparator circuit.
- Learn how to design the circle of digital comparator with ranks.

Lecture Six

Arithmetic Logic Circuits

Half Adder:

From the verbal explanation of a half adder, we find that this circuit needs two binary inputs and two binary outputs. The input variables designate the augend and addend bits; the output variables produce the sum and carry. We assign symbols x and y to the two inputs and S (for sum) and C (for carry) to the outputs. The truth table for the half adder is listed in Table below . The C output is 1 only when both inputs are 1. The S output represents the least significant bit of the sum.

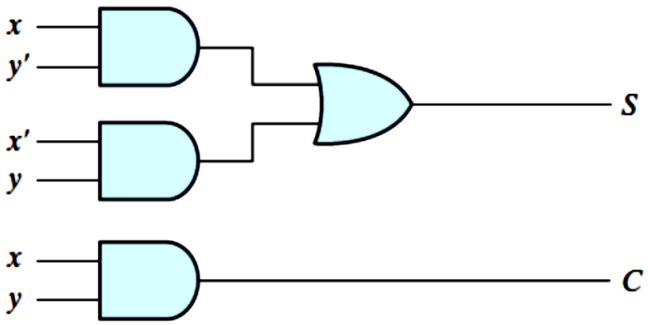
Half Adder

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

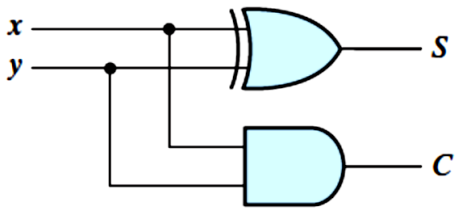
The simplified Boolean functions for the two outputs can be obtained directly from the truth table. The simplified sum-of-products expressions are

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

$$C = xy$$



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



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

Full adder:

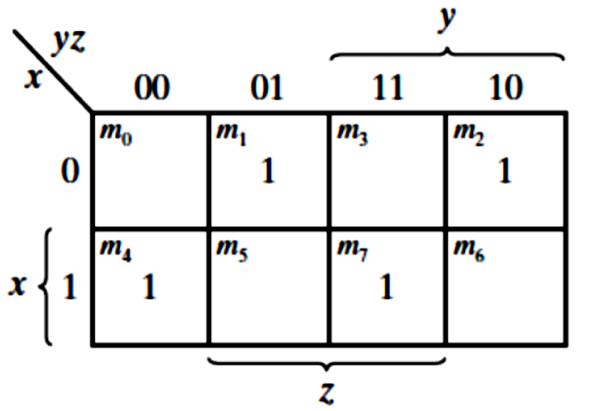
A full adder is a combinational circuit that forms the arithmetic sum of three bits. It consists of three inputs and two outputs. Two of the input variables, denoted by x and y, represent the two significant bits to be added. The third input, z, represents the carry from the previous lower significant position. Two outputs are necessary because the arithmetic sum of three binary digits ranges in value from 0 to 3, and binary representation of 2 or 3 needs two bits. The two outputs are designated by the symbols S for sum and C for carry. The binary variable S gives the value of the least significant bit of the sum. The binary variable C gives the output carry formed by adding the input carry and the bits of the words. The truth table of the full adder is listed in Table below. The eight rows under the input variables designate all possible combinations of the three variables. The output variables are determined from the arithmetic sum of the input bits. When The S output is equal to 1 when only one input is equal to 1 or when all three inputs are equal to 1. The C output has a carry of 1 if two or three inputs are equal to 1.

The input and output bits of the combinational circuit have different interpretations at various stages of the problem. On the one hand, physically, the binary signals of the inputs are considered binary digits to be added arithmetically to form a two-digit sum at the output. On the other hand, the same binary values are considered as variables of Boolean functions when expressed in the truth table or when the circuit is implemented with logic gates. The maps for the outputs of the full adder are shown in Fig. 2 . The simplified expressions are:

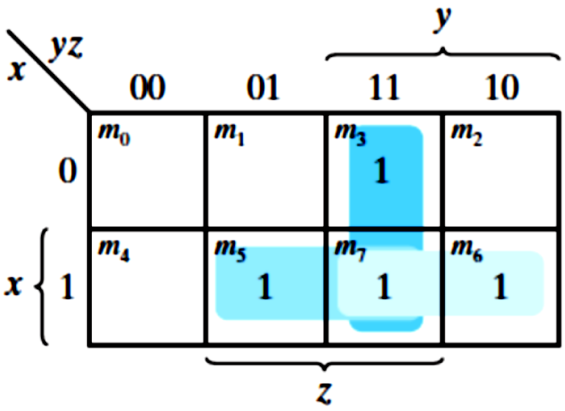
$$S = x'y'z + x'yz' + xy'z' + xyz$$
$$C = xy + xz + yz$$

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

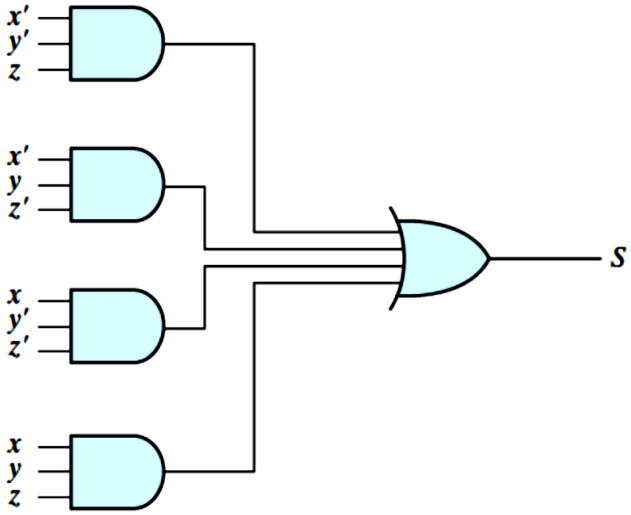


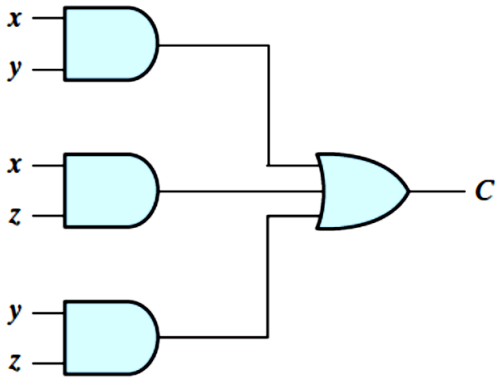
(a) $S = x'y'z + x'yz' + xy'z' + xyz$



(b) $C = xy + xz + yz$

K-Maps for full adder





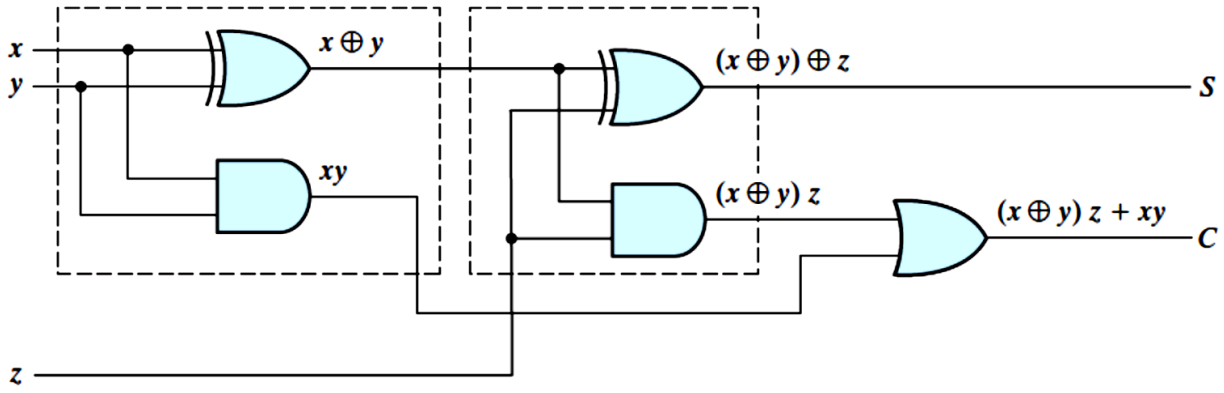
The S output from the second half adder is the exclusive-OR of z and the output of the first half adder, giving:

$$\begin{aligned} S &= z \oplus (x \oplus y) \\ &= z'(xy' + x'y) + z(xy' + x'y)' \\ &= z'(xy' + x'y) + z(xy + x'y') \\ &= xy'z' + x'yz' + xyz + x'y'z \end{aligned}$$

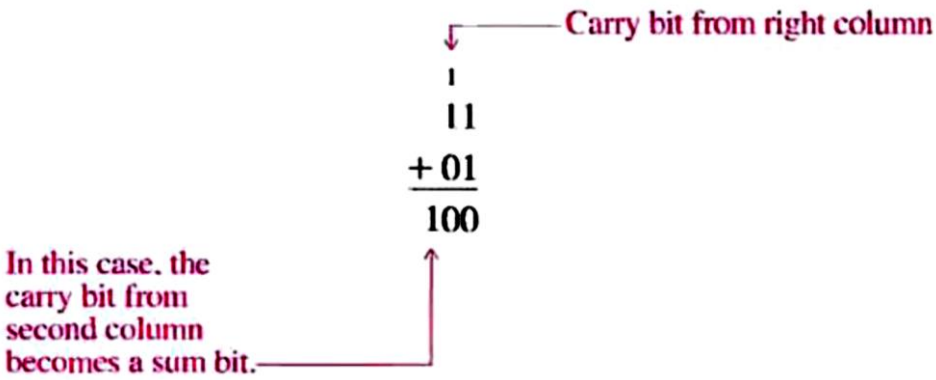
$$C = z(xy' + x'y) + xy = xy'z + x'yz + xy$$

Binary Adder:

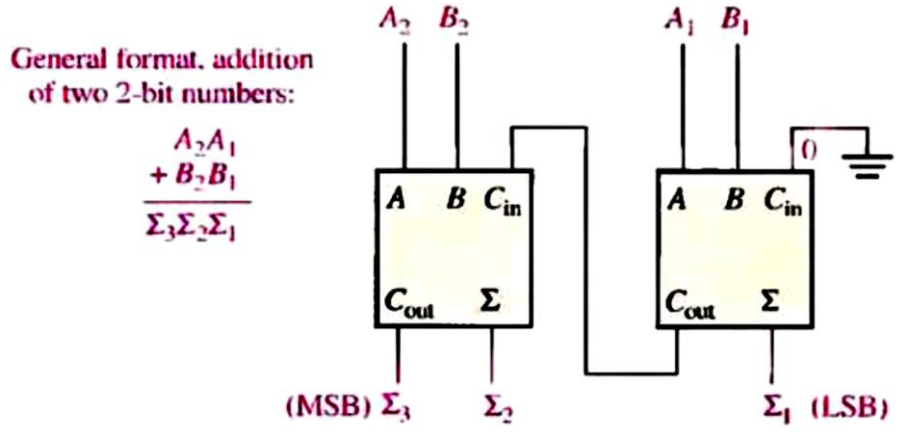
A binary adder is a digital circuit that produces the arithmetic sum of two binary numbers. It can be constructed with full adders connected in cascade, with the output carry from each full adder connected to the input carry of the next full adder in the chain.



A single full-adder is capable of adding two 1-bit numbers and an input carry. To add binary numbers with more than one bit, you must use additional full-adders. When one binary number is added to another, each column generates a sum bit and a 1 or 0 carry bit to the next column to the left, as illustrated here with 2-bit numbers.



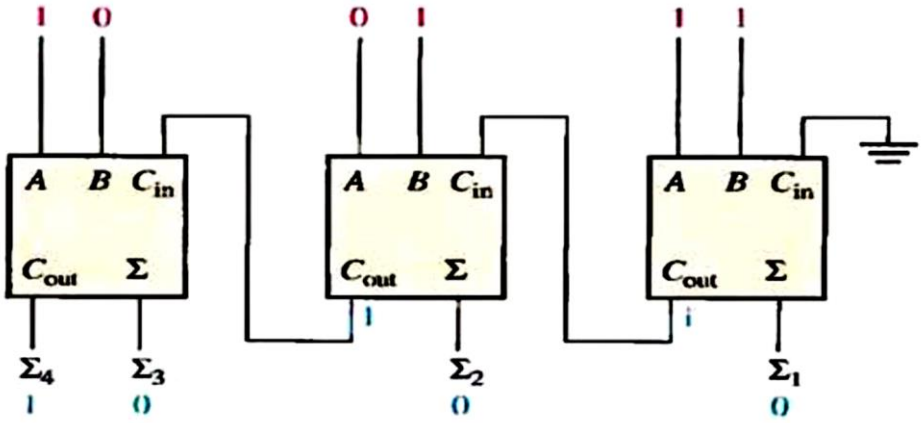
To add two binary numbers, a full-adder is required for each bit in the numbers. So for 2-bit numbers, two adders are needed; for 4-bit numbers, four adders are used; and so on. The carry output of each adder is connected to the carry input of the next higher-order adder, as shown in Figure below for a 2-bit adder. Notice that either a half-adder can be used for the least significant position or the carry input of a full-adder can be made 0 (grounded) because there is no carry input to the least significant bit position.



In this figure the least significant bits (LSB) of the two numbers are represented by A1 and B1 . The next higher-order bits are represented by A2 and B2 . Notice that the output carry from the left-most full-adder becomes the most significant bit (MSB) in the sum 3.

