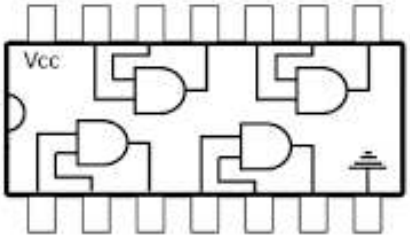
The Public Authority for Applied Education and Training
الهيئة العامة للتعليم التطبيقي و التدريب

The Higher Institute of Telecommunication and Navigation
المعهد العالي للاتصالات و الملاحه

Switching Department
قسم السنترالات

Integrated circuit
الدوائر المتكاملة

SW178



Prepared: Engr. *Naser Alkhaldi*

First edition: *Nov-2019*

اهداء و عرفان

الى الاخ و الزميل الذي تعلمنا منه الكثير و الذي زاملناه سنوات عديده لم يبخل بها علينا بالمشوره و العلم الى من قمنا بتدريس مذكراته الى طلبتنا و التي كانت تمتاز بالعلم و النوعيه ... فلم اجد افضل من ان انقل هنا في هذه المذكره ما كنت ادرب طلبتي عليه من مذكراته و اخص بالذكر مذكرة التي لم تعد تدرس رغم جودة الاعداد EE113 فالكاتب للفصل الاول و الثاني ماهو الا اخي / م.محمد الشيشتاوي الذي عاد الى موطنه بعد سنين عديده بذلها في خدمه الكويت و ابنائها رغم محاولاتي العديده في التواصل معه لابلاغه بذلك فأنني أمل ان يستفاد من علمه و ان يجد ذلك في علم نافع ينتفع به من بعده , فلك كل التحيه و المحبه و العرفان نسطره هنا في هذه المذكره فانت غبت عنا فبعلمك الذي نقلته ها هنا ينتفع به .
و نهدي لك الفصول اللاحقه التي قمنا بصياغتها و تعديلها و تأليفها بما يتناسب مع الحاجه الى التخصصات الجديده عرفانا منا بدورك الكبير في توجيهنا و زمالتك لنا طوال تلك السنين فلك كل الشكر و العرفان

م. ناصر الخالدي

Table of contents :

Introduction to Digital Logic

Digital logic design techniques form the basis of all digital integrated circuits. Understanding the methods and components are critical both to hardware designers but also software developers who will utilize these hardware components. This course starts from first principles of digital information representation and presents the common components and design methodologies needed to design more advanced systems. If you like logic and solving puzzles and want to understand how computers and other digital systems work, this course is for you!

Mark Redekopp
Associate Professor of Engineering Practice
University of Southern California

In this course we will tech you both the theoretical and practical digital logic gates and IC's we will start with what is the binary system to advance integrated circuit and counters .

For each part of the course we will start with the theory and then we will do together many experiments to strength the the theoretical topics that we had explained before

Course Objective
On completing the course successfully, trainee students will be able to: • Verify An integrated circuit (IC) which sometimes called a chip or microchip, is a semiconductor • Verify the logic gates functions and construct the truth tables for their operations. • Construct combination of logic circuits and verify their truth tables. • Construct counters & registers from flip-flops and verify their operations. • Implement some IC applications, starting from gates by IC's

Course Content
1. Verify An integrated circuit (IC). 2. Verify the logic gates functions. 3. Construct combination of logic circuits. 4. Construct counters & registers from flip-flops. 5. Implement some IC applications, starting from gates by IC's.

الخطة الدراسية للمقرر العملي
Practical course Lesson Plan

switching	القسمDept.	Integrated circuit	اسم المقرر Course Name
4	عدد الساعات الأسبوعيةWeekly Hrs.	SW178	رمز المقررCourse Code

Weeks	Content
1	**INTRODUCTION TO DIGITAL CONCEPTS And systems**
2	**Number system (decimal and binary)**
3	**Decimal to binary , addition and subtraction**
4	**Boolean functions[NOT,AND AND OR] WITH EXPERMENTS**
5	**Boolean functions[NAND & NOR] WITH EXPERMENTS**
6	**Boolean algebra [communicative laws, Associative laws ,]**
7 Mid-term Exams	**Boolean algebra[distributive laws and absorption]**
8 Mid-term Exams	**REVISION Midterm exam**
9 Mid-term Exams	**Negation laws(de Morgan law)**
10	**Additional Boolean functions [EXOR AND EXNOR]**
11	Realizing gates by combining NAND / NOR gates
12	Fault simulation experiments
13	Jk flip- flop
14	**Final Review**
15	**موعد بداية الاختبارات النهائية Final Exams**
16	

أهداف المقررCourse Objectives

- Verify An integrated circuit (IC) which sometimes called a chip or microchip, is a semiconductor
- Verify the logic gates functions and construct the truth tables for their operations.
- Construct combination of logic circuits and verify their truth tables.
- Construct counters & registers from flip-flops and verify their operations.
- Implement some IC applications, starting from gates by IC's

موعد اختبار منتصف الفصل
Mid-term Exam Date
/ / 20
يبلغ المدرب جميع المتدربين بالموعد

توزيع الدرجات الكلية والتي مجموعها درجة100
Grading Policy (Total of 100)

الاختبار الفصلي Final Exam	**40%**
اختبار منتصف الفصل Mid-term Exam	**30%**
الاختبارات القصيرة Quizzes	**10%**
التطبيقات و النشاط Applications & Classroom/Lab Activities	**10%**
الانتظام Attendance	**10%**

ختم القسم
Dept. Seal

Chapter 1

Introduction to digital concepts

1.1 Introduction

The term digital is derived from the way computers perform operations, by counting digits. For many years, applications of digital electronics were confined to computer systems.

Today, digital technology is applied in a wide range of areas in addition to computers. Such applications as television, communication systems, radar, navigation and guidance systems, military systems, medical instrumentation, industrial process control, and consumer electronics use digital techniques.

Digital technology has progressed from vacuum tube circuits to discrete transistors to complex integrated circuit, some of which contain millions of transistors. This chapter introduces you to digital electronics and provides a broad overview of many important concepts.

1. 2: ANALOG AND DIGITAL SIGNALS

Electronic circuits can be divided into two broad categories: **digital** and analog.

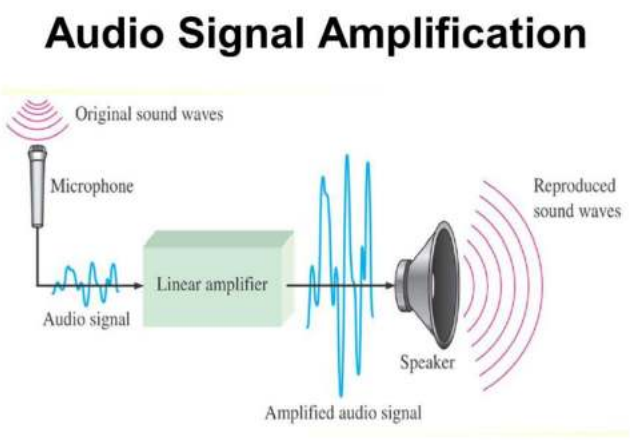
Digital electronics involves quantities with discrete values, and analog electronics involves quantities with continuous values.

1·2·1 : ANALOG QUANTITY

An analog quantity is one having continuous values, most things that can be measured quantitatively appear in nature in analog form. For example, the air temperature change over a continuous range of values during a given day, the temperature does not go from 70° to 71^0 instantaneously; it takes on all the infinite values in between. If you graphed the temperature on a typical summer day, you would have a smooth, continuous curve. Other examples of analog quantities are time, pressure, distance, and sound.

An Analog Electronic System

A public address system, used to amplify sound so that it can be heard by a large audience, is one example of an application of analog electronics. The basic diagram in Figure 1-1 illustrates that sound waves, which analog in nature, are picked up by a microphone and converted to a small analog voltage called audio signal. This voltage varies continuously as the volume and frequency of the sound changes and is applied to the input of a linear amplifier.
The output of the amplifier, which is an increased reproduction of the input voltage, goes to the speaker(s). The speaker changes the amplified audio signal back to sound waves that have much greater volume than the original sound waves picked up by the microphone.



Amplified audio signal

Figure 1-1

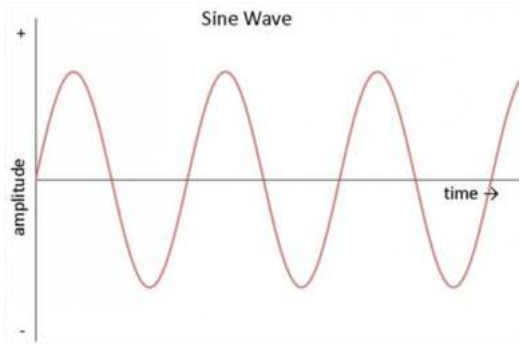
A basic public address system.

1. 2-2: EXAMPLES OF ANALOG SIGNALS

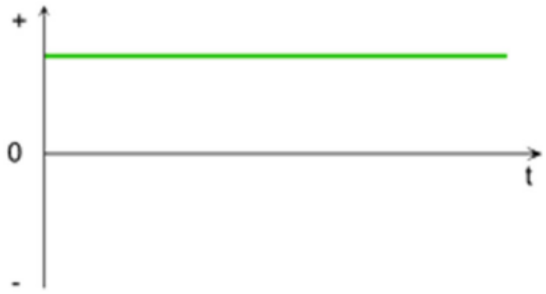
Analog signal is as DC or AC voltage or current that varies smoothly and continuously, as shown in Figure 1-2, in the following examples:

1) Sine wave.
2) Fixed DC voltage.
3) Random AC voltage.