Ex.1: **Determine the result of the following: 101 and 011**

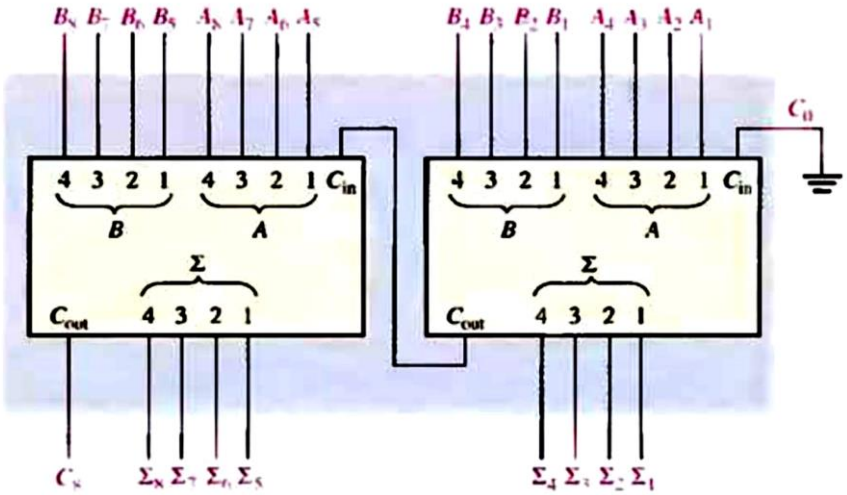
Sol:



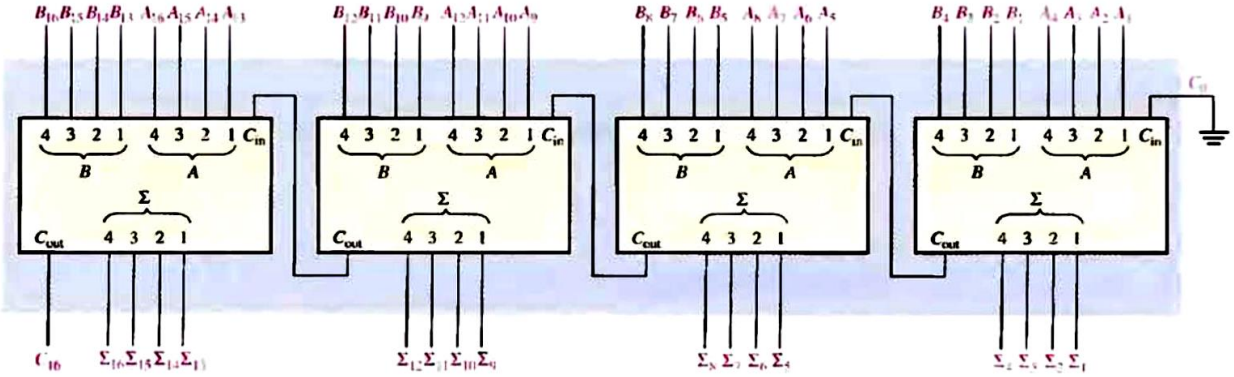
Parallel Adders:

The 4-bit parallel adder can be expanded to handle the addition of two 8-bit numbers by using two 4-bit adders. The carry input of the low-order adder (Co) is connected to ground because there is no carry into the least significant bit position, and the carry output of the low-order adder is connected to the carry input of the high-order adder, as shown in Figure below. This process is known as cascading. Notice that, in this case, the output carry is designated C8 because it is generated from the eighth bit position. The low-order adder is the one that adds the lower or less significant four bits in the numbers, and the high-order adder is the one that adds the higher or more significant four bits in the 8-bit numbers .

Similarly, four 4-bit adders can be cascaded to handle two 16-bit numbers as shown in Figure below. Notice that the output carry is designated C16 because it is generated from the sixteenth bit position.



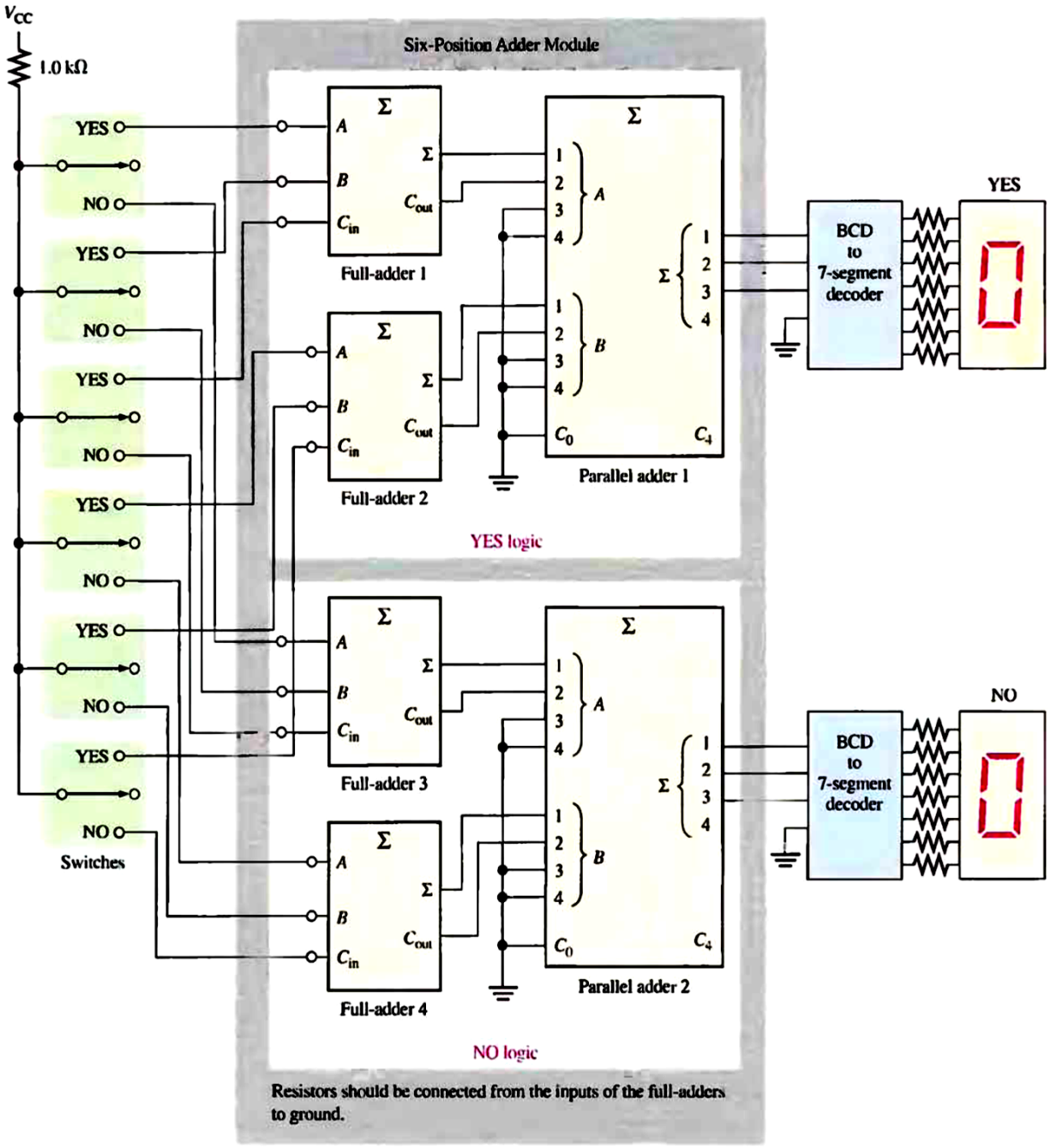
(a) Cascading of two 4-bit adders to form an 8-bit adder



(b) Cascading of four 4-bit adders to form a 16-bit adder

An Application:

An example of full-adder and parallel adder application is a simple voting system that can be used to simultaneously provide the number of "yes" votes and the number of "no" votes . This type of system can be used where a group of people are assembled and there is a need for immediately determining opinions (for or against), making decisions, or voting on certain issues or other matters.


V_{CC}
1.0 kΩ
Six-Position Adder Module
YES
NO
YES
NO
YES
NO
YES
NO
YES
NO
YES
NO
Switches
Full-adder 1
Full-adder 2
Parallel adder 1
YES logic
Full-adder 3
Full-adder 4
Parallel adder 2
NO logic
BCD to 7-segment decoder
BCD to 7-segment decoder
YES
NO
Resistors should be connected from the inputs of the full-adders to ground.

Half Subtractor:

We can explain half subtractor as the following by two inputs and two output:

x	Y	D	B
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Where B is the barrow and the D is the diffraction

$$D = x \oplus y$$

$$B = \overline{x}\, y$$

Full Subtractor:

Full subtractor implemented by three inputs and two outputs:

X	Y	z	D	B
0	0	0	0	0
0	0	1	1	1

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

$$
\begin{aligned}
S &= z \oplus (x \oplus y) \\
&= z'(xy' + x'y) + z(xy' + x'y)' \\
&= z'(xy' + x'y) + z(xy + x'y') \\
&= xy'z' + x'yz' + xyz + x'y'z
\end{aligned}
$$

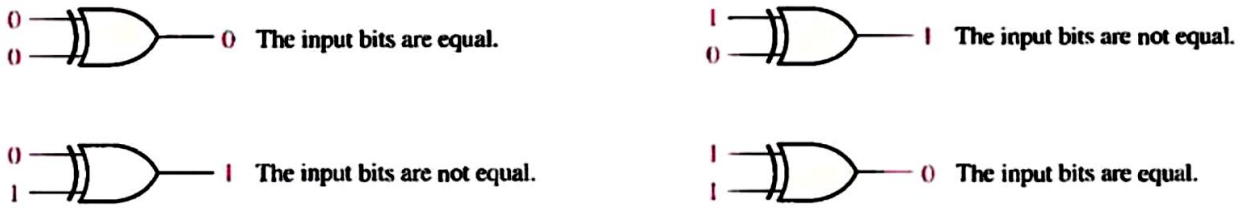
$$B=\overline{X}\,Y+\overline{X}\,Z+YZ$$

Comparators:

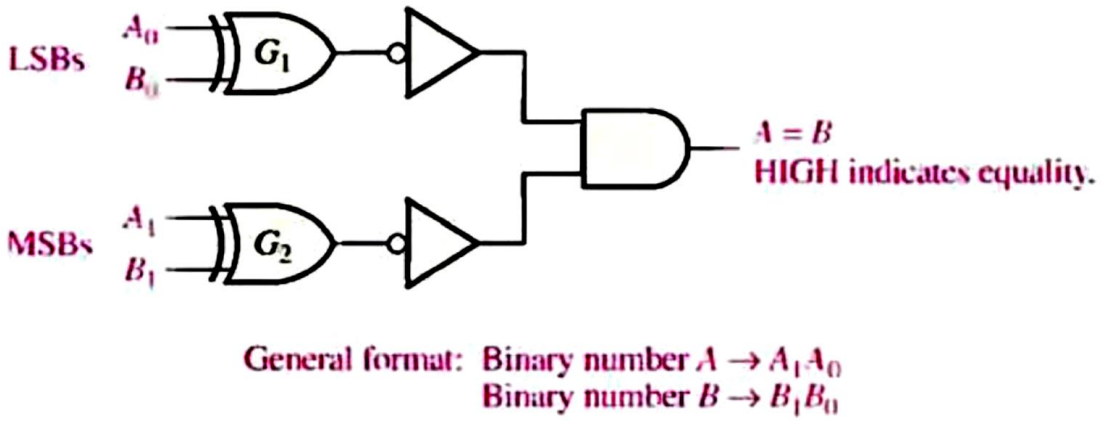
The basic function of a comparator is to compare the magnitudes of two binary quantities to determine the relationship of those quantities. In its simplest form. A comparator circuit determines whether two numbers are equal.

Equality:

The exclusive-OR gate can be used as a basic comparator because its output is a 1 if the two input bits are not equal and a 0 if the input bits are equal. Figure below shows the exclusive-OR gate as a 2-bit comparator.

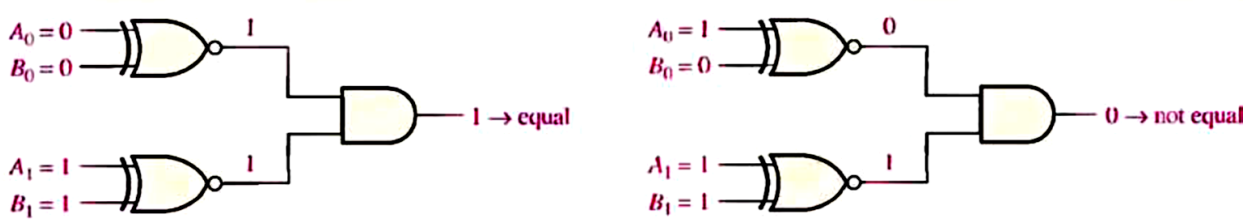


In order to compare binary numbers containing two bits each, an additional exclusive - OR gate is necessary.



Ex. **Apply each of the following sets of binary numbers to the comparator inputs in Figure below and determine the output by following the logic levels through the circuit.**

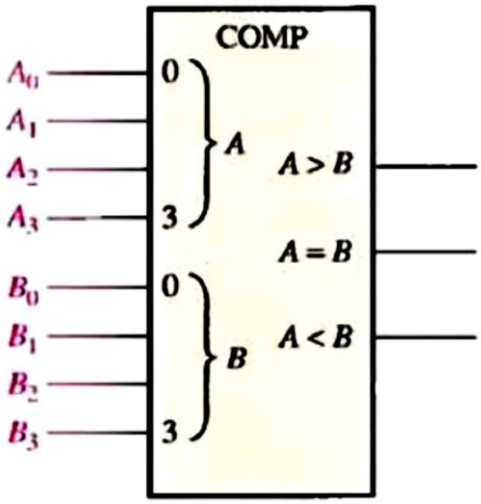
a) 10 and 10 **b) 11 and 10**



Sol:

Inequality:

In addition to the equality output, many IC comparators provide additional outputs that indicate which of the two binary numbers being compared is the larger. There is an output that indicates when number A is greater than number B (A > B) and an output that indicates when number A is less than number B (A < B), as shown in the logic symbol for a 4-bit comparator in Figure below:

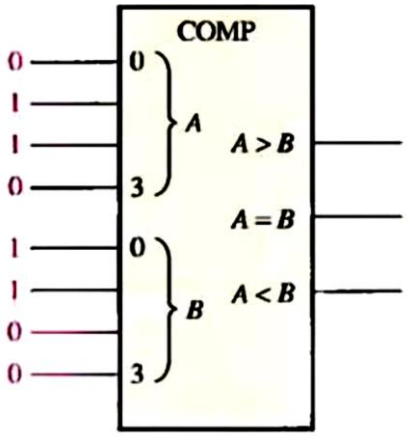


To determine an inequality of binary numbers A and B, you first examine the highest-order bit in each number. The following conditions are possible:

1. If A3 = 1 and B3 = 0, number A is greater than number B.
2. If A3 = 0 and B3 = 1 number A is less than number B.
3. If A3 = B 3, then you must examine the next lower bit position for an inequality.