1



2 – positive DC



3- random AC voltage

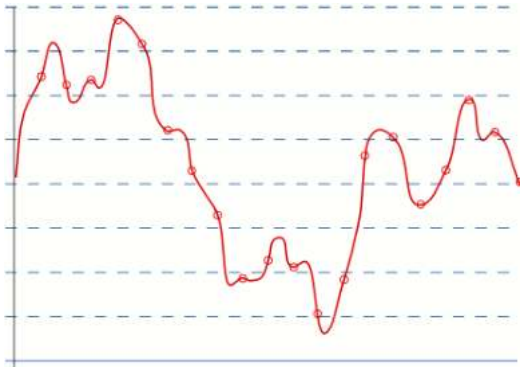


Fig 1-2
Examples of analog signals

1·2·3: DIGITAL QUANTITY

Digital electronics involves circuits and systems in which there are only two possible states. These states are represented by two different voltage levels: A **HIGH** and a LOW. The two states can be also represented by current levels, open and closed switches, or lamps turned on and off.

In digital systems such as computers, combinations of the two states, called codes, are used to represent numbers, symbols, alphabetic characters, and other types of information. The two-state number system is called binary, and its two digits are 0 and 1. A binary digit is called a bit.

1·2·4: THE DIGITAL ADVANTAGES

Digital representation has certain advantages over analog representation in electronics applications. For one thing, digital data can be processed and transmitted more efficiently and reliably than analog data. Also, digital data has a great advantage when storage is necessary. For example, music when converted to digital form can be stored more compactly and reproduced with greater accuracy and clarity than is possible when it is in analog form. Noise (unwanted voltage fluctuations) does not affect nearly as much as it does analog signals.

1·2·5: EXAMPLES OF DIGITAL SIGNALS

Digital signals are essentially a series of voltage pulses that usually switch between to fixed *levels.* Figure 1-3 shows some examples of the digital signals. ~

1) Positive pulses.
2) Negative pulses.
3) Pulses with one positive level and the other negative level.

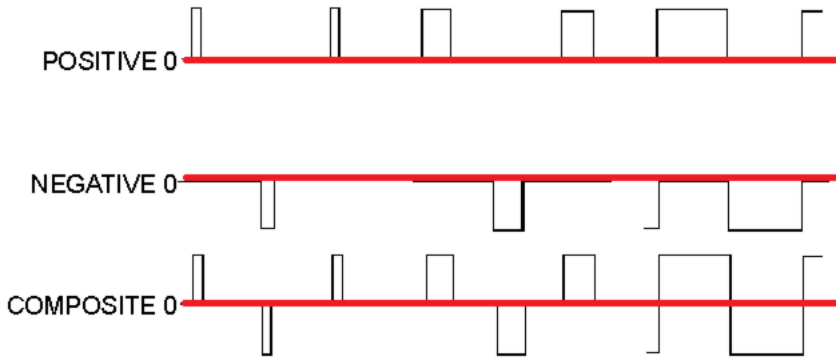


Fig 1-3
Examples of digital signals

1·3: BINARY DIGITS, LOGIC LEVELS, AND DIGITAL WAVEFORMS

1 - 3-1 : BINARY DIGITS

The two digits in the binary system, 1 and 0, are called bits, which is a contraction words binary digits. In digital circuits, two different voltage levels are used to represent the two bits. Generally, 1 is represented by the higher voltage, which we will refer to as a HIGH, and 0 is represented by the lower voltage level, which we will refer as a LOW. This is called positive logic and will be used throughout this book.

HIGH = 1 and LOW = 0

Another system in which a 1 is represented by a LOW and a 0 is represented by a HIGH is called negative logic.

Groups of bits (combinations of Is and Os), called codes, are used to represent numbers, letters, symbols, instructions, and anything else required in a given application.

1-3-2: LOGIC LEVELS

The voltages used to represent a 1 and a 0 are called logic levels. Ideally, one voltage level represents a HIGH and another voltage level represents a LOW.

In a practical digital circuit, however, a HIGH can be any voltage between a specified minimum value and a specified maximum value. Likewise, a LOW can be any voltage between a specified minimum and a specified maximum. There can be no overlap between the accepted HIGH levels and accepted LOW levels.

1·3·3: DIGITAL WAVEFORMS

Digital waveforms consist of voltage levels that are changing back and forth between the HIGH and LOW levels or states. Figure 1-4(a) shows that a single positive-going pulse is generated when the voltage (or current) goes from its normally LOW level to its HIGH level and then back to its LOW level.
The negative-going pulse in Figure 1-4(b) is generated when the voltage *goes* from its normally HIGH level to its LOW level and back to its HIGH level. A digital waveform is made up of a series of pulses.

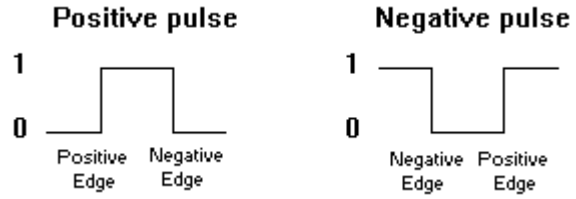


Fig 1-4
Ideal pulse

1·3·4: THE PULSE

As indicated in Figure 1-4, the pulse has two edges: a leading edge that occurs first at to and a trailing edge that occurs at time t_1. For a positive-going pulse, the leading edge is a rising edge, and the trailing edge is a falling edge.

The pulse in Figure 1-4 are ideal because the rising and falling edges are assumed to change in zero time (instantaneously). In practice, these transitions never occur instantaneously, although for most digital work you can assume ideal pulses.

Figure 1-5 shows a non ideal pulse. The time required for the pulse to go from its LOW level to its HIGH level is called the rise time *(t_r)* and the time required for the transition from the HIGH level to the LOW level is called the fall time *(tf)*.

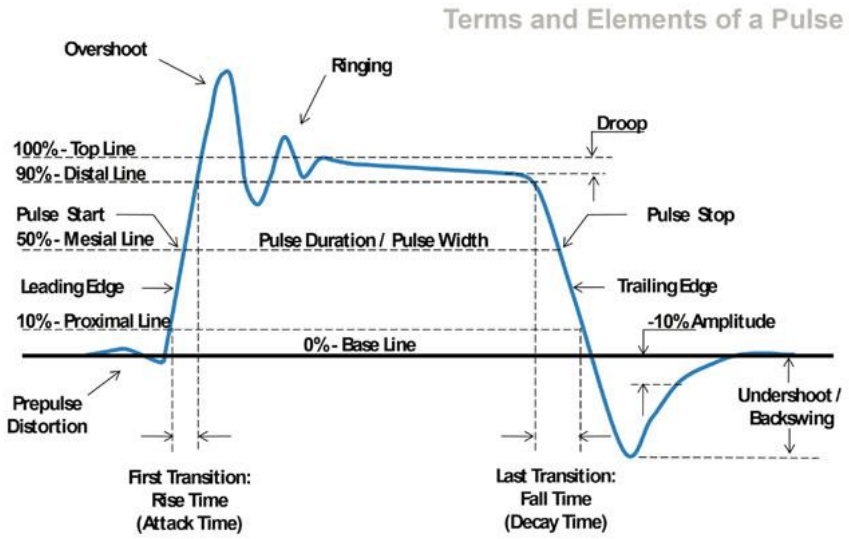
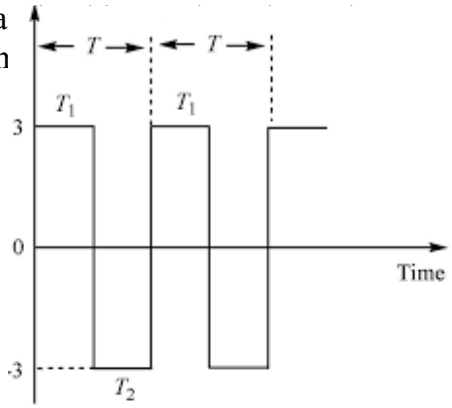


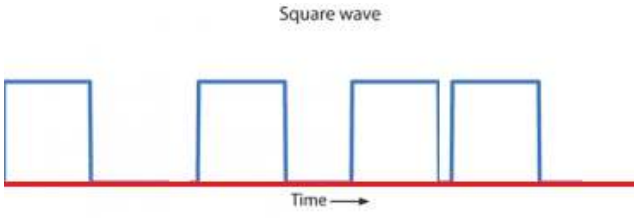
Fig 1-5
Non ideal pulse characteristics

In practice, it is common to measure rise time from *10%* of the pulse amplitude (height from baseline) to *90%* of the amplitude and to measure the fall time from *90%* to *10%* of the pulse amplitude, as indicated in Figure 1-5. The bottom *10%* and the top *10%* of the pulse are not included in the rise and fall times because the nonlinearities in the waveform in these areas. The pulse width *(tw)* is a measure of the duration of the pulse and is often defined as the time interval between *50%* points on the rising and falling edges, as indicated in Figure 1-5.

1·3·5: WAVE CHARACTERISTICS

Most waveforms encountered in digital systems are composed of series of pulses, sometimes called pulse trains, and can be classified as either periodic or non periodic. A periodic pulse waveform is one that repeats itself at fixed interval, called period *(T)*. The frequency *(f)* is the rate at which it repeats itself and measured in hertz (Hz). A non periodic pulse waveform, of course, does not repeat itself a sed of pulses of randomly differing pulse widths and/or randon the pulses. An example of each type is shown in Figure 1-6.