These three operations are valid for each bit position in the numbers. The general procedure used in a comparator is to check for an inequality in a bit position, starting with the highest-order bits (MSBs). When such an inequality is found, the relationship of the two numbers is established, and any other inequalities in lower-order bit positions must be ignored because it is possible for an opposite indication to occur; the highest-order indication must take precedence.

Ex. Determine the A = B, A > B, and A < B outputs for the input numbers shown on the comparator in Figure below: A= 0110 B= 0011



Sol:

The number on the A inputs is 0110 and the number on the B inputs is 0011. The A > B output is HIGH and the other outputs are LOW.

Post-Test

1. For a two-bit comparator, F1=1 when (A2,B2) is: A) (0,0) B) (0,1) C) (1,0) D) (1,1).

2. For a two-bit comparator, F2=1 when (A2,B2) is: A) (0,0) B) (0,1) C) (1,0) D) (1,1).

3. For a two-bit comparator, F3=1 when (A2,B2) is: A) (0,1) B) (1,0) C) (1,1) D) (0,0).

Post-Test

1. Draw the internal structure of a binary-to-decimal decoder with an active-high enable input.
2. A decoder is a logic circuit located after _____ and before _____ in electronic computers.
3. Connect the gate corresponding to output value 5 to the appropriate inputs.
4. Connect the gate corresponding to output value 6 to the appropriate inputs.
5. Connect the gate corresponding to output value 7 to the appropriate inputs.
6. Connect the gate corresponding to output value 9 to the appropriate inputs.

Post-Test

1. Draw the internal structure of a decimal-to-binary encoder with an active-high enable input.
2. An encoder is a logic circuit located after _____ and before _____ in electronic computers.
3. Connect switch 7 to the three appropriate OR gates.
4. Connect switch 9 to the two appropriate OR gates.

Post-Test

1. The gate representing carry in a full adder is: A) XOR B) AND C) OR D) NOR.
2. The gate representing the sum in a half adder is: A) AND B) NAND C) XNOR D) XOR.
3. The gate representing carry in a half adder is: A) XOR B) AND C) OR D) NOR.
4. Design a full-adder circuit.

Post-Test

1. The gate representing borrow in a full subtractor is: A) XOR B) AND C) OR D) NOR.
2. The gate representing the difference in a half subtractor is: A) AND B) NAND C) XNOR D) XOR.
3. The gate representing borrow in a half subtractor is: A) XOR B) AND C) OR D) NOR.
4. Design a full-subtractor circuit.

Post-Test

1. The borrow output from the last stage of a parallel binary subtractor is: A) Added to the first stage B) Added to the second stage C) Passed as the MSB of the result D) Ignored.

2. Find the subtraction result using a parallel adder and inverters for the binary numbers shown in the source: 1101 and 1011.

3. Repeat for 1101 and 1010.

4. To add two 6-bit binary numbers, we need: A) One half adder + one full adder B) One half adder + five full adders C) One half adder + six full adders D) One half adder + four full adders.

Digital Circuits – Integrated Assessment Section

Unit 11 – Flip-Flops

Pre-Test

1. The main feature shared by sequential circuits, distinguishing them from combinational circuits, is: the present output depends on the previous output as well as current inputs.

2. An SR flip-flop has three allowed input states _____ and one forbidden state _____.

3. A JK flip-flop has four allowed input states: _____.

4. The relationship between the two outputs of a flip-flop is: the first output is the inverse of the second.

Digital Circuits – Integrated Assessment Section

Unit 12 – Flip-Flop Applications

Pre-Test

1. Draw the output waveforms for a master/slave SR flip-flop.
2. Draw the output waveforms for a master/slave JK flip-flop.
3. Draw a master/slave SR flip-flop circuit.

1. **Rationale**

Due to the widespread use of vibrators as basic building blocks for meter circuits, displacement registers, storage, control and control circuits, in addition to many other applications such as coding detectors, comparison, frequency division circuits and others.

This lecture is designed for the student to learn the installation of different swing circles and deduce their realistic tables.

C. Central Ideas

First: Study the structure of the RS weighted circuit and deduce its realistic table.

Second: Study the structure of the T-weighted circuit and deduce its realistic table.

Third: Study the structure of the JK swing circuit and deduce its realistic table.

Fourth: Study the structure of the D-weighted circuit and deduce its realistic table.

Fifth: Adding pulse control to the aforementioned swings.

D. Objectives of the lecture

After studying this lecture, the student will be able to:

- Draws a RS weighted circle and writes a table of realism.
- Draws a T-weighted circle and writes a table of its realism.
- Draws a JK swing circle and writes its realistic table.
- Draws a D-swing circle and writes a table of its realism.
- Draws the control pulses entering and exiting the aforementioned swings.

Lecture Seven

Multivibrators

Flip-flops (Multivibrators) are actually an application of logic gates. With the help of Boolean logic, you can create memory with them. Flip flops can also be considered as the most basic idea of a Random Access Memory [RAM]. When a certain input value is given to them, they will be remembered and executed, if the logic gates are designed correctly. A higher application of flip-flops is helpful in designing better electronic circuits.

The most commonly used application of flip-flops is in the implementation of a feedback circuit. As a memory relies on the feedback concept, flip flops can be used to design it.

There are mainly four types of flip-flops that are used in electronic circuits. They are:

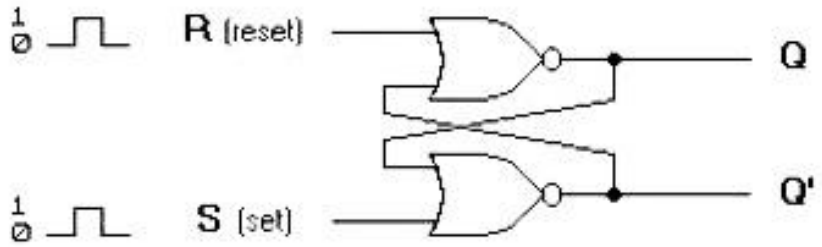
1. **The basic Flip Flop or S-R Flip Flop**
2. **Delay Flip Flop [D Flip Flop]**
3. **J-K Flip Flop**
4. **T Flip Flop**

1. S-R Flip Flop

The SET-RESET flip-flop is designed with the help of two NOR gates and also two NAND gates. These flip-flops are also called S-R Latch.

- **S-R Flip Flop using NOR Gate**

The design of such a flip-flop includes two inputs, called the SET [S] and RESET [R]. There are also two outputs, Q and Q'. The diagram and truth table is shown in figure 1 below.



(a) Logic diagram

S	R	Q	Q'	
1	0	1	0	
0	0	1	0	(after S=1, R=0)
0	1	0	1	
0	0	0	1	(after S=0, R=1)
1	1	0	0	

(b) Truth table

Figure 1: Basic flip-flop circuit with NOR gates.

From the diagram it is evident that the flip flop has mainly four states. They are:

S=1, R=0 then Q=1, Q'=0

This state is also called the ***SET*** state.

S=0, R=1 then Q=0, Q'=1

This state is known as the ***RESET*** state.

In both the states you can see that the outputs are just compliments of each other and that the value of Q follows the value of S.

S=0, R=0 then Q & Q' = Remember

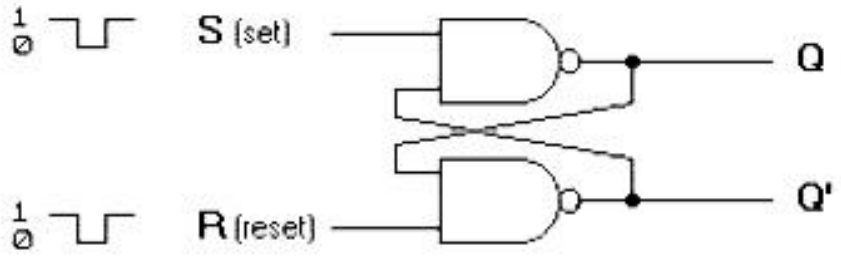
If both the values of S and R are switched to 0, then the circuit remembers the value of S and R in their previous state.

S=1, R=1 then Q=0, Q'=0 [Invalid]

This is an ***invalid*** state because the values of both Q and Q' are 0. They are supposed to be compliments of each other. Normally, this state must be avoided.

- **S-R Flip Flop using NAND Gate**

The circuit of the S-R flip flop using NAND Gate and its truth table is shown in figure 2 below.



(a) Logic diagram

S R	Q Q'	
1 0	0 1	
1 1	0 1	(after S=1, R=0)
0 1	1 0	
1 1	1 0	(after S=0, R=1)
0 0	1 1	

(b) Truth table

Figure 2: S-R Flip Flop using NAND Gate

Like the NOR Gate S-R flip-flop, this one also has four states. They are

S=1, R=0 then Q=0, Q'=1

This state is also called the ***SET*** state.

S=0, R=1 then Q=1, Q'=0

This state is known as the ***RESET*** state.

In both the states, you can see that the outputs are just compliments of each other and that the value of Q follows the compliment value of S.

S=0, R=0 then Q=1, & Q' =1 [Invalid]

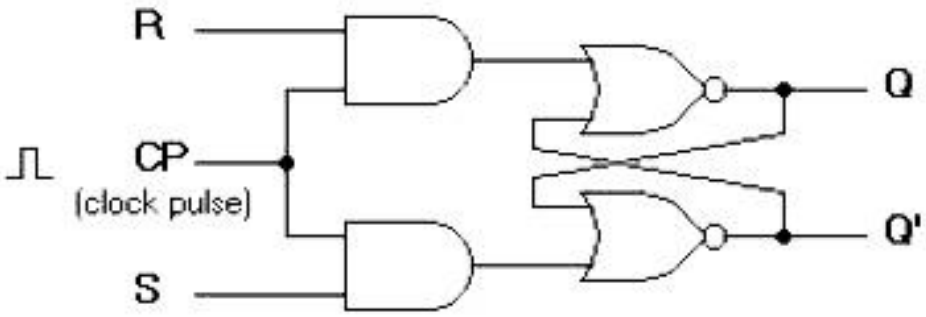
If both the values of S and R are switched to 0 it is an ***invalid*** state because the values of both Q and Q' are 1. They are supposed to be compliments of each other. Normally, this state must be avoided.

S=1, R=1 Q & Q'= Remember

If both the values of S and R are switched to 1, then the circuit remembers the value of S and R in their previous state.

- **Clocked S-R Flip Flop**

It is also called a Gated S-R flip flop. The problems with S-R flip flops using NOR and NAND gate is the invalid state. This problem can be overcome by using a bi-stable SR flip-flop that can change outputs when certain invalid states are met, regardless of the condition of either the Set or the Reset inputs. For this, a clocked S-R flip-flop is designed by adding two AND gates to a basic NOR Gate flip-flop. The circuit diagram and truth table is shown in figure 3 below.



(a) Logic diagram

Q S R	Q(t+1)
0 0 0	0
0 0 1	0
0 1 0	1
0 1 1	indeterminate
1 0 0	1
1 0 1	0
1 1 0	1
1 1 1	indeterminate

(b) Truth table

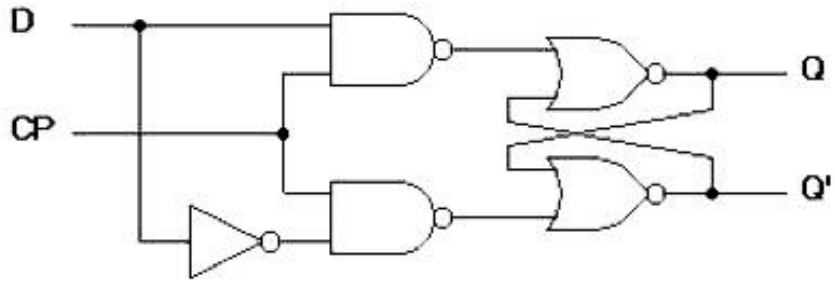
Figure 3: Clocked S-R Flip Flop

A clock pulse [CP] is given to the inputs of the AND Gate. When the value of the clock pulse is ‘0’, the outputs of both the AND Gates remain ‘0’. As soon as a pulse is given the value of CP turns ‘1’. This makes the values at S and R to pass through the NOR Gate flip flop. But when the values of both S and R values turn ‘1’, the HIGH value of CP causes both of them to turn to ‘0’ for a short moment. As soon as the pulse is removed, the flip-flop state becomes intermediate. Thus either of the

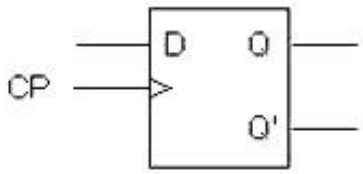
two states may be caused, and it depends on whether the set or reset input of the flip-flop remains a '1' longer than the transition to '0' at the end of the pulse. Thus, the invalid states can be eliminated.

2. D Flip Flop

The circuit diagram and truth table of D Flip-Flop is given in figure 4 below.



(a) Logic diagram with NAND gates



(b) Graphical symbol

Q D	Q(t+1)
0 0	0
0 1	1
1 0	0
1 1	1

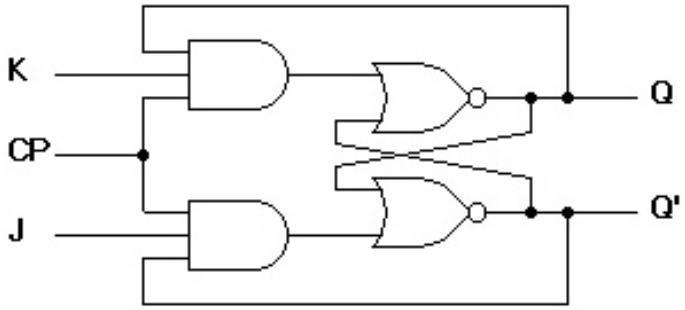
(c) Transition table

Figure 4: Clocked D Flip Flop

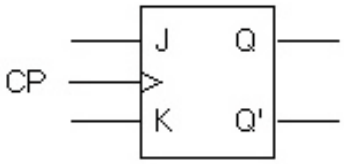
D flip-flop is actually a slight modification of the above-explained clocked SR flip-flop. From the figure, you can see that the D input is connected to the S input and the complement of the D input is connected to the R input. The D input is passed on to the flip flop when the value of CP is ‘1’. When CP is HIGH, the flip flop moves to the SET state. If it is ‘0’, the flip-flop switches to the CLEAR state.

3. J-K Flip Flop

The circuit diagram and truth table of a J-K flip-flop is shown in figure 5 below.



(a) Logic diagram



(b) Graphical symbol

Q J K	Q(t+1)
0 0 0	0
0 0 1	0
0 1 0	1
0 1 1	1
1 0 0	1
1 0 1	0
1 1 0	1
1 1 1	0

(c) Transition table

Figure 5: J-K Flip Flop

A J-K flip-flop can also be defined as a modification of the S-R flip-flop. The only difference is that the intermediate state is more refined and precise than that of a S-R flip flop.

The behavior of inputs J and K is same as the S and R inputs of the S-R flip-flop. The letter J stands for SET and the letter K stands for CLEAR.

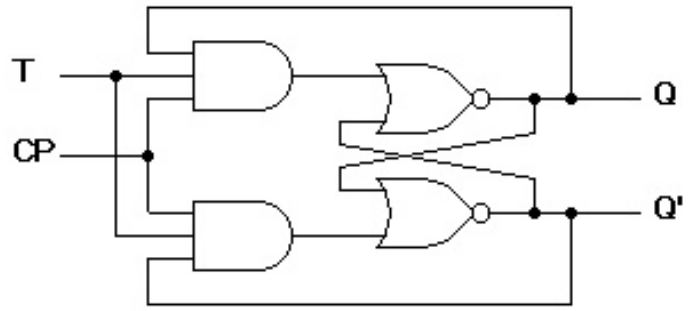
When both the inputs J and K have a HIGH state, the flip-flop switch to the complement state. So, for a value of Q = 1, it switches to Q=0 and for a value of Q = 0, it switches to Q=1.

The circuit includes two 3-input AND gates. The output Q of the flip flop is returned back as a feedback to the input of the AND along with other inputs like K and clock pulse [CP]. So, if the value of CP is '1', the flip flop gets a CLEAR signal and with the condition that the value of Q was earlier 1. Similarly output Q' of the flip flop is given as a feedback to the input of the AND along with other inputs like J and clock pulse [CP]. So the output becomes SET when the value of CP is 1 only if the value of Q' was earlier 1.

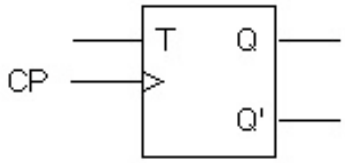
The output may be repeated in transitions once they have been complimented for J=K=1 because of the feedback connection in the JK flip-flop. This can be avoided by setting a time duration lesser than the propagation delay through the flip-flop. The restriction on the pulse width can be eliminated with a master-slave or edge-triggered construction.

4. T Flip Flop

This is a much simpler version of the J-K flip-flop. Both the J and K inputs are connected together and thus are called a single input J-K flip-flop. When clock pulse is given to the flip-flop, the output begins to toggle. Here also the restriction on the pulse width can be eliminated with a master-slave or edge-triggered construction. Take a look at the circuit and truth table in the figure 6 below.



(a) Logic diagram



(b) Graphical symbol

Q T	Q(t+1)
0 0	0
0 1	1
1 0	1
1 1	0

(c) Transition table

Figure 6: T Flip Flop

Examples

Ex.1: If the S and R waveforms in Figure 7(a) are applied to the inputs of the S-R latch, determine the waveform that will be observed on the Q output. Assume that Q is initially LOW .

Sol:

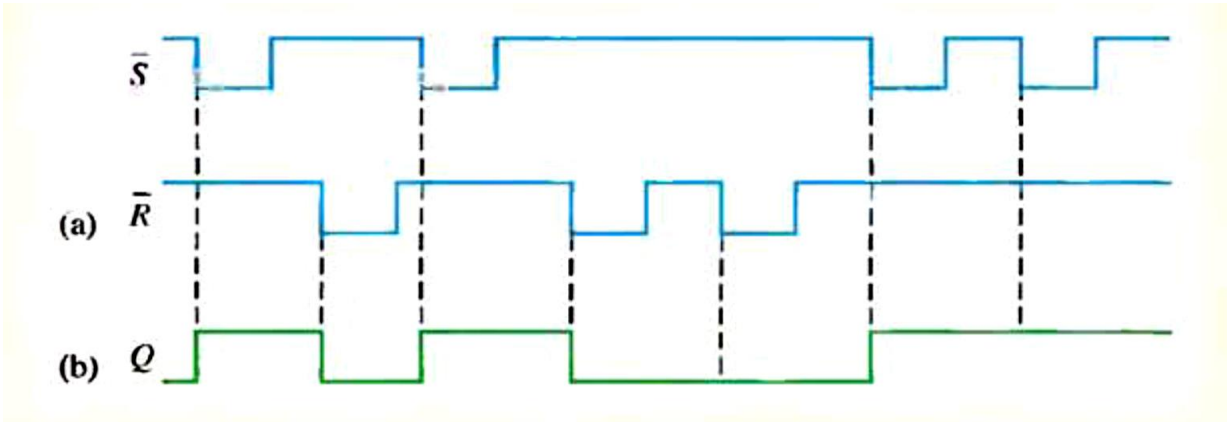


Figure 7

Ex.2: Determine the Q output waveform if the inputs shown in Figure 8 (a) are applied to a gated S-R latch that is initially RESET (*EN* = *CP*) .

Sol:

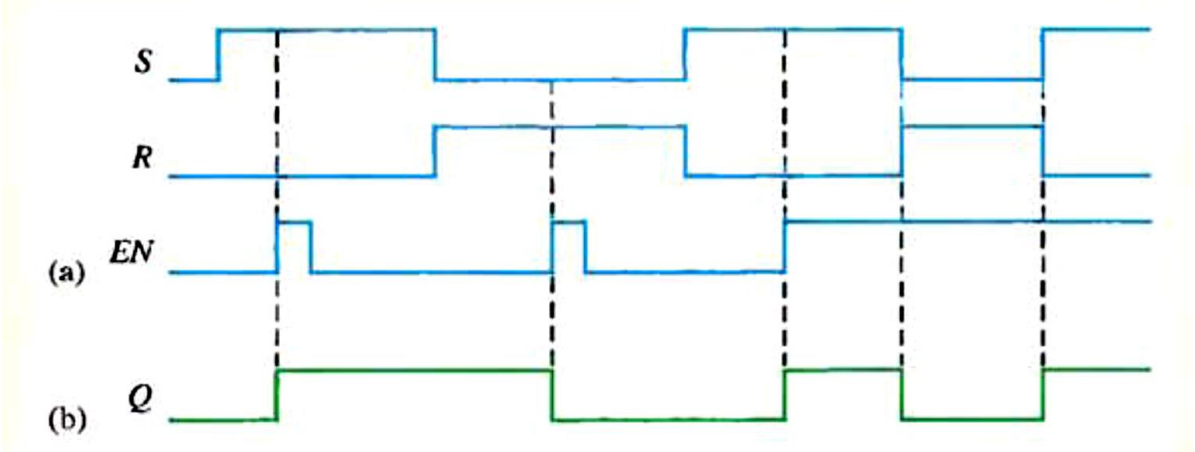


Figure ٨

H.W.1: Determine the Q output of a gated S-R latch if the S and R inputs in Figure 8 (a) are inverted**.**

Ex.3: Determine the Q output waveform if the inputs shown in Figure 9 (a) are applied to a gated D latch. which is initially RESET .

Sol:

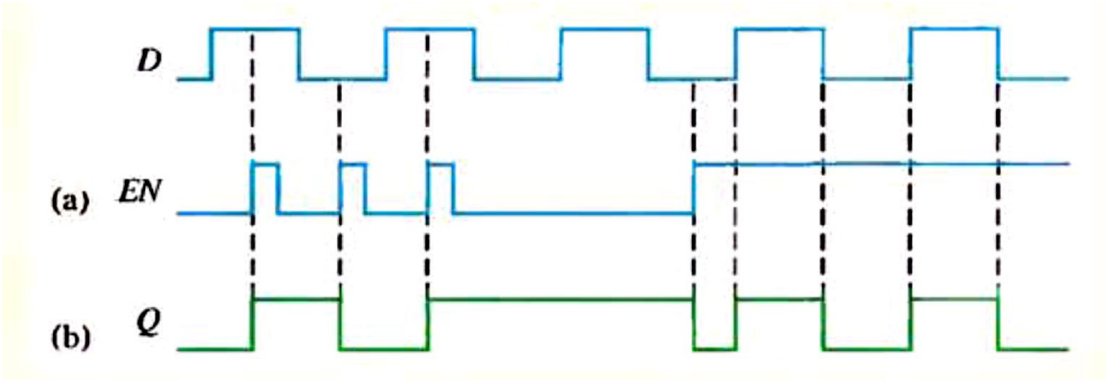


Figure 9

H.W.2: Determine the Q output of the gated D latch if the D input in Figure 9 (a) is inverted.

Ex.4: Determine the Q and Q' output waveforms of the flip-flop in Figure 10 for the S, R ,and CLK inputs in Figure 11 (a). Assume that the positive edge-triggered flip-flop is initially RESET .

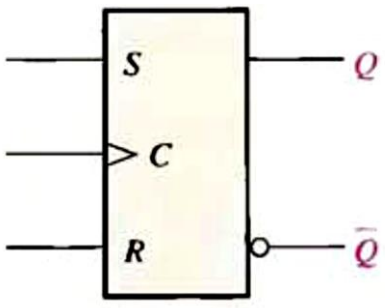


Figure 10

Sol:

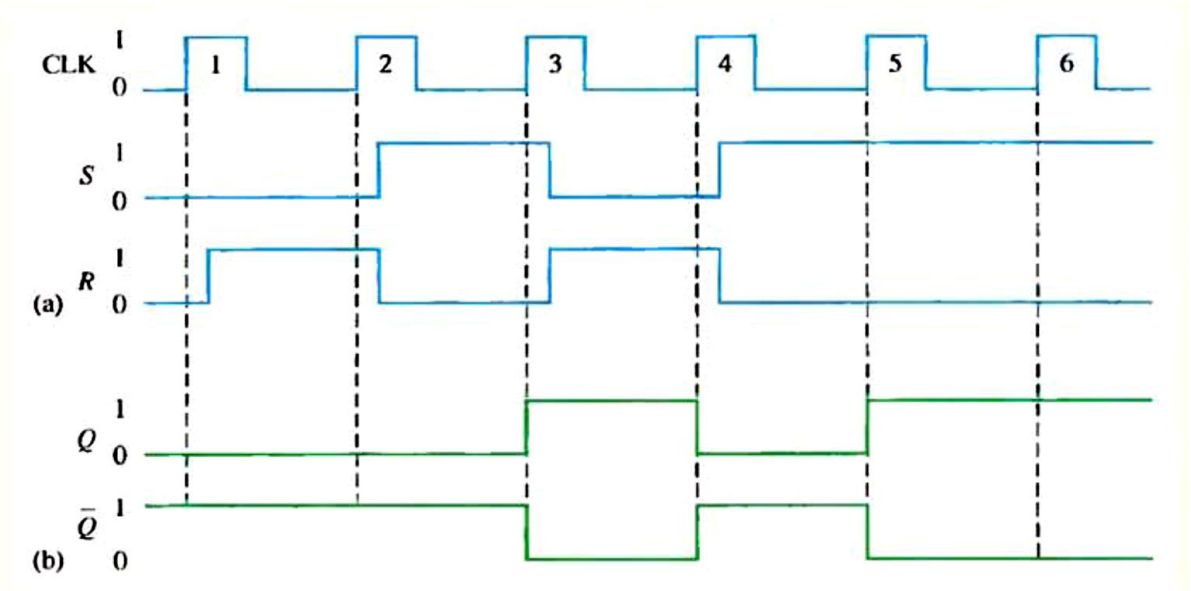


Figure 11

H.W.3: Determine Q and Q for the Sand R inputs in Figure 11(a) if the flip-flop is a negative edge-triggered device.

Ex.5: Given the waveforms in Figure 12 (a) for the D input and the clock. Determine the Q output waveform if the flip-flop starts out RESET.

Sol:

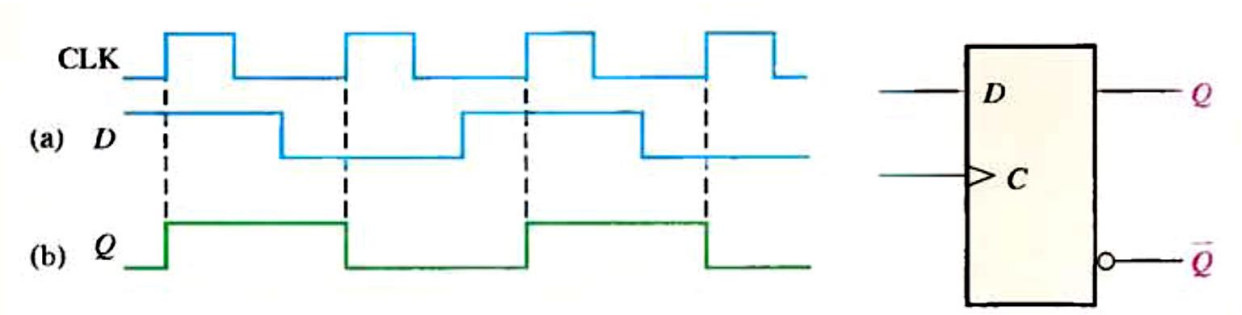


Figure 12

Ex.5: The waveforms in Figure 13 (a) are applied to the J, K, and clock inputs as indicated. Determine the Q output, assuming that the flip-flop is initially RESET .