(a)Periodic pulse (square wave) (b) non periodic pulse

Figure 1-6
Examples of digital waveforms.

The frequency *(f)* of a pulse (digital) waveform is the reciprocal of the period. The relationship between the frequency and period is expressed as follows:

$$f=1/T$$

$$T=1/f$$

An important characteristic of a periodic digital waveform is its duty cycle. The duty cycle is the ratio of the pulse width(t_w) to the period *(T)* and can be expressed as percentage.

$$Duty\ Cycle\% = (tw/T) \times 100$$

EXAMPLE 1·1:

A portion of a periodic digital waveform is shown in Figure 1-7. The measurements are in milliseconds. Determine the following:
(a) period (b) frequency (c)duty cycle.

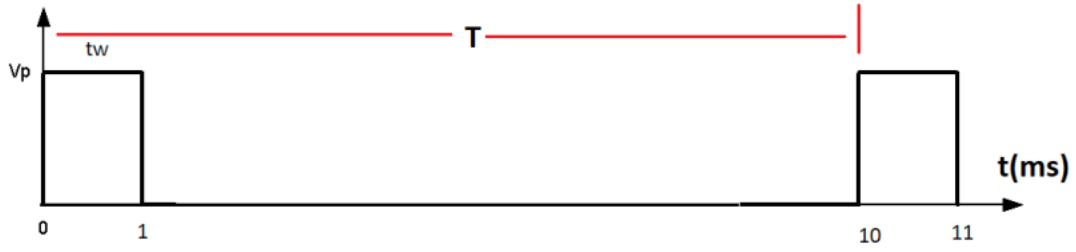


Figure 1-7

SOLUTION:

(a) The period is measured from the edge of one pulse to the corresponding edge of the next pulse. In this case T is measured from leading edge to leading edge, as indicated.

T = 10 ms.

(b) $f = 1/T = 1/(10 \times 10^{-3}) = 100$ Hz.

(c) $Duty\ Cycle\% = tw/T \times 100 = (1ms/10\ ms) \times 100 = 10\%$

1·3·6: A DIGITAL INFORMATION CARRIES BINARY INFORMATION

Binary information that is handled by digital systems appears as waveforms that represent sequences of bits. When the waveform is HIGH, a binary 1 is present; when the waveform is LOW, a binary 0 is present. Each bit in a sequence occupies a defined time interval called bit time. The Clock

In digital systems, all waveforms are synchronized with a basic timing waveform called the clock. The clock is a periodic waveform in which each interval between pulses (the period) equals the time for one bit.

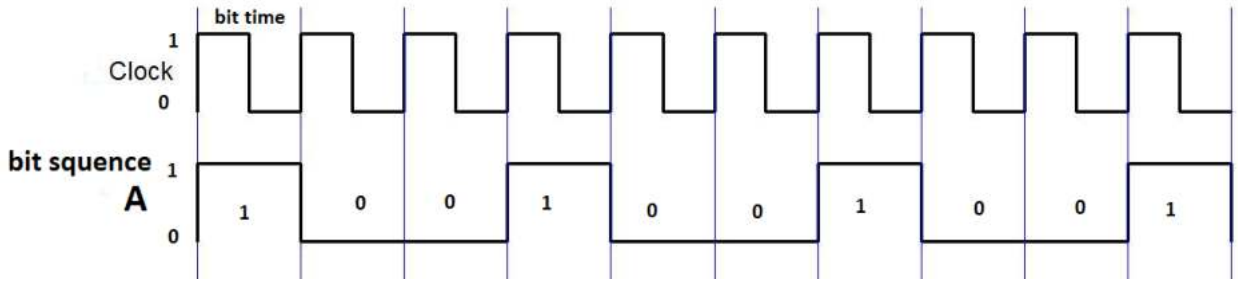


Fig 1-8

Clock waveform synchronized with a waveform representation of a sequence of bits.

An example of a clock waveform is shown in Figure. 1-8. Notice that, in this case, each change in level of waveform *A* occurs at the trailing edge of the clock. During each bit of the clock, waveform *A* is either HIGH or LOW. These HIGHs and LOWs represent a sequence of bits as indicated. A group of several bits can be used as a piece of binary information, such as a number or a letter. The clock waveform itself does not carry information.

1·3-7: TIMING DIAGRAMS

A timing diagram is a graph of digital waveforms showing the actual time relationship of two or more waveforms and how each waveform changes in relation to the others. Figure 1-8 is an example of a simple diagram that shows how the clock waveform and waveform *A* are related on a time base. By looking at timing diagram, you can determine the states (HIGH or LOW) of all the
waveforms at any specified point in time and exact time that a waveform changes state relative to the other waveforms.

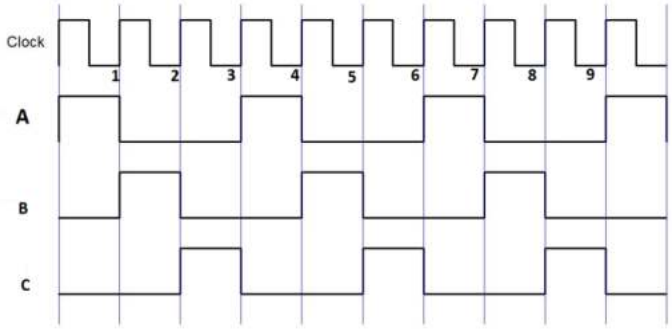


Figure 1-9

Example of a time diagram

Figure 1-9 is an example of a timing diagram made up of four waveforms. From this timing diagram you can use see.

1·4: DIGITAL TECHNIQUES IN DIFFERENT FIELDS

The greatest use of digital techniques today is in computers. Digital computers are used in all areas of business, industry, education, etc. Digital techniques are not only used in computer field but in almost every area of
electronics, let us *see* a few examples:

1·4-1: COMPUTER

- Computers are very useful machines that can save man's energy and efforts and give various capabilities.
- Many years of technological developments have given digital computers greater capability and have become smaller, cheaper and easier to use.
- Advanced semiconductor technology has developed microprocessors, an IC (Integrated Circuit) that puts most of the minicomputer stages into a single chip. Due to this microcomputers are within reach of almost everyone.

1·4-2: COMMUNICATIONS

- In the past, transmission of information over wire lines or by radio was done by analog methods. Now-a-days much data is transmitted in digital form instead of analog form.
- It has been found that pulse type or digital signals are easier to work with and are less susceptible to noise and other problems.
- Digital computers can readily communicate with one another by transmitting
- information over the telephone lines using digital techniques.

1·4-3: TELEMETRY

- These systems used for transmitting measurements data from a remote (far distance) location, use digital techniques very much.
- In an unmanned satellite, for example: Sensors are used to measure and show different environmental conditions such as Temperature, light and radiation.
- An improvement in transmission reliability and accuracy can be observed by converting the analog variables like temperature, light etc. into digital signals.
- Because of improved reliability and accuracy digital techniques are preferred for Telemetry systems.

1·4-4: TEST INSTRUMENTS

- Now-a-days, we *see* that testing and measuring equipment also use digital techniques.
- Digital techniques in testing and measuring instruments give increased precision in measurements and increased convenience of direct decimal display.
- Because of digital techniques, it is possible to interface many digital instruments with a computer. This allows automatic monitoring, controlling, measuring and recording of data.
- Digital Millimeter (DMM) and electronic counter are the instruments widely used which give greater accuracy and less error in measurement. These instruments display the exact quantity measured in convenient read-out form.

1·4-5: INDUSTRIAL CONTROLS

- In manufacturing plants or refineries where complex operations must be accurately controlled, digital techniques are widely used.
- These systems use sensors to get information about different stages of operation. The outputs of these sensors produce signals that will control the different stages of operation.
- The popular industrial application of digital techniques is machine tool control. Here digital computer controls the drilling, cutting, punching and stamping of materials to produce specific metal parts accurately and automatically.

1 -4-6: CONSUMER ELECTRONIC EQUIPMENT

- With the drop in prices of digital circuits, the common consumer items like radio, TV, Hi-fi systems and other products has given way for digital techniques.
- These days, it is difficult to find a consumer electronic equipment without the use of digital techniques.
- In TV, the circuitry for channel selector is digital circuitry.
- In Hi-fi receivers, digital frequency synthesizers are used for tuning.
- Digital clocks, electronic calculators, home digital thermometer are other examples.

SUMMARY:

- An analog quantity has continuous values.
- A digital quantity has a discrete set of values.
- A binary digit is called a bit.
- A pulse is characterized by rise time, tall time, pulse width, and amplitude.
- The frequency of a periodic waveform is the reciprocal of the period. The formula relating frequency and period are:

$$f = 1/T \quad \text{and } T=1/f$$

- The duty cycle of a pulse waveform is the ratio of the pulse width to the period, expressed by the following formula as a percentage:

$$\textit{Duty cycle } \% = tw/T \times 100$$

- A timing diagram is an arrangement of two or more waveforms showing their relationship with respect to time.

Review questions

Q1. Fill the space :

1)The two basic types of the electronic techniques are:…….. and ……..
2) ……., …….. and …….. are examples of the analog signals .
3) …….,……and ……...are examples of digital signals.
4)The electronic circuits that process analog signals are called…… circuits and that process digital signals are called……circuit.
5)The digital signals have ……. States or levels.
6)The greatest use of digital techniques today is in …….. .

Q2. Select the correct answer :

1) A quantity having continuous values is:
   (a) a digital quantity. (b) an analog quantity.
   (c) a binary quantity. (d) a natural quantity.
2) The term **bit** means :
   (a) a small amount of data. (b) a 1 or a 0
   (c) binary digit. (d) both answers (b) and (c).
3) The time interval on the leading edge of a pulse between 10% and 90% of the amplitude is the:
   (a) rise time. (b) fall time.
   (c) pulse width. (d) period.
4) A pulse in a certain waveform occurs every 10 ms. The frequency is:
   (a) 1 kHz. (b) 1 Hz. (c) 100 Hz. (d) 10 Hz.
5)In certain digital waveform, the period is twice of the pulse width The duty cycle is:
   (a) 100%. (b) 200%. (c) 50%. (d) 20%.

Q3 Answer the following questions :

1) Name two advantages of digital data as composed to analog data?
   …….
   …….
2) Name an analog quantity other than temperature and sound ?
   .....
   .....

3) Define the sequence of bits (Is and Os) represented by each of the following sequences of levels:

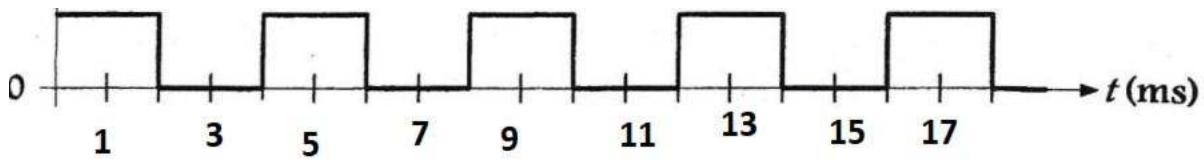
(a) HIGH,HIGH, LOW, HIGH, LOW, LOW, LOW, HIGH
(b) LOW, LOW. LOW, HIGH, LOW, HIGH, LOW, HIGH, LOW

……
……
……
……
……..

4) List the sequences o levels (HIGH and LOW)that represent each of the following bit sequences:

(a) 1011101 (b) 11101001

……
……



5-1) Determine the period of the digital waveform in Figure above ?

……
……
……

5-2) What is the frequency of the waveform in Figure above ?

……
……
……

5-3) Is the pulse waveform in Figure above periodic or non-periodic ?

……
……

5-4) Determine the duty cycle of the waveform in Figure above ?

……
……
……

Chapter 2
Number systems

2 -1: INTRODUCTION

The binary number system and digital codes are fundamental to computers and to digital electronics in general. **In** this chapter, the binary number system and its relationship to decimal is the principal focus .Arithmetic operations with binary numbers are covered to produce a basis for understanding how computers and other types of digital systems work .

Decimal, also called Hindu-Arabic, or Arabic, number system, in mathematics, positional numeral system employing 10 as the base and requiring 10 different numerals, the digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. It also requires a dot (decimal point) to represent decimal fractions. In this scheme, the numerals used in denoting a number take different place values depending upon position. In a base-10 system the number 543.21 represents the sum $(5 \times 10^2) + (4 \times 10^1) + (3 \times 10^0) + (2 \times 10^{-1}) + (1 \times 10^{-2})$.
This number system, with its associated arithmetic algorithms, has furnished the basis for the development of Western commerce and science since its introduction to the West in the 12th century AD.

2-2: DECIMAL NUMBER SYSTEM

You are familiar with decimal number because you use decimal numbers every day. Although decimal numbers are commonplace, their weighted structure is often is often not understood. In this section, the structure of decimal numbers is reviewed. This review will help you more easily understand the structure of binary number system

The decimal number system has ten digits, 0 through 9. Each of the ten digits represents a certain quality. Since there are ten distinct digit used in the decimal number system, therefore, the decimal number system has a base (or radix) of 10, because it used ten (10) digits 0, 1,2,3,4,5,6, 7,8,9.

As you know, the ten symbols (digits) do not limit you to expressing only ten different quantities because you use the various digits in appropriate positions within a number to indicate the magnitude of the quantity.

You can express quantities up through nine before running out of digits; if you wish to express a quantity greater than nine, you will use two or more digits and the position of each digit within the number tells you the magnitude it represents.

If, you for example, you wish to express the quantity five hundred-ninety-two, you use (by their respective positions in the number) the 5 digit to represent the quantity five-hundred, the digit 9 to represent the quantity ninety, and the digit 2 to represent the quantity two. As illustrated below:

$$(5 \times 10^2) + (9 \times 10^1) + (2 \times 10^\circ) = 500 + 90 + 2 = 592$$

The position of each digit in decimal number indicates the magnitude of the quantity represented and can be assigned a weight. The weights for whole numbers are positive powers of ten that increase from right to left, beginning with $10^\circ = 1$.

$$\ldots \ldots \ldots . 10^5\ 10^4\ 10^3\ 10^2\ 10^1\ 10^\circ$$

So, in decimal number system, positional weights are units, tens, hundreds, thousands etc ...

$$10^\circ = 1$$
$$10^1 = 10$$
$$10^2 = 100$$
$$10^3 = 1000$$
$$10^4 = 10000$$

For fractional numbers, the weights are negative powers of ten that decrease from left to right beginning with $10^{-1} = 0.1$

$$10^2\ 10^1\ 10^\circ . 10^{-1}\ 10^{-2}\ 10^{-3} \quad \ldots\ldots\ldots\ldots\ldots\ldots$$

2·3: BINARY NUMBER SYSTEM

The binary number system is simply another way to represent quantities. The binary system is less complicated than the decimal system because it has only two digits. It may seem more difficult at first because it is unfamiliar to you. The decimal system with its ten digits is a base-ten system; the binary system with its two digits is a base-two system. The two binary digits (bits) are 1 and 0. The position of a 1 or 0 in a binary number indicates its weight, or value within the number, just as the position of a decimal digit determines the value of that digit. The weights in binary number are based on power of two.

2·3·1 : COUNTING IN BINARY

To learn to count in the binary system, first look at how you count in the decimal system, you start at zero and count up to nine before run out of digits. You then start another digit position (to the left) and continue counting 10 through 99. At this point you exhausted all two-digit combinations, so a third digit position is needed to count 100 through 999.

A comparable situation occurs when you count in binary, except that you have only two digits, called bits. Begin counting 0, 1. At this point you have used both digits, so include another digit position and continue 10, 11. You have now exhausted all combinations of two digits, *so* a third position is required. With three digit positions you can continue to count 100, 101, 110. Now, you need a fourth digit position to continue, and so on. A binary count of zero through fifteen is shown in Table 2-1. Notice the patterns with which the 1 s and Os alternate in each column.

Decimal Number	Binary Number
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 a 1 0
11	1 a 1 1
12	1 1 0 0
13	1 1 0 1
14	1 1 1 0
15	1 1 1 1

Table 2-1

2·3·2: BINARY NUMBER SIZE

As you have seen in Table 2-1, four bits are required to count from 0 to 15. The value of a bit is determined by its position in the number. In general, with n bits you can count up to a number equal to 2^n - 1.

Largest decimal number $= 2^n - 1$

• For example, five bits (n = 5) you can count from zero to thirty-one.

$2^5 - 1 = 32 - 1 = 31$

With six bits (n = 6) you can count from zero to sixty-three.

$2^6 - 1 = 64 - 1 = 63$

2-3-3: THE WEIGHTING STRUCTURE OF BINARY NUMBERS

A binary number is a weighted number. The right-most bit LSB (least significant bit) in a binary whole number and has a weight of $2^\circ = 1$. The weights increase from right to left by , a power of two for each bit. The left-most bit is the MSB (most significant bit); its weight depends on the size of the binary number. Fractional numbers can also be represented in binary by placing bits to the right of the binary point, just as fractional decimal digits are placed to the right of the decimal point. The left-most bit is the MSB in a binary fractional number and has a weight of $2^{-1} = 0.5$

The fractional weights decrease from left to the right by a negative power of two for each bit. The weight structure of a binary number is:

$$2^{n-1} \ldots\ldots 24\ 2^3\ 22\ 21\ 2^\circ.2^{-1}2^{-2}2^{-3} \ldots\ldots\ldots 2^{-n}$$

Where n is the number of bits from the binary point. Thus, all the bits to the left of the binary point have weights that are positive powers of two, as previously discussed for whole number. All bits to the right of the binary point have weights that are negative powers of two, or fractional weights. The weight structure of a binary can be summarized as:

$$2^0 = 1$$
$$2^1 = 2$$
$$2^2 = 4$$
$$2^3 = 8$$
$$2^4 = 16$$
$$2^5 = 32$$
$$2^6 = 64$$
$$2^7 = 128$$
$$2^8 = 256$$
$$2^9 = 512$$
$$2^{10} = 1024$$

2-4: 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. The following example will illustrate this.

EXAMPLE 2- 1 :

Convert the binary number 1101101 to decimal.

SOLUTION: Determine the weight of each bit that is a I, and then find the sum of the weights to get the decimal number.