Sol:

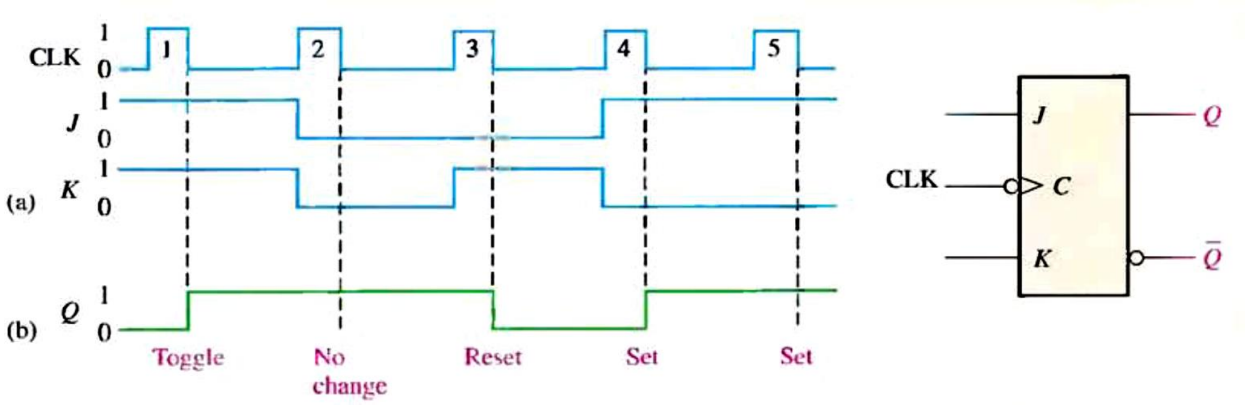


Figure 13

H.W.4: Determine the Q output of the J-K flip-flop if the J and K inputs in Figure 13 (a) are inverted .

Ex.6: The waveforms in Figure 14 (a) are applied to the flip-flop as shown. Determine the Q output, starting in the RESET state .

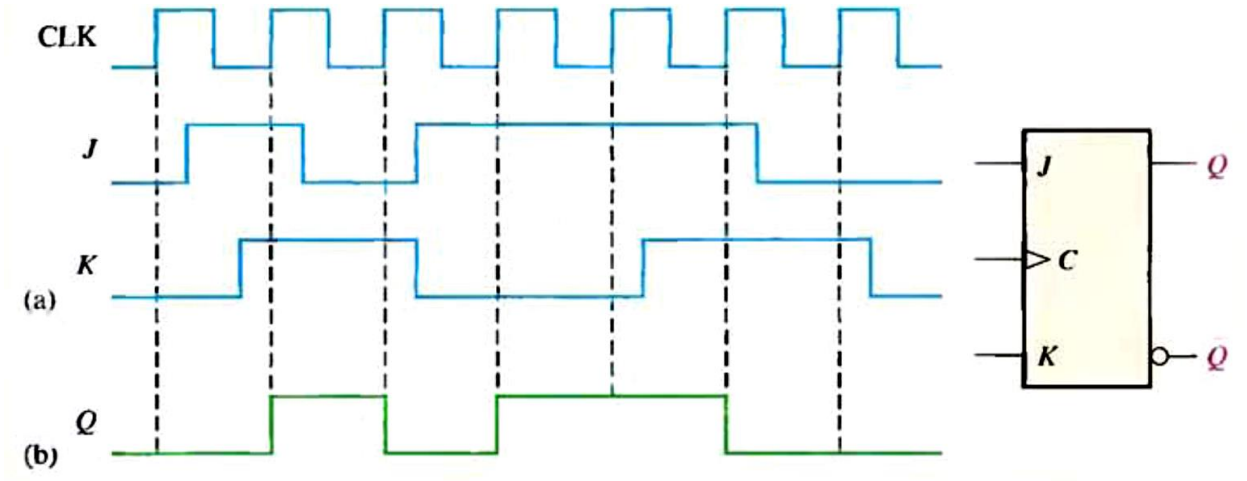


Figure 14

Flip-Flops Applications:

1) Frequency Division:

One application of a flip-flop is dividing (reducing) the frequency of a periodic waveform. When a pulse waveform is applied to the clock input of a J-K flip-flop that is connected to toggle (J = K = I), the Q output is a square wave with one-half the frequency of the clock input. Thus, a single flip-flop can be applied as a divide-by-2 device, as is illustrated in Figure 15. As you can see, the flip-flop changes state on each triggering clock edge (positive edge-triggered in this case). This results in an output that changes at half the frequency of the clock waveform.

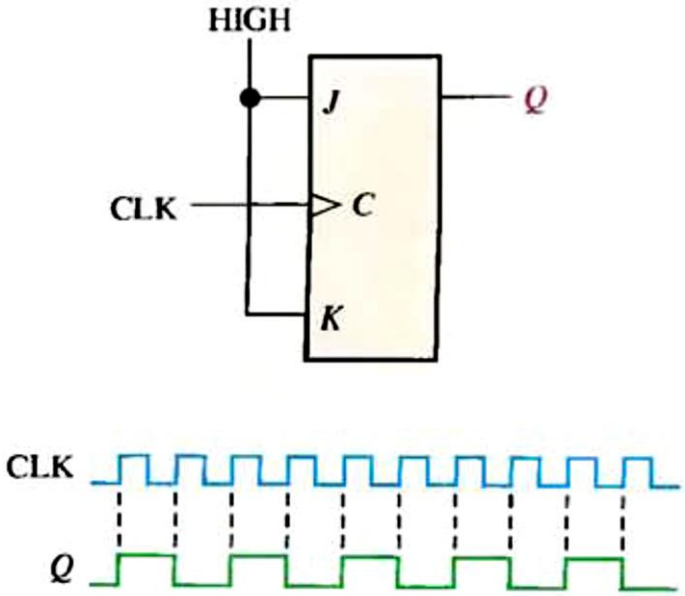


Figure 15

Further division of a clock frequency can be achieved by using the output of one flip-flop as the clock input to a second flip-flop, as shown in Figure 16. The frequency of the Q_A output is divided by 2 by flip-flop B. The Q_B output is, therefore, one-fourth the frequency of the original clock input.

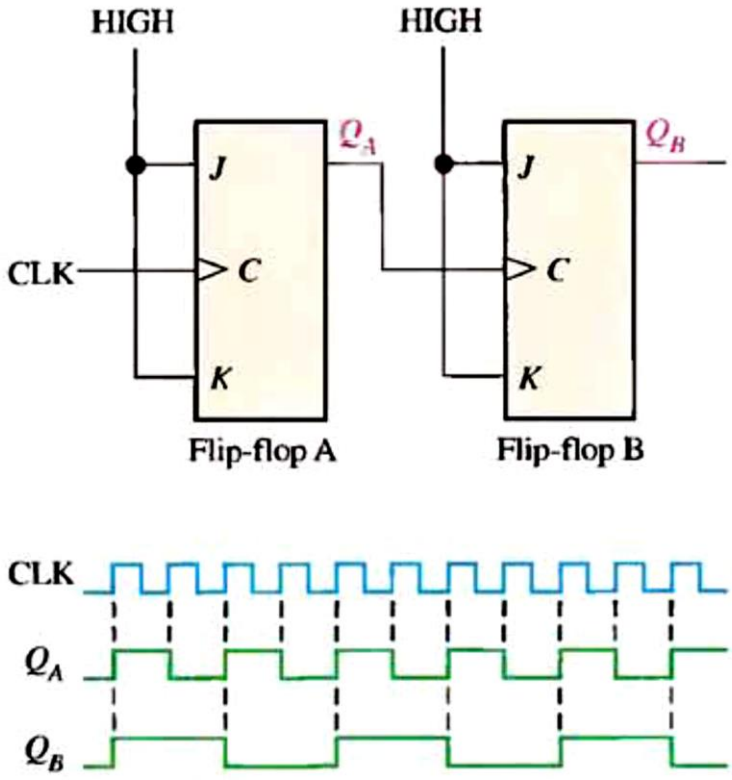


Figure 16

By connecting flip-flops in this way, a frequency division of 2^n is achieved, where n is the number of flip-flops. For example, three flip-flops divide the clock frequency by $2^3 = 8$, four flip-flops divide the clock frequency by $2^4 = 16$; and so on .

Ex.7: Develop the f_{out}, waveform for the circuit in Figure 17 when an 8 kHz square wave input is applied to the clock input of flip-flop A .

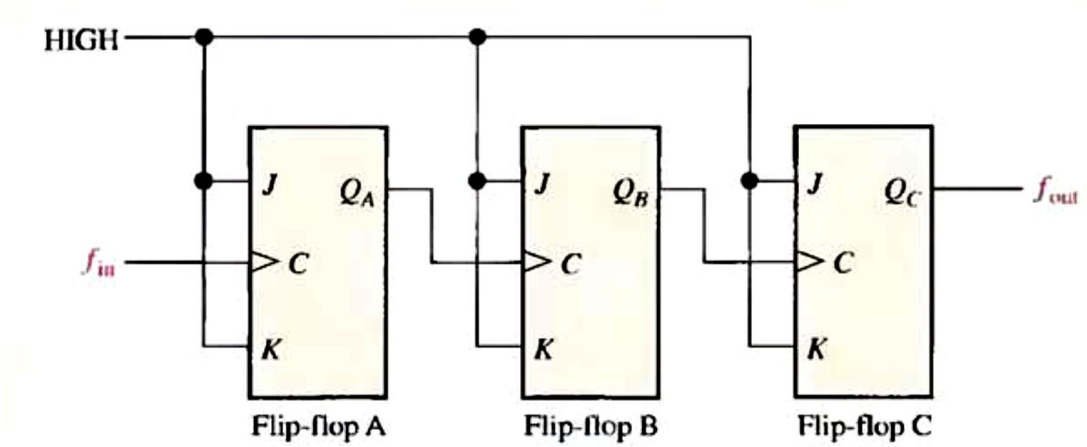


Figure 17

Sol: The three flip-flops are connected to divide the input frequency by eight (2^3 = 8) and the f_{out}, waveform is shown in Figure 18. Since these are positive edge-triggered flip -flops, the outputs change on the positive-going clock edge. There is one output pulse for every eight input pulses, so the output frequency is 1 kHz. Waveforms of Q_A and Q_B are also shown.

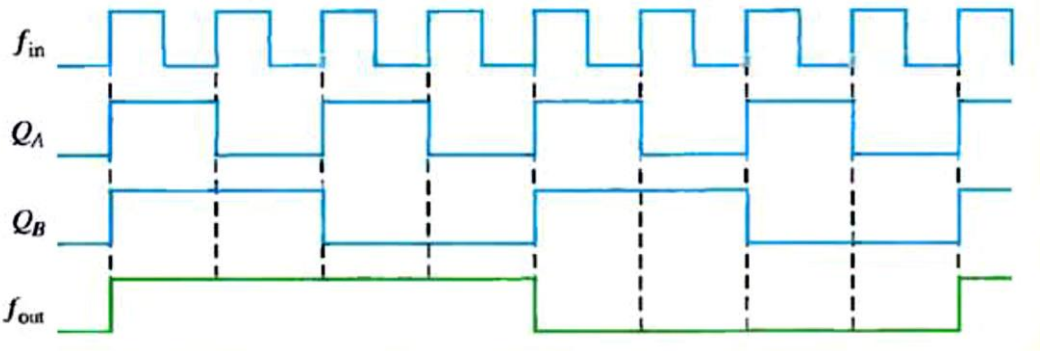


Figure 18

H.W.5: Design the logic circuit to reduce the frequency to 1/32 its value and draw its waveforms .

2) Counting:

Another important application of flip-flops is in digital counters, which are covered in details later. The concept is illustrated in Figure 19. The flip-flops are negative edge-triggered J-Ks. Both flip-flops are initially RESET. Flip-flop A toggles on the negative-going transition of each clock pulse. The Q output of flip-flop A clocks flip-flop B, so each time Q_A makes a HIGH-to-LOW transition, flip-flop B toggles. The resulting Q_A and Q_B waveforms are shown in the figure.

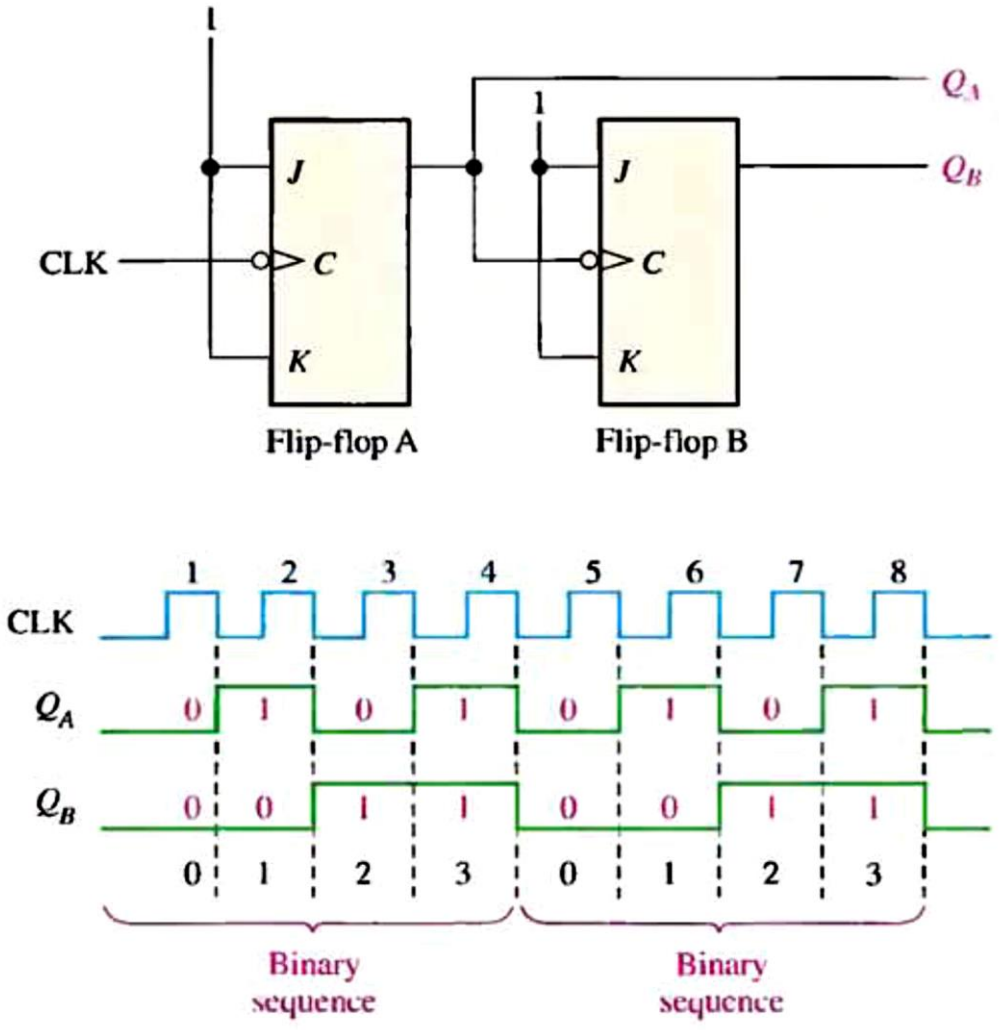


Figure 19

Ex.8: Determine the output waveforms in relation to the clock for Q_A, Q_B, and Q_c in the circuit of Figure 20 and show the binary sequence represented by these waveforms .