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

Or you can do it as:

64	32		8	4		1
1	1	0	1	1	0	1

So, the decimal number is:

$$64 + 32 + 8 + 4 + 1 = 109$$

This operation can be written as:

$$1101101_{(2)} = 109_{(10)}$$

RELATED EXERCISE:

Convert the binary number 1000101101 to decimal:

……

……

2-5: DECIMAL-TO-BINARY CONVERSION

A systematic method of converting whole numbers from decimal to binary is the repeated division-by-Z process. For example, to convert the decimal number 12 to binary, begin by dividing 12 by 2. Then, divide each resulting quotient by 2 until there is a 0 whole-number quotient. The remainders generated by each division form the binary number. The first remainder to be produced is the LSB (least significant bit) in the binary number, and the last remainder to be produced is the MSB (most significant bit). This procedure is shown in the following steps for converting the decimal number 12 to binary.

- First: Divide the decimal number by 2 and note the remainder, the remainder always is 0 or 1.
- Second: l or 0 are called remainder while 6 is called quotient .Divide the quotient 6 by 2 again and note the remainder, continue dividing until the quotient equal O.
- Third: When the quotient = 0 then note the remainder and write down the result at the remainder from the down to up.

	12	Remainder		
2	6	0	12 / 2 = 6	Remainder 0
2	3	0	6 / 2 = 3	Remainder 0
2	1	1	3 / 2 = 1	Remainder 1
2	0	1	1 / 2 = 0	Remainder 1

So, the result of the operation is written as:

$12_{(10)} = 1100_{(2)}$

EXAMPLE 2-2:

Convert the following decimal numbers to binary:

(a) 175 (b) 60

SOLUTION:

(a)

	175	Remainder
2	87	1 (LSB)
2	43	1
2	21	1
2	10	1
2	5	0
2	2	1
2	1	0
2	0	1 (MSB)

$175_{(10)}=10101111_{(2)}$

(b)

	60	Remainder
2	30	0 (LSB)
2	15	0
2	7	1
2	3	1
2	1	1
2	0	1 (MSB)

$60_{(10)}=111100_{(2)}$

RELATED EXERCISE:

Convert the decimal **100**(10) and **256**(10) into binary numbers.

……

……

……

……

2-6: BINARY ARITHEMATIC

Binary arithmetic is essential in all digital computers and in many other types of digital systems. To understand digital systems, you must know the basics of binary addition, ~ subtraction.

2-6-1: BINARY ADDITION

The four basic rules for adding binary digits (bits) are as follows:

0+0 = 0
0+1 = 1
1+0 = 1
1+1 = 0

PLUS A CARRY OF 1

Notice that the first three rules result in a single bit and in the fourth rule the addition of two Is yields a binary two (10). When binary numbers are added, the last condition creates a sum of 0 in a given column and a carry of 1 over to the next column to the left, as illustrated in the following addition of 11 + 1:

```
  1  1
  0  1  1
+ 0  0  1
---------
  1  0  0
```

In the right column, 1 + 1 = 0 with a carry of 1 to the next column to the left. In the middle column, 1 + 1 + 0 = 0 with a carry of binary of 1 over to the next column to the left. In the left column, 1 + 0 + 0 = 1.

When there is a carry of 1, you have a situation in which three bits are being added (a bit in each of the two numbers and a carry bit). This situation is illustrated as follows:

Carry bits

1+ 0+0 =01	Sum of 1 with carry of 0
1+1+0 =10	Sum of 0 with carry of 1
1+0+1 = 10	Sum of 0 with carry of 1
1+1+1=11	Sum of 1 with carry of 1

EXAMPLE 2-3:

Add the following binary numbers. Show the equivalent decimal addition for reference:

(a) 11 + 11 (b) 101 + 110 (c) 11100 + 11010

SOLUTION:

(a)

```
  1  1
  0  1  1     3
+ 0  1  1   + 3
  1  1  0     6
```

(b)

```
     1
  0  1  0  1     5
+ 0  1  1  0   + 6
  1  0  1  1    11
```

(c)

```
     1  1
  0  1  1  1  0  0    28
+ 0  1  1  0  1  0  + 26
  1  1  0  1  1  0    54
```

RELATED EXERCISE:

Add the following binary numbers. Show the equivalent decimal addition for reference :

(a)

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

(b)

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

2-6-2: BINARY SUBTRACTION

Subtraction is the inverse operation of addition. To subtract, subtract column by column, the same as you do with decimal numbers.

The four basic rules for subtracting bits as follows:

0 – 0 = 0
1 – 1 = 0
1 – 0 = 1
0 – 1 = 1

With a borrow of 1

When subtracting numbers, you sometimes have to borrow from the next column to the left. A borrow is required in binary only when you to try to subtract a 1 from 0. In this case, when a 1 is borrowed from the next column to the left, a lO is created in the column being subtracted, and the last of four basic rules just listed must be applied.

EXAMPLE 2-4:

Perform the following binary subtractions. Show the equivalent decimal subtractions:

(a) 1101 - 1010 (b) 11100 - 01101

SOLUTION:

(a)

```
      0  10
    1  1  0  1    13
-   1  0  1  0  - 10
    ------------  ----
    0  0  1  1     3
```

(b)

```
        10 10  1
    0   0  0 10 10
   1   1  1  0  0     28
-- 0   1  1  0  1   - 13
   ------------------  ----
   0   1  1  1  1     15
```

RELATED EXERCISE:

Add the following binary numbers. Show the equivalent decimal addition for reference:

(a) 1 0 1 1 0 0
_ 0 1 1 1 1 1

(b) 1 0 0 1 1 1
_ 0 1 1 0 1 0

REVIEW QUESTIONS

Q.1: Convert the following binary numbers to the equivalent decimal numbers:

110101 (2) = ……(10)

101110101(2) = ……(10)

110110101 (2) = …… (10)

111000111(2) = …… (10)

Q.2: Convert the following decimal numbers to the equivalent binary numbers:

73 *(10)* = …… *(2)*

186 *(10)* = …… *(2)*

375 *(10)* = …….. *(2)*

896 *(10)* = …… *(2)*

Q.3: Perform the following additions and then check each operation by converting the binary numbers to decimal:

(a) 1 0 0 1 1 1
+ 1 1 1 0 0 1

(b) 1 1 1 0 0 1 0 1
+ 0 1 1 0 1 1 1 0

(c) 0 0 1 1 1 0
+ 1 0 1 1 1 0

(d) 1 1 0 1 0 1 1 1
+ 0 1 1 0 0 1 1 0

Q.4: Perform the following subtractions and then check each operation by converting the
binary numbers to decimal:

(a) 1 1 0 0 1 1
_ 1 0 0 1 0 1
———————

(b) 1 0 0 1 1 1
_ 0 0 1 1 1 1
———————

(c) 1 1 0 1 1 0 0 1
_ 0 1 1 0 1 1 1 1
———————

(d) 1 0 0 1 0 1 1 0
_ 0 1 1 0 0 1 1 0
———————

Chapter 3
Basic Logic Gates

INTEGRATED CIRCUITS

The basic logic operations that have been discussed and many more which will be discussed in later chapters are available in integrated circuit (IC)form. Digital systems have incorporated ICs for long because of their small size, high reliability, low cost, and low power consumption. It is important to be able to recognize the Ic packages and to know how the pin connections are numbered.

I-Cs can be classified into two types: Digital K's and Linear lCs. les used for processing analog signals are Linear ICs and ICs used for digital signals are Digital ICs. There are different methods of manufacturing ICs. Monolithic technique is the most popular one. A Monolithic integrated circuit (IC) is an electronic circuit that is constructed entirely on a single small chip of silicon. All components that make up the circuit-transistors, diodes, resistors, and capacitors-are an integral part of that single chip. Fixed-function logic and programmable logic are two broad categories of digital ICs. In Fixed - functions are set by the manufacturer and cannot be altered.

Ic PACKAGES

Integrated circuit (IC) packages are classified according to the way they are mounted on printed circuit- (PC) boards as either through-hole mounted or surface mounted. The through-hole type packages have pins (leads) that are inserted through holes in the PC board and can be soldered to conductors on the opposite side. The most common type through-hole packages is the dual in-line (DIP) shown in Figure below

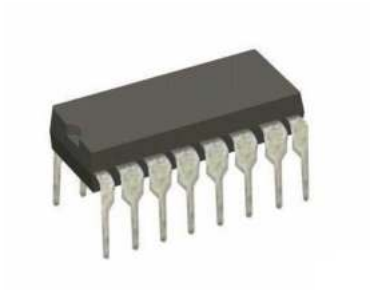
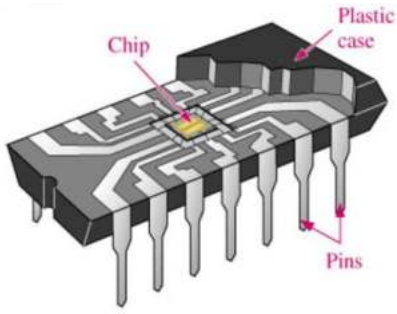
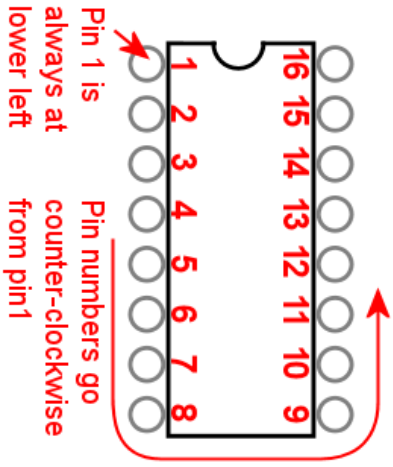


Figure below shows a cutaway view with the circuit chip shown within the package. Points on the chip are connected to the package pins to allow input and output connections to the outside world.