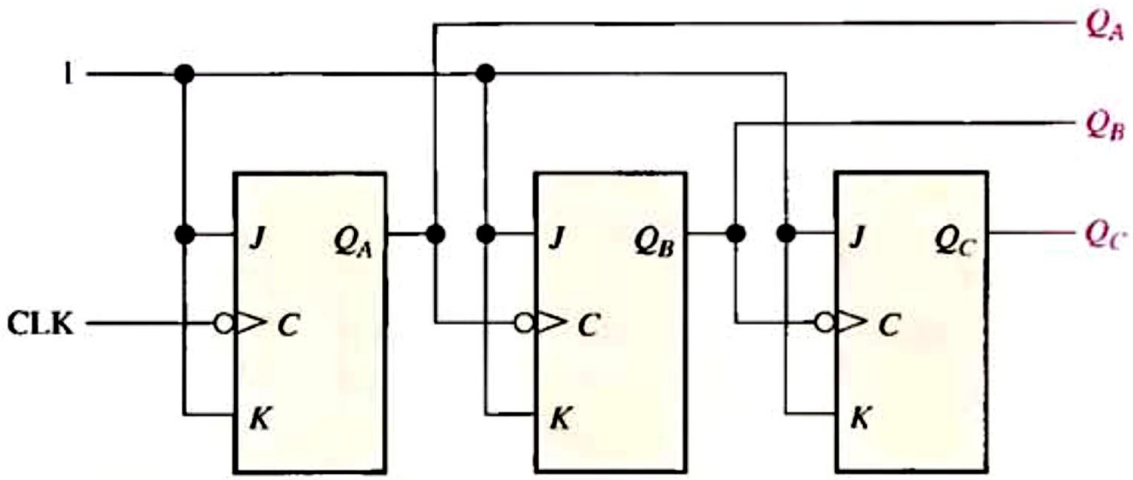


Figure ٢٠

Sol: The output timing diagram is shown in Figure 21 Notice that the outputs change on the negative-going edge of the clock pulses. The outputs go through the binary sequence 000, 001, 010, 011, 100, 101, 110, and 111 as indicated.

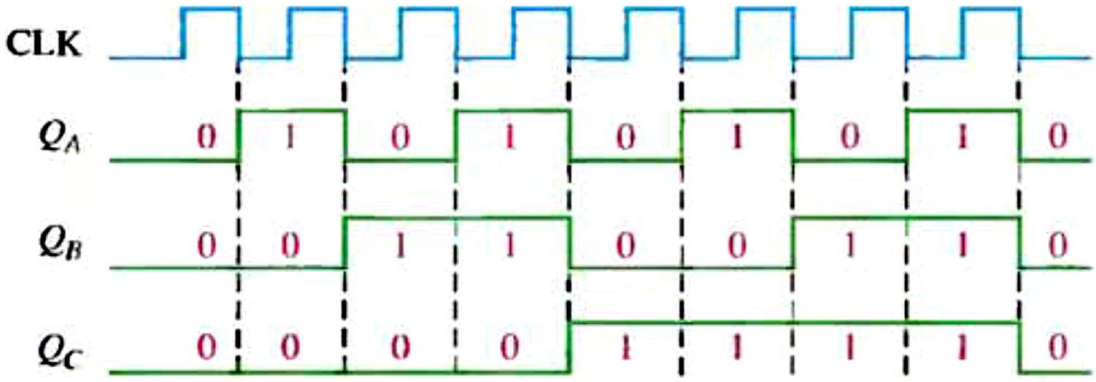


Figure ٢1

H.W.6: How many flip-flops are required to produce a binary sequence representing decimal numbers 0 through 15 ? Design the circuit and waveforms.

Post-Test

1. Draw the SRT flip-flop and write its truth table.
2. Draw the JK flip-flop and write its truth table.
3. Draw the D flip-flop and write its truth table.

Post-Test

1. The JK flip-flop differs from the synchronous SR flip-flop for the input condition that produces _____ in SR and _____ in JK.

2. Draw the logic symbol for the T flip-flop.

3. Draw the logic symbol for the JK flip-flop.

4. Draw the Q output waveform of the synchronous SR flip-flop for the input waveforms in the original source figure, assuming Q=0 before the first pulse.

Digital Circuits – Integrated Assessment Section

Unit 13 – Counters

Pre-Test

1. Draw a four-stage asynchronous binary up counter.
2. Derive the truth table of the above circuit.
3. Draw the output pulse waveforms of the above circuit.
4. Draw a four-stage asynchronous binary down counter.
5. Derive its truth table.
6. Draw its output pulse waveforms.

Digital Circuits – Integrated Assessment Section

Unit 14 – Shift Registers

Pre-Test

1. Draw a four-stage serial-in/parallel-out shift register.

2. Draw a four-stage parallel-in/serial-out shift register.

3. A stage in a shift register consists of: A) latch B) flip-flop C) one byte D) four bytes.

4. To shift one byte serially into a shift register, we need: A) one clock pulse B) one load pulse C) eight clock pulses D) four clock pulses.

5. To shift one byte in parallel into a shift register, we need: A) one clock pulse B) two clock pulses C) four clock pulses D) eight clock pulses.

Digital Circuits – Integrated Assessment Section

Unit 15 – Memory

Pre-Test

1. Mark True/False: a byte consists of eight bits.
2. Mark True/False: a memory cell can store one byte of data.
3. Mark True/False: the write operation stores data in memory.
4. Mark True/False: the read operation erases the data read.
5. Mark True/False: RAM is random-access memory.

Digital Circuits – Integrated Assessment Section

Unit 16 – Digital-to-Analog and Analog-to-Digital Conversion

Pre-Test

1. What is meant by an analog signal?

2. What is meant by a digital signal?

3. A 4-bit DAC has Vinput=3 V, Rf=20 kΩ and R=150 kΩ. Find the analog output for inputs 0000, 0001, 0011 and 1111.

4. A 5-bit DAC has Vinput=10 V, Rf=20 kΩ and R=150 kΩ. Find the analog output for input 10101.

1. **Rationale**

The counter is one of the basic circuits in the work of most digital devices and systems.
Therefore, these lectures are designed in order for the student to learn about the idea of working and how to design the different counters (ascending ripple counter - descending ripple counter - decimal ripple counter, ascending rippling counter - descending and successive synchronous counter, parallel synchronous counter - binary divider for the number (6) and a binary divider for the number (5)) and on the common integrated circuits of the meters.

C. Central Ideas

First: a general idea of meters.
Second: Ascending ripple counter.
Third: Descending ripple counter.
Fourth: Decimal ripple counter.
Fifth: Ascending-descending ripple counter.
Sixth: Consecutive synchronous counter.
Seventh: Parallel synchronous counter.
Eighth: Binary divider for number (6), binary divider for number (5), common integrated circuits for meters.

D. Objectives of the lecture

After studying this lecture, the student will be able to:
- Recognize the types of meters.
- Designs the circle of the upward ripple counter and writes its realistic table
- Designs the circle of the descending ripple counter and writes its realistic table.
- Designs the circle of the decimal undulating counter and writes its realistic table.

- Recognize the two types of ascending-descending ripple counter and consecutive synchronous counter.
- Designs the circle of the upward - descending ripple counter and writes its realistic table.
- Designs the circuit of the sequential synchronous counter and writes its realistic table.

- Identify the types of meters: parallel synchronous - binary divider for the number (6) - double divider for the number (5) and how to design them and the common integrated circuits of meters.

- Designs a circle of binary divider for the number (6) and writes a table of realism.

- Designs a circle divided binary for the number (5) and writes a table of realism.

- Identify the types of common integrated circuits for meters.

Lecture Eight

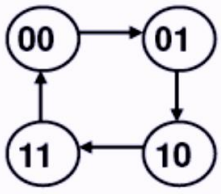
Counters

- Counters are circuits that cycle through a specified number of states.
- Two types of counters:
  - synchronous (parallel) counters
  - asynchronous (ripple) counters
- Ripple counters allow some flip-flop outputs to be used as a source of clock for other flip-flops.
- Synchronous counters apply the same clock to all flip-flops.

Synchronous (Parallel) Counters:

- Synchronous (parallel) counters: the flip-flops are clocked at the same time by a common clock pulse.

- Example: 2-bit synchronous binary counter (using T flip-flops, or JK flip-flops with identical J,K inputs).

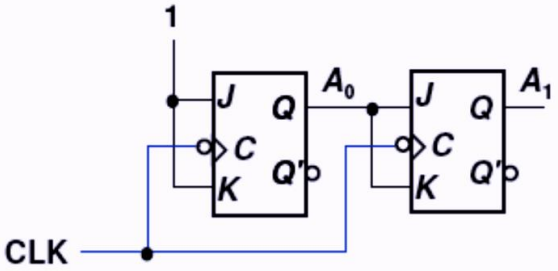


Present state		Next state		Flip-flop inputs	
A_1	A_0	A_1^+	A_0^+	TA_1	TA_0
0	0	0	1	0	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	0	0	1	1

Present state		Next state		Flip-flop inputs	
A_1	A_0	A_1^+	A_0^+	TA_1	TA_0
0	0	0	1	0	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	0	0	1	1

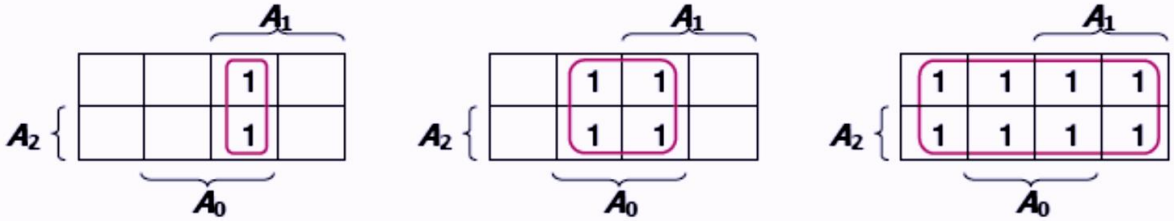
$TA_1 = A_0$

$TA_0 = 1$

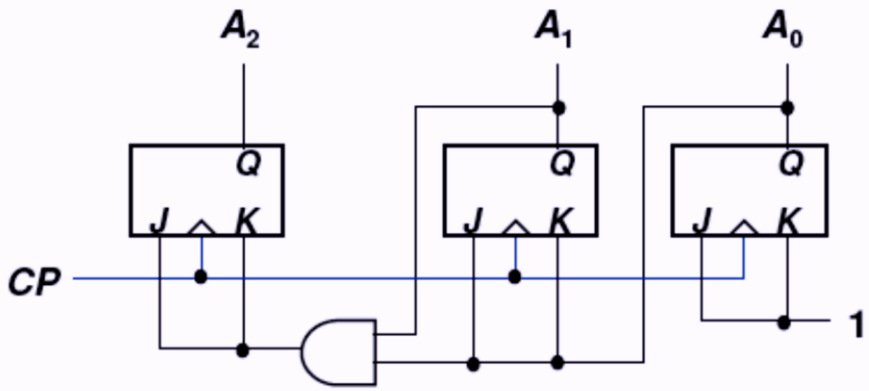


Example: 3-bit synchronous binary counter (using T flip-flops, or JK flip-flops with identical J, K inputs).

Present state			Next state			Flip-flop inputs		
A_2	A_1	A_0	A_2^+	A_1^+	A_0^+	TA_2	TA_1	TA_0
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1
0	1	0	0	1	1	0	0	1
0	1	1	1	0	0	1	1	1
1	0	0	1	0	1	0	0	1
1	0	1	1	1	0	0	1	1
1	1	0	1	1	1	0	0	1
1	1	1	0	0	0	1	1	1



$TA_2 = A_1.A_0$ $TA_1 = A_0$ $TA_0 = 1$



Note that in a binary counter, the n^{th} bit (shown underlined) is always complemented whenever

$\underline{0}11\ldots11 \rightarrow \underline{1}00\ldots00$

or $\underline{1}11\ldots11 \rightarrow \underline{0}00\ldots00$

Hence, X_n is complemented whenever

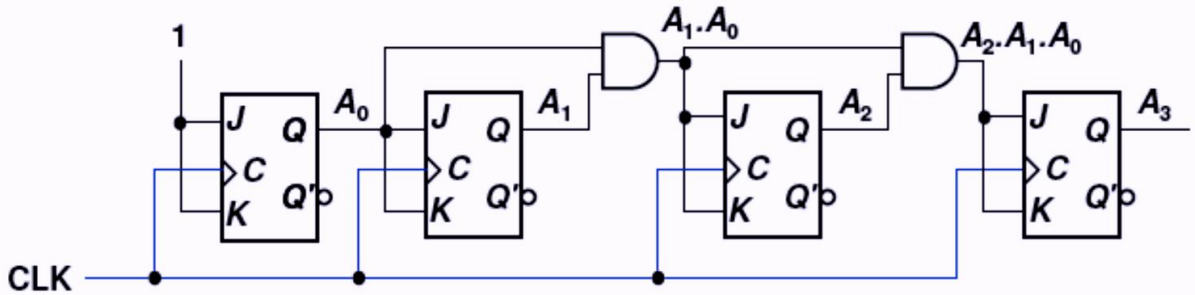
$X_{n-1}X_{n-2} \ldots X_1X_0 = 11\ldots11.$

As a result, if T flip-flops are used, then

$TX_n = X_{n-1} \cdot X_{n-2} \cdot \ldots \cdot X_1 \cdot X_0$

Example: 4-bit synchronous binary counter.

$$TA_3 = A_2 \,.\, A_1 \,.\, A_0$$
$$TA_2 = A_1 \,.\, A_0$$
$$TA_1 = A_0$$
$$TA_0 = 1$$

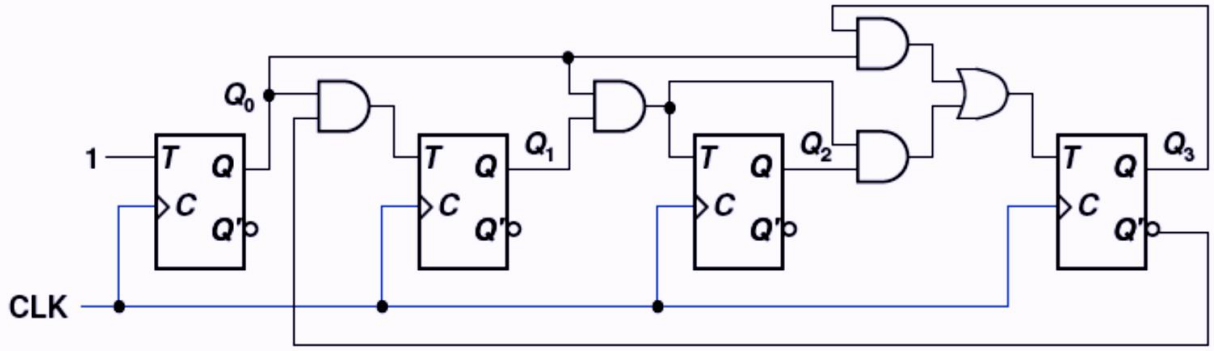


Example: Synchronous decade/BCD counter.