PIN NUMBERING

All IC packages have a standard format for numbering the pins (leads). The dual in-line packages numbering arrangement is illustrated in Figure 3-26 for 16-pin package. Looking at the top of the package, pin 1 is indicated by an identifier that can be either a small dot, a notch, or a beveled edge. The dot is always next to pin 1. Also, with the notch oriented upward, pin 1 is always the top left pin as indicated. Starting with pin 1, the pin numbers increase as you go down, then across and up. The highest pin number is always to the right of the notch or opposite the dot.



MEAN TYPES OF DIGITAL ICs

There are three digital integrated circuit (IC) technologies that are used to implement the basic logic gates. Two of these, CMOS and TIL, are the mostly widely used and the third ECL, is used in more specialized applications. The logic operations of NOT, AND, OR, NAND,NOR, and exclusive-OR are the same regardless of the IC technology used; that is, an AND.
gate has the same logic function whether it is implemented with CMOS, TIL, or ECL.

CMOS stands for Complementary Metal-Oxide Semiconductor and is implemented with a type of field-effect transistor. TTL stands for Transistor-Transistor Logic, is also a bipolar technology.

The *types* of digital ICs could be listed as:

- RTL (Resistor- Transistor Logic)
- DTL (Diode- Transistor Logic)
- TTL (Transistor-Transistor Logic)
- ECL (Emitter-Coupled Logic)
- PMOS or NMOS (P or N channel Metal-Oxide-Semiconductor)
- CMOS (Complementary Metal-Oxide-Semiconductor)

• TTL and CMOS ICs are the two main types of ICs used in the most of digital systems.

3-1 NOT Gate

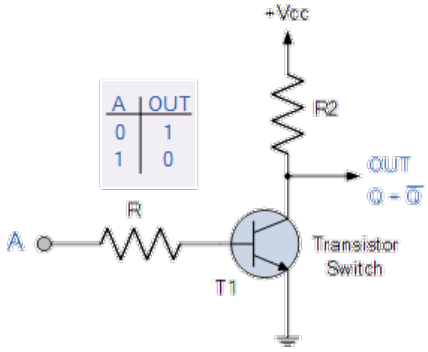
Inverting NOT gates are single input devices which have an output level that is normally at logic level “1” and goes “LOW” to a logic level “0” when its single input is at logic level “1”, in other words it “inverts” (complements) its input signal. The output from a NOT gate only returns “HIGH” again when its input is at logic level “0” giving us the Boolean expression of: A = Q.

Then we can define the operation of a single input digital logic NOT gate as being:

“If A is NOT true, then Q is true”

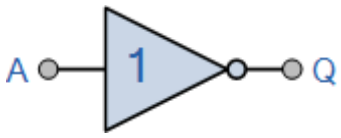
Transistor NOT Gate

A simple 2-input logic NOT gate can be constructed using a RTL Resistor-transistor switches as shown below with the input connected directly to the transistor base. The transistor must be saturated “ON” for an inverted output “OFF” at Q.



Logic NOT Gates are available using digital circuits to produce the desired logical function. The standard NOT gate is given a symbol whose shape is of a triangle pointing to the right with a circle at its end. This circle is known as an “inversion bubble” and is used in NOT, NAND and NOR symbols at their output to represent the logical operation of the NOT function. This bubble denotes a signal inversion (complementation) of the signal and can be present on either or both the output and/or the input terminals.

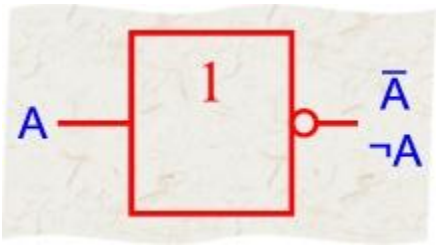
The Logic NOT Gate Truth Table

Symbol	Truth Table	
 Inverter or NOT Gate	**A**	**Q**
	0	**1**
	1	**0**
Boolean Expression **Q = not A or A**	Read as inverse of **A** gives Q	

Logic NOT gates provide the complement of their input signal and are so called because when their input signal is “HIGH” their output state will **NOT** be “HIGH”. Likewise, when their input signal is “LOW” their output state will **NOT** be “LOW”. As they are single input devices, logic NOT gates are not normally classed as “decision” making devices or even as a gate, such as the AND or OR gates which have two or more logic inputs. Commercially available NOT gates IC’s are available in either 4 or 6 individual gates within a single IC package.

The “bubble” (o) present at the end of the NOT gate symbol above denotes a signal inversion (complementation) of the output signal. But this bubble can also be present at the gates input to indicate an *active-LOW* input. This inversion of the input signal is not restricted to the NOT gate only but can be used on any digital circuit or gate as shown with the operation of inversion being the same whether on the input or output terminal. The easiest way is to think of the bubble as simply an inverter.

NOT (Negation) in the lab board



The two symbols shown are used to represent the NOT function.

According to DIN, the second variation Q = ¬A is preferred.

The first variation Q = A is also permissible. It continues to be used for reasons of clarity.

Exercise 1:Experiment set-up

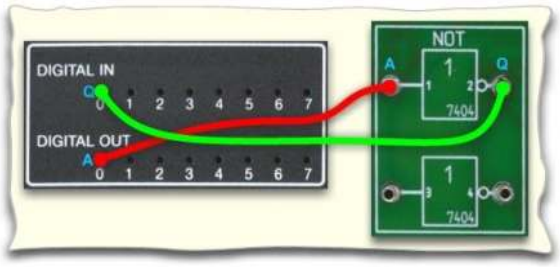


Fig.3-.1.1 : Experiment set-up - Negation

Procedure:

1.1 Construct the circuit as shown in *Fig.3.1.1*.

1.2 Enter the reaction of the outputs to triggering by various logical levels into the table. Use the "Inputs/outputs" VI.

Notes on the VI:
The UniTr@in's outputs are designated $Q_{0..15}$ and its inputs $I_{0..15}$.
UniTr@in's outputs go to the inputs of the experiment card and the outputs of the experiment card go to UniTr@in's inputs. Change the value of Q_0 as suggested and note the response at the output I_0 of the circuit.
[Help available under the menu option: Help → Help topics → Virtual instruments → Standard → Digital → Inputs/outputs]

Q_0	I_0
A	Q = A
0	☐
1	☐

Question:
What is the function of a NOT gate?

……
……
……
……

Exercise 2:Experiment set-up

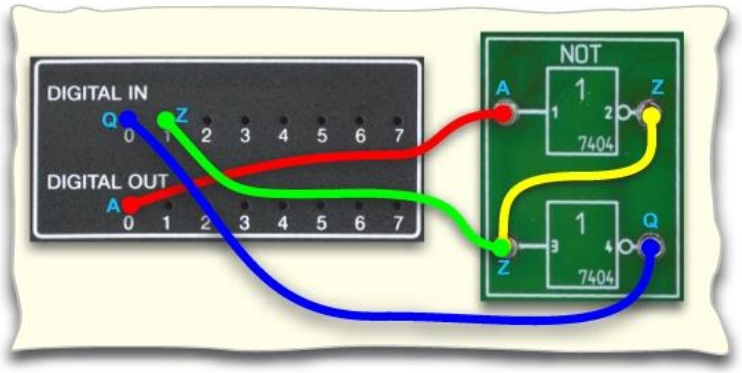


Fig3.1.2 : Experiment set-up - Double negation

Procedure:

2.1 Construct the circuit as shown in *Fig.2*.

2.2 Enter the reaction of the outputs to triggering by various logical levels into the table. Use the "Inputs/outputs" VI.

Q_0	I_1	I_0
A	Z = A	$Q = \overline{\overline{A}}$
0	☐	☐
1	☐	☐

Question:
What happens when two negations are performed:

……
……
……

Complete the equations:

$Z = \overline{\square}$; $Q = \overline{\square} = \overline{\overline{\square}} = \square$

3-2 AND Gates

The output state of a digital logic AND gate only returns “LOW” again when **ANY** of its inputs are at a logic level “0”. In other words for a logic AND gate, any LOW input will give a LOW output.

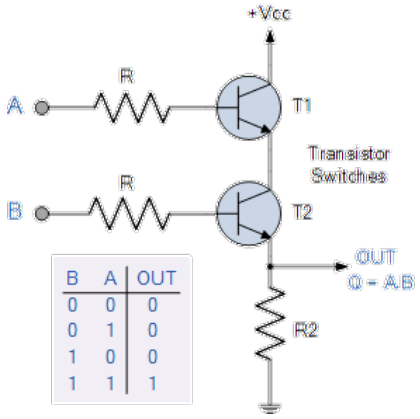
The logic or Boolean expression given for a digital logic AND gate is that for *Logical Multiplication* which is denoted by a single dot or full stop symbol, (.) giving us the Boolean expression of: A.B = Q.

Then we can define the operation of a digital 2-input logic AND gate as being:

“If both A and B are true, then Q is true”

2-input Transistor AND Gate

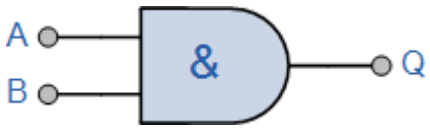
A simple 2-input logic AND gate can be constructed using RTL Resistor-transistor switches connected together as shown below with the inputs connected directly to the transistor bases. Both transistors must be saturated “ON” for an output at Q.



Logic AND Gates are available using digital circuits to produce the desired logical function and is given a symbol whose shape represents the logical operation of the AND gate.

Digital Logic “AND” Gate Types

The 2-input Logic AND Gate

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

Because the Boolean expression for the logic AND function is defined as (.), which is a binary operation, AND gates can be cascaded together to form any number of individual inputs. However, commercial available AND gate IC's are only available in standard 2, 3, or 4-input packages. If additional inputs are required, then standard AND gates will need to be cascaded together to obtain the required input value, for example.

Commonly available digital logic AND gate IC's include:

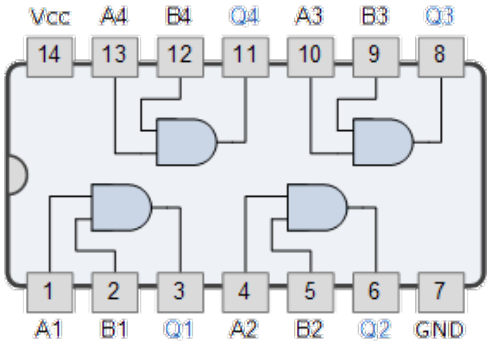
TTL Logic AND Gate

- 74LS08 Quad 2-input
- 74LS11 Triple 3-input
- 74LS21 Dual 4-input

CMOS Logic AND Gate

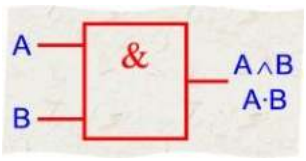
- CD4081 Quad 2-input
- CD4073 Triple 3-input
- CD4082 Dual 4-input

7408 Quad 2-input AND Gate



In the next tutorial about **Digital Logic Gates**, we will look at the digital logic OR Gate function as used in both TTL and CMOS logic circuits as well as its Boolean Algebra definition and truth tables.

AND Gate in the Lab board



The Boolean AND operation can be written in the forms $Q=A \wedge B$ or $Q = A \cdot B$. The latter will be used in this course.

It should be noted that although the Boolean AND function bears certain similarities to algebraic multiplication, there are also distinct differences.

Exercise 1: Experiment set-up

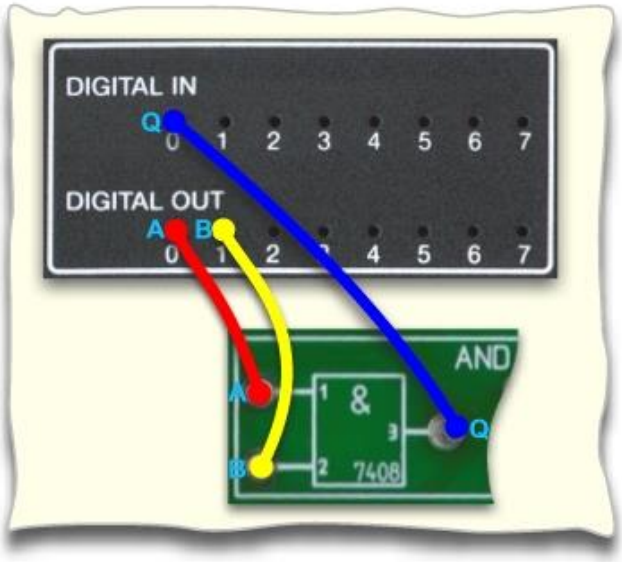


Fig.3.2.1: Experiment set-up - AND

Procedure:

1.1 Construct the circuit as shown in *Fig.3.2.1.*

1.2 Observe the response of the output to various logic levels at the input and enter them into the table. Use the ”Extended inputs/outputs" VI.

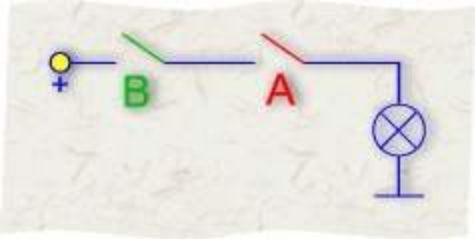
Tip:
You can alter the bit status quickly and easily by using the up and down buttons on the VI.

Q_1 Q_0 I_0

B	A	Q
0	0	
0	1	
1	0	
1	1	

Describe the functionality of the AND gate:

……
……
……



AND operation demonstrated by switches and a lamp

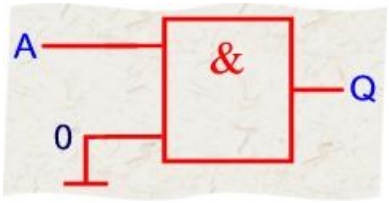
Compare the switch analogy to the actual AND gate:

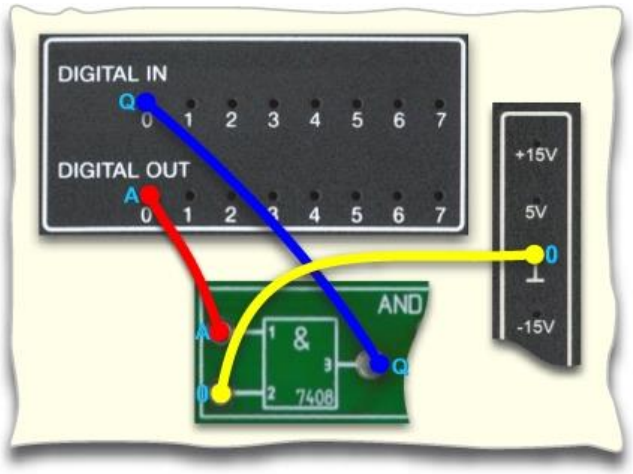
……
……
……

Exercise 2

Construct the following experiments and note down the results. Describe in a few words the rules that you discover.

a) ANDing with "0"



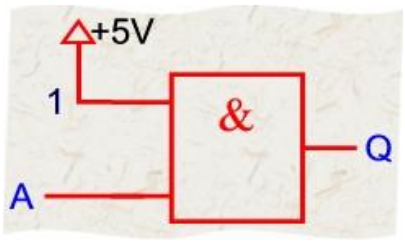


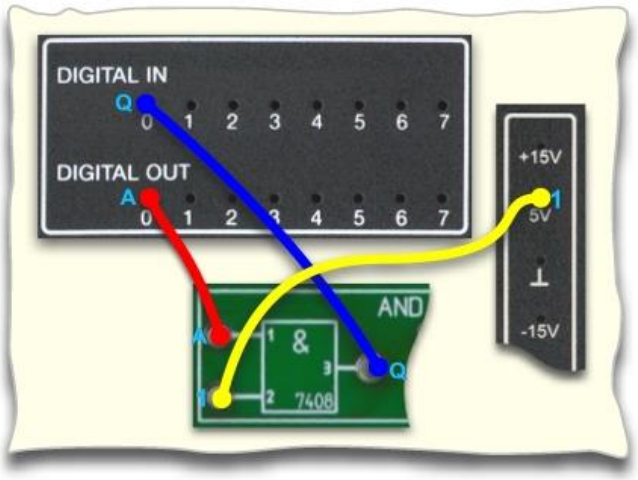
Result: **A · 0 =** ☐

How do you interpret this result?

……
……
……

b) ANDing with "1"



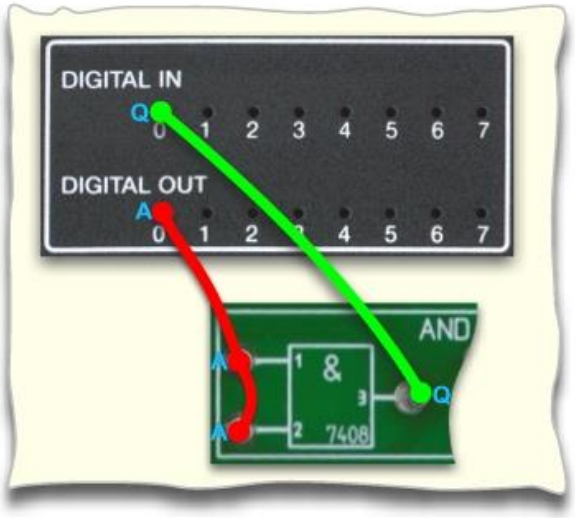


Result: **A · 1 =** ☐

How do you interpret this result?

……
……
……

c) ANDing A·A (Tautology)

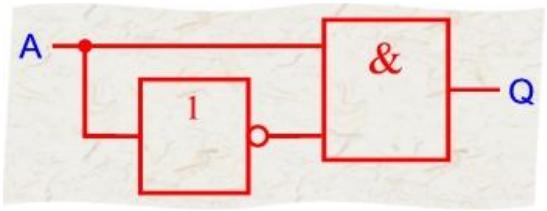


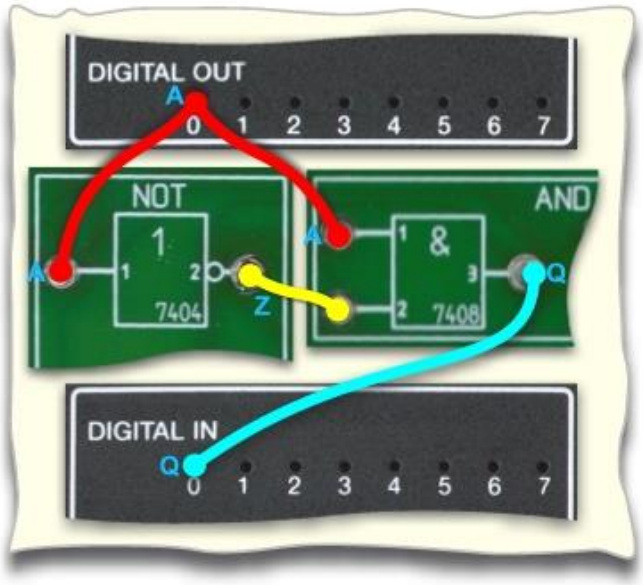
Result: $\mathbf{A \cdot A =}$ ☐

How do you interpret the result?