Clock pulse	Q_3	Q_2	Q_1	Q_0
Initially	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10 (recycle)	0	0	0	0

$$T_0 = 1$$
$$T_1 = Q_3'.Q_0$$
$$T_2 = Q_1.Q_0$$
$$T_3 = Q_2.Q_1.Q_0 + Q_3.Q_0$$

$T_0 = 1$
$T_1 = Q_3'.Q_0$
$T_2 = Q_1.Q_0$
$T_3 = Q_2.Q_1.Q_0 + Q_3.Q_0$



Up/Down Synchronous Counter:

- Up/down synchronous counter: a *bidirectional* counter that is capable of counting either up or down.
- An input (control) line *Up*/$\overline{Down}$ (or simply *Up*) specifies the direction of counting.
  - *Up*/$\overline{Down}$ = 1 → Count upward
  - *Up*/$\overline{Down}$ = 0 → Count downward

- Example: A 3-bit up/down synchronous binary counter.

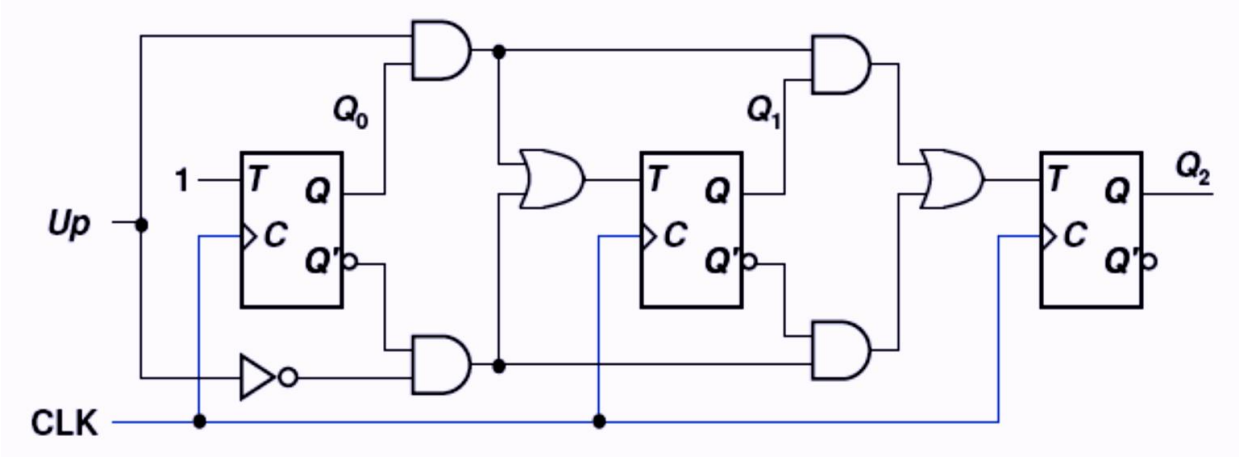
Clock pulse	*Up*	Q_2	Q_1	Q_0	*Down*
0		0	0	0	
1		0	0	1	
2		0	1	0	
3		0	1	1	
4		1	0	0	
5		1	0	1	
6		1	1	0	
7		1	1	1	

$TQ_0 = 1$

$TQ_1 = (Q_0.Up) + (Q_0'.Up')$

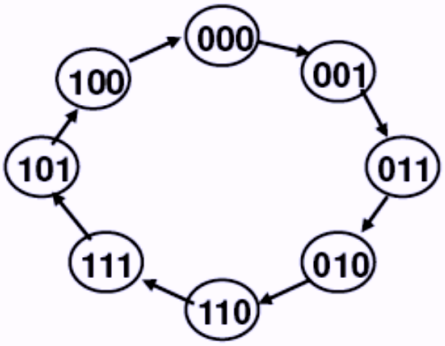
$TQ_2 = (Q_0.Q_1.Up) + (Q_0'.Q_1'.Up')$

Up counter	Down counter
$TQ_0 = 1$	$TQ_0 = 1$
$TQ_1 = Q_0$	$TQ_1 = Q_0'$
$TQ_2 = Q_0.Q_1$	$TQ_2 = Q_0'.Q_1'$



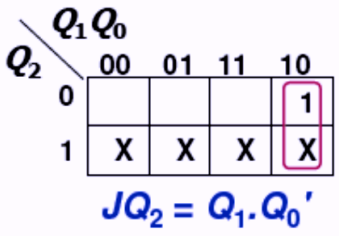
Designing Synchronous Counters:

- Example: A 3-bit Gray code counter (using JK flip-flops).

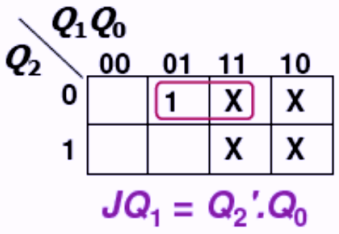


Present state			Next state			Flip-flop inputs					
Q_2	Q_1	Q_0	Q_2^+	Q_1^+	Q_0^+	JQ_2	KQ_2	JQ_1	KQ_1	JQ_0	KQ_0
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	0	1	1	0	X	1	X	X	0
0	1	0	1	1	0	1	X	X	0	0	X
0	1	1	0	1	0	0	X	X	0	X	1
1	0	0	0	0	0	X	1	0	X	0	X
1	0	1	1	0	0	X	0	0	X	X	1
1	1	0	1	1	1	X	0	X	0	1	X
1	1	1	1	0	1	X	0	X	1	X	0

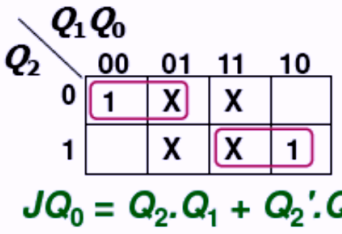
- 3-bit Gray code counter: flip-flop inputs.



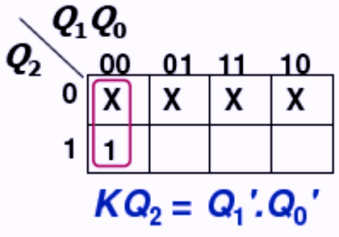
$JQ_2 = Q_1.Q_0'$



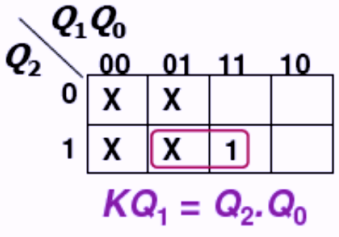
$JQ_1 = Q_2'.Q_0$



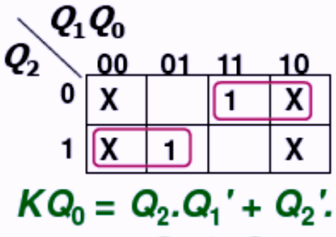
$JQ_0 = Q_2.Q_1 + Q_2'.Q_1'$
$= (Q_2 \oplus Q_1)'$



$KQ_2 = Q_1'.Q_0'$



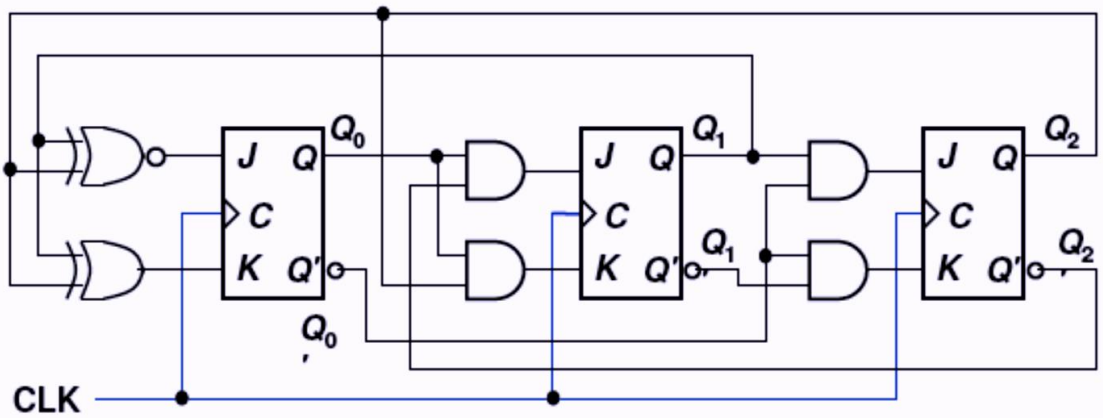
$KQ_1 = Q_2.Q_0$



$KQ_0 = Q_2.Q_1' + Q_2'.Q_1$
$= Q_2 \oplus Q_1$

- 3-bit Gray code counter: logic diagram.

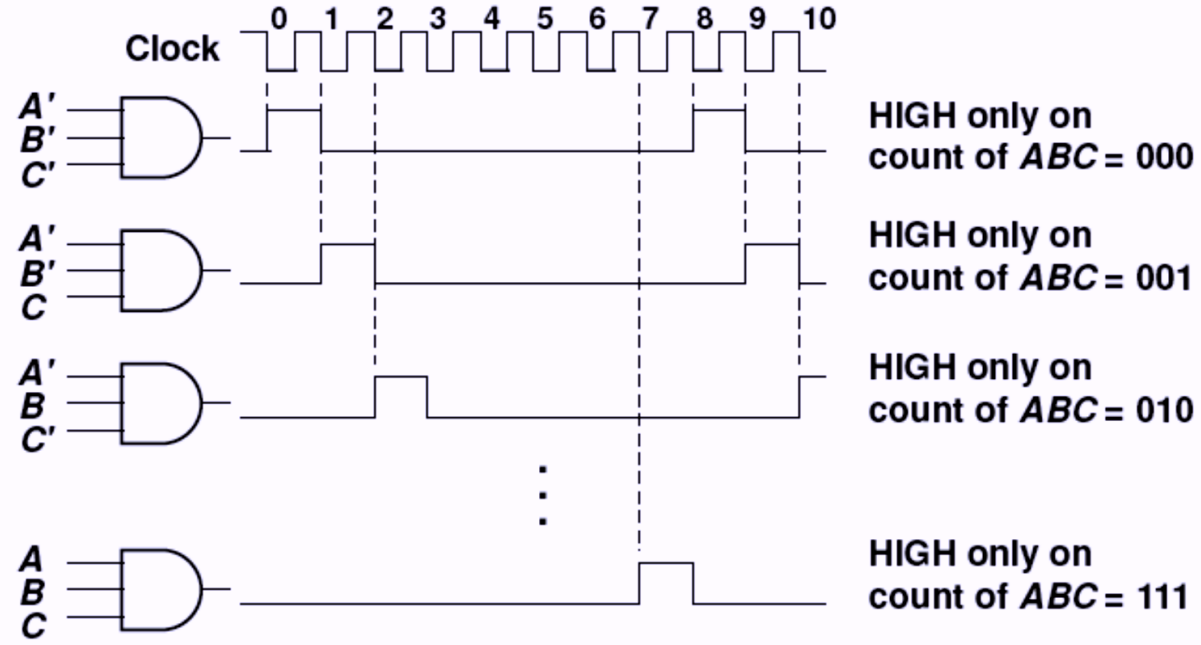
$JQ_2 = Q_1.Q_0'$ $JQ_1 = Q_2'.Q_0$ $JQ_0 = (Q_2 \oplus Q_1)'$
$KQ_2 = Q_1'.Q_0'$ $KQ_1 = Q_2.Q_0$ $KQ_0 = Q_2 \oplus Q_1$



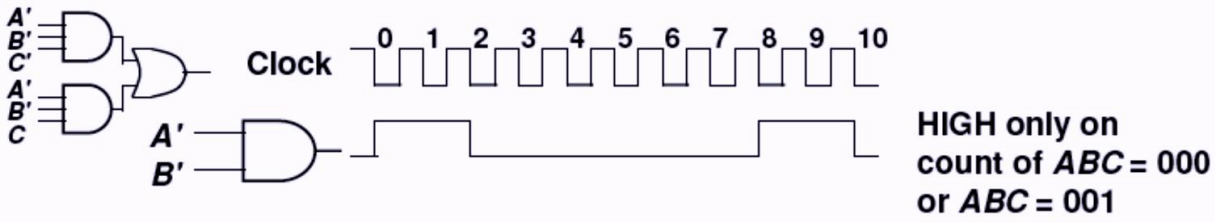
Decoding a Counter:

- Decoding a counter involves determining which state in the sequence the counter is in.
- Differentiate between *active-HIGH* and *active-LOW* decoding.
- Active-HIGH decoding: output HIGH if the counter is in the state concerned.
- Active-LOW decoding: output LOW if the counter is in the state concerned.