……
……
……

d) ANDing A·A (Negation law)





Result: $\mathbf{A \cdot A =}$ ☐

How do you interpret the result?

……
……
……

3.3 OR Gates

The output, Q of a “Logic OR Gate” only returns “LOW” again when **ALL** of its inputs are at a logic level “0”. In other words for a logic OR gate, any “HIGH” input will give a “HIGH”, logic level “1” output.

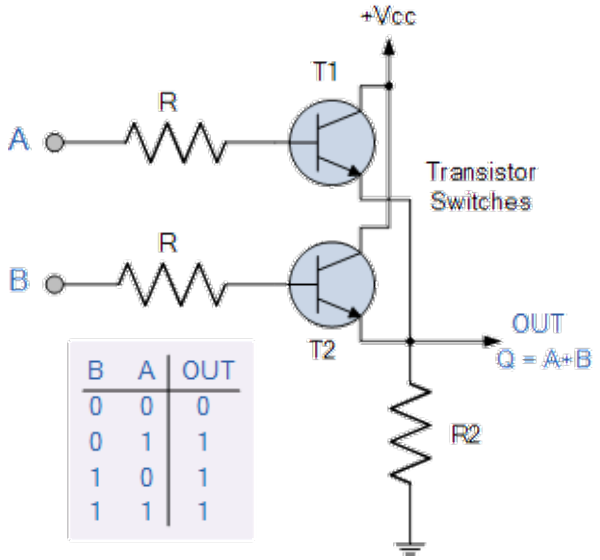
The logic or Boolean expression given for a digital logic OR gate is that for *Logical Addition* which is denoted by a plus sign, (+) giving us the Boolean expression of: A+B = Q.

Thus a logic OR gate can be correctly described as an “Inclusive OR gate” because the output is true when both of its inputs are true (HIGH). Then we can define the operation of a 2-input logic OR gate as being:

“If either A or B is true, then Q is true”

2-input Transistor OR Gate

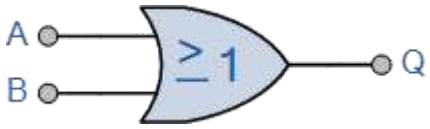
A simple 2-input inclusive OR gate can be constructed using RTL Resistor-transistor switches connected together as shown below with the inputs connected directly to the transistor bases. Either transistor must be saturated “ON” for an output at Q.



B	A	OUT
0	0	0
0	1	1
1	0	1
1	1	1

Logic OR Gates are available using digital circuits to produce the desired logical function and is given a symbol whose shape represents the logical operation of the OR gate.

The 2-input Logic OR Gate

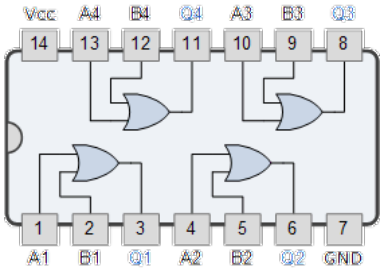
Symbol	Truth Table		
 2-input OR Gate	B	A	Q
	0	0	0
	0	1	1
	1	0	1
	1	1	1
Boolean Expression **Q = A+B**	Read as A **OR** B gives Q		

TTL Logic OR Gates

- 74LS32 Quad 2-input

CMOS Logic OR Gates

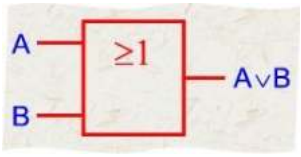
- CD4071 Quad 2-input
- CD4075 Triple 3-input
- CD4072 Dual 4-input



7432 Quad 2-input Logic OR Gate

In the next tutorial about **Digital Logic Gates**, we will look at the digital logic NOT Gate function as used in both TTL and CMOS logic circuits as well as its Boolean Algebra definition and truth table.

OR Gate in the Lab board



The Boolean OR operation is usually written in the form $Q=A \vee B$. Occasionally the following format is seen $Q = A + B$. This form will not be used in this course.

It should be noted that although the Boolean OR function bears certain similarities to algebraic addition, there are also distinct differences.

Exercise 1:Experiment set-up

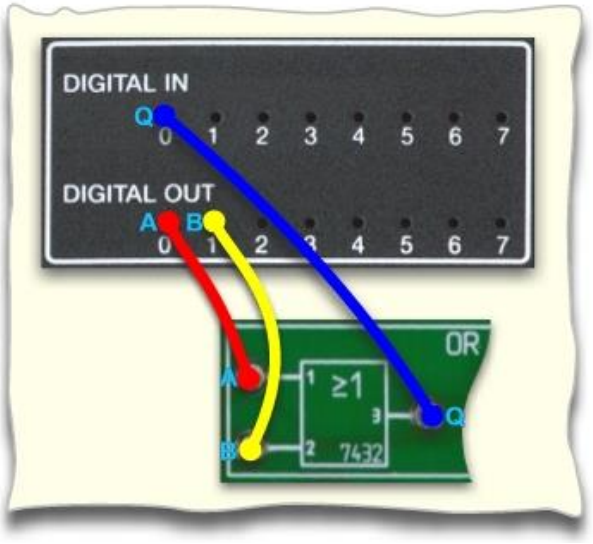


Fig.3.3.1 : Experiment set-up - OR

Procedure:

1.1 Construct the circuit as shown in *Fig.3.3.1*.

1.2 Observe the response of the output to various logic levels at the input and enter them into the table. Use the "Extended inputs/outputs" VI.

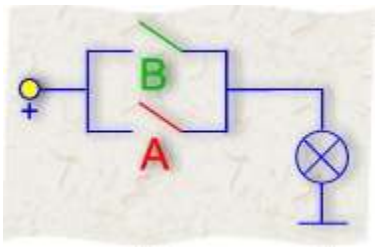
Tip:
You can alter the bit status quickly and easily by using the up and down buttons on the VI.

Q_1 Q_0 I_0

B	A	Q
0	0	
0	1	
1	0	
1	1	

Describe the functionality of the OR gate:

……
……
……
……



OR operation demonstrated

by switches and a lamp

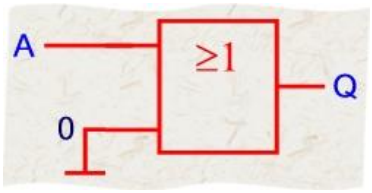
Compare the switch analogy to the actual OR gate:

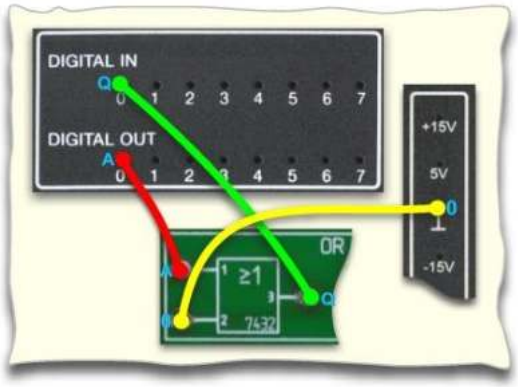
……..

Exercise 2

Construct the following experiments and note down the results. Describe in a few words the rules that you discover.

a) ORing with "0"



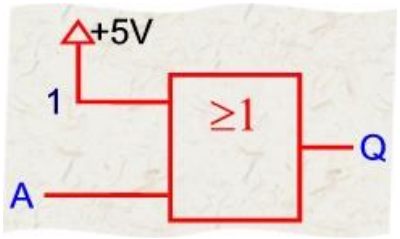


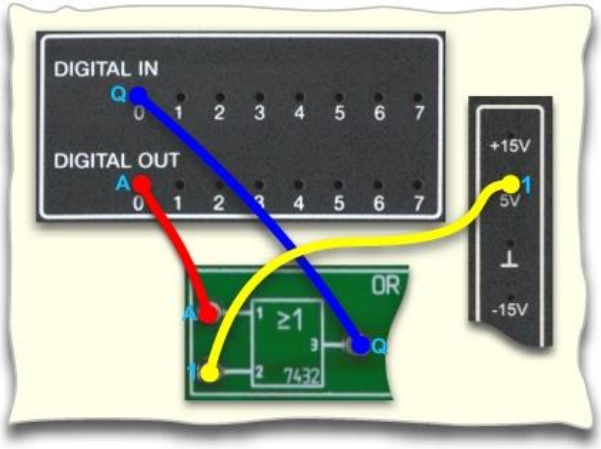
Result: **A ∨ 0 =** ☐

How do you interpret the results?

……
……
……

b) ORing with "1"



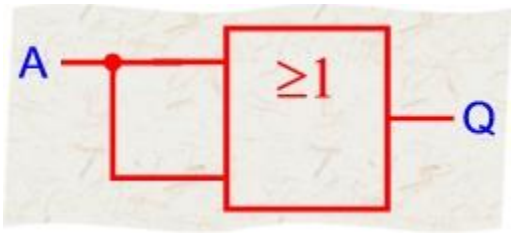