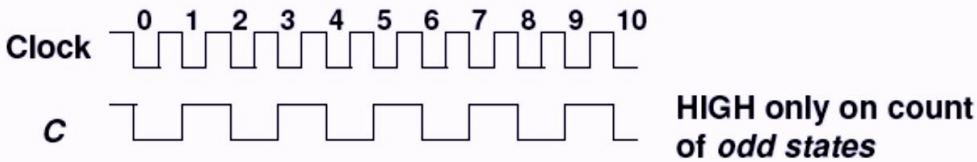
Example: MOD-8 ripple counter (active-HIGH decoding).



- Example: To detect that a MOD-8 counter is in state 0 (000) or state 1 (001).



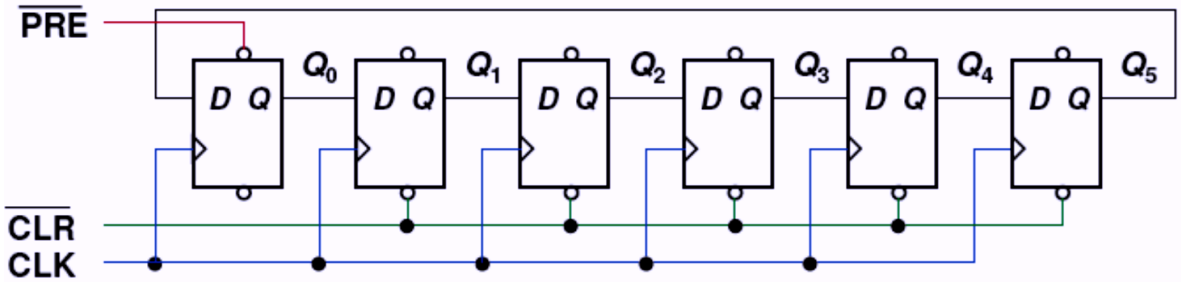
- Example: To detect that a MOD-8 counter is in the odd states (states 1, 3, 5 or 7), simply use *C*.



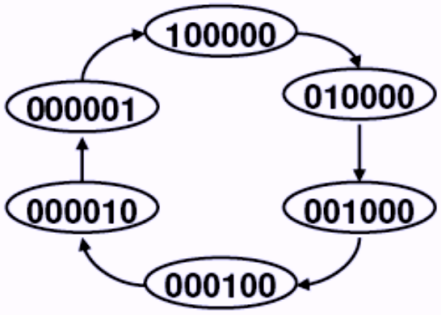
Ring Counters:

- One flip-flop (stage) for each state in the sequence.
- The output of the last stage is connected to the D input of the first stage.
- An *n*-bit ring counter cycles through *n* states.
- No decoding gates are required, as there is an output that corresponds to every state the counter is in.

Example: A 6-bit (MOD-6) ring counter.



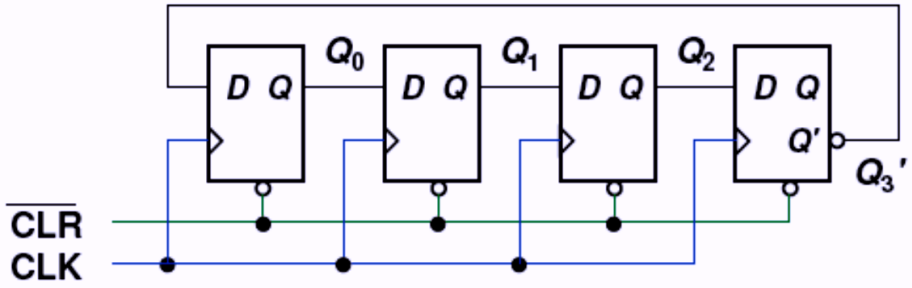
Clock	Q_0	Q_1	Q_2	Q_3	Q_4	Q_5
→0	1	0	0	0	0	0
1	0	1	0	0	0	0
2	0	0	1	0	0	0
3	0	0	0	1	0	0
4	0	0	0	0	1	0
5	0	0	0	0	0	1



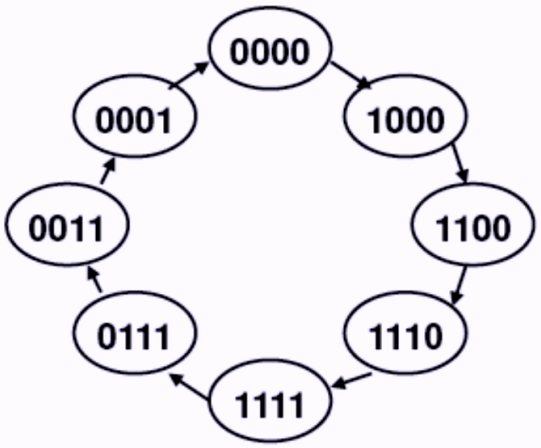
Johnson Counters:

- The complement of the output of the last stage is connected back to the D input of the first stage.
- Also called the *twisted-ring counter*.
- Require fewer flip-flops than ring counters but more flip-flops than binary counters.
- An *n*-bit Johnson counter cycles through 2*n* states.
- Require more decoding circuitry than ring counter but less than binary counters.

Example: A 4-bit (MOD-8) Johnson counter.

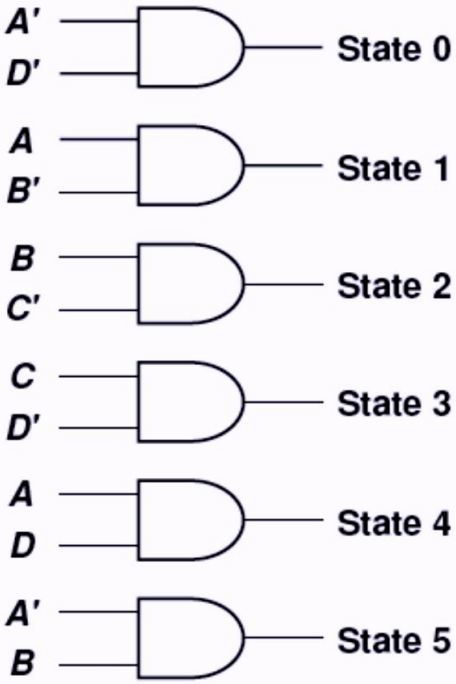


Clock	Q_0	Q_1	Q_2	Q_3
→0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1



Decoding logic for a 4-bit Johnson counter.

Clock	A	B	C	D	Decoding
→0	0	0	0	0	A'.D'
1	1	0	0	0	A.B'
2	1	1	0	0	B.C'
3	1	1	1	0	C.D'
4	1	1	1	1	A.D
5	0	1	1	1	A'.B
6	0	0	1	1	B'.C
7	0	0	0	1	C'.D



Post-Test

1. What type of logic circuits are counters?
2. What is a counter called when all flip-flops receive the synchronizing pulses at the same time?
3. In an asynchronous counter, are the clock inputs connected in parallel or serially?
4. In a synchronous counter, are the clock inputs connected in parallel or serially?
5. Design an asynchronous up counter with modulus 8.
6. Design an asynchronous up counter with modulus 6.

Post-Test

1. Draw a four-stage serial-in/serial-out shift register moving data to the right.

2. Draw the same type moving data to the left.

3. Mark True/False: a four-stage shift register can store up to 15; a Johnson counter is a special shift-register type; an 8-bit Johnson counter has modulus 8; a shift register can be used as a time delay.

Post-Test

1. A memory has 1024 addresses and each address stores one byte. How many bits are in the memory? A) 1024 B) 8192 C) 8 D) 4096.

2. A 32-bit word consists of: A) 2 bytes B) 4 nibbles C) 4 bytes D) 3 bytes.

3. Data are stored in RAM during: A) reading B) enable C) writing D) addressing.

4. Data stored at a RAM address are lost when: A) power is interrupted B) the data are read C) new data are written there D) both A and C.

5. ROM is: A) nonvolatile memory B) volatile memory C) read/write memory D) random-access memory.

6. A memory containing 256 addresses needs: A) 256 address lines B) 6 address lines C) 1 address line D) 8 address lines.

7. A static RAM storage cell consists of: A) flip-flop B) capacitor C) fuse D) transistor.

8. DRAM should be: A) replaced periodically B) refreshed periodically C) permanently enabled D) programmed every time it is used.

9. Flash memory is: A) volatile B) read-only C) read/write D) nonvolatile E) A and C F) C and D.

Post-Test

1. Calculate the output voltage of a 5-bit resistor-ladder DAC for inputs 01101, 10011 and 11111 when logic 0=0 V and logic 1=8 V.

2. For a four-bit resistor network with maximum output 5 V, what is the accuracy, and which is more suitable: 0.5 V or 0.1 V?

3. Calculate the error percentage for the above converter.

4. Using an 8-bit ADC with a clock-cycle time of 1.5 μs, calculate conversion time when the input voltage is one quarter of the maximum value.

5. Calculate the conversion rate of the previous converter.

Answer Key

Pre-Test: 1-C, 2-B, 3-A, 4-D, 5-A, 6-C, 7-B, 8-D. Post-Test answers: 1) 54.78125; 2) (2673.24)■; 3) (1011101101.10010)■; 4) (6E6.D58)■■; 5) (111 110 000 101 . 100 010 000 011)■; 6) (101101)■; 7) (-1001111)■; 8) (1100001)■; 9) (-101110)■; 10) (1010001)■. Source answer key retained; no correction applied.

Note: This assessment material is translated and inserted from the source assessment package. The original Zainab Noman Hamdi course-pack content, terminology, logo, name, and order are not replaced or rewritten.

Answer Key

Pre-Test: 1-C, 2-B, 3-A, 4-A. Post-Test: 1-B (OR), 2-D (XNOR), 3-A (NOR), 4-D (AND).

Note: This assessment material is translated and inserted from the source assessment package. The original Zainab Noman Hamdi course-pack content, terminology, logo, name, and order are not replaced or rewritten.

Answer Key

Answer Key: the source table is largely blank for this unit. The source explicitly records Y = A + BC for the corresponding simplification exercise. Graphical/overbar-dependent items are left unchanged rather than being reconstructed.

Note: This assessment material is translated and inserted from the source assessment package. The original Zainab Noman Hamdi course-pack content, terminology, logo, name, and order are not replaced or rewritten.

Answer Key

Answer Key: the source answer table is incomplete; graphical expressions are not replaced by invented equations. The original figures in the package remain the authority.

Answer Key

Pre-Test source key: Q1=3, Q2=2, Q3=4. Post-Test source key: Q1=C, Q2=B, Q3=C. Remaining source fields are blank.

Note: This assessment material is translated and inserted from the source assessment package. The original Zainab Noman Hamdi course-pack content, terminology, logo, name, and order are not replaced or rewritten.

Answer Key

Pre-Test source key: Q1=A, Q2=B; remaining fields are blank. Post-Test source records: Q2 = CPU, display; Q3=A′BC′D; Q4=A′BCD′; Q5=A′BCD; Q6=AB′C′D. These source expressions are preserved.

Note: This assessment material is translated and inserted from the source assessment package. The original Zainab Noman Hamdi course-pack content, terminology, logo, name, and order are not replaced or rewritten.

Answer Key

Pre-Test source key: Q1=C, Q2=A. Post-Test source answers recorded: Q2 = keyboard, CPU; Q3 = 3,2,1; Q4 = 4,1.

Note: This assessment material is translated and inserted from the source assessment package. The original Zainab Noman Hamdi course-pack content, terminology, logo, name, and order are not replaced or rewritten.

Answer Key

Pre-Test source key: C, A, D, C. Post-Test source key: C, D, B.

Note: This assessment material is translated and inserted from the source assessment package. The original Zainab Noman Hamdi course-pack content, terminology, logo, name, and order are not replaced or rewritten.

Answer Key

Pre-Test source key: C, A, D, C. Post-Test source key: C, D, B.

Note: This assessment material is translated and inserted from the source assessment package. The original Zainab Noman Hamdi course-pack content, terminology, logo, name, and order are not replaced or rewritten.

Answer Key

Pre-Test source key: C, B, B, C. Post-Test source key: Q1=A; Q2=(0010)■; Q3=(11000)■; Q4=B.

Note: This assessment material is translated and inserted from the source assessment package. The original Zainab Noman Hamdi course-pack content, terminology, logo, name, and order are not replaced or rewritten.

Answer Key

Pre-Test source answers: Q1 = present output depends on previous output plus current inputs; Q2 = (10,01,00), forbidden (11); Q3 = (11,10,01,00); Q4 = first output is the inverse of the second. Post-Test answer fields are blank in the source.

Note: This assessment material is translated and inserted from the source assessment package. The original Zainab Noman Hamdi course-pack content, terminology, logo, name, and order are not replaced or rewritten.

Answer Key

Pre-Test source answer fields are blank. Self-Test source answers include: optional; reverses / counter circuits; same input / storage circuits. Post-Test source: Q1 = undefined, inverse of previous state; remaining answers blank.

Answer Key

Post-Test source answers: Q1 = sequential logic circuits; Q2 = synchronous counter; Q3 = serially; Q4 = in parallel. Q5–Q6 are design questions with no source answer recorded.

Answer Key

Pre-Test source key is incomplete. Self-Test source MCQ key: 1-A, 2-D, 3-C, 4-A, 5-B. True/False source key: True, True, False, True. Post-Test True/False source key: True, True, False, True.

Note: This assessment material is translated and inserted from the source assessment package. The original Zainab Noman Hamdi course-pack content, terminology, logo, name, and order are not replaced or rewritten.

Answer Key

Pre-Test source key: T, F, T, F, F. Self-Test: T, T, T, F, F. Post-Test: 1-B, 2-C, 3-C, 4-D, 5-A, 6-D, 7-A, 8-B, 9-F.

Note: This assessment material is translated and inserted from the source assessment package. The original Zainab Noman Hamdi course-pack content, terminology, logo, name, and order are not replaced or rewritten.

Answer Key

Pre-Test source answers: Q1 = continuously varying signal; Q2 = signal changing between two discrete levels; Q3 = 6 V, 1.2 V, 0.4 V, 0 V; Q4 = 6.77 V. Self-Test source includes Vo=6.77 and the listed resistor-network weights. Post-Test source: 01101→5.5 V; 10011→4.75 V; 11111→7.75 V; accuracy 0.33 V; error 8.6%; Tc=96 (source unit retained); TA=192 (source unit retained).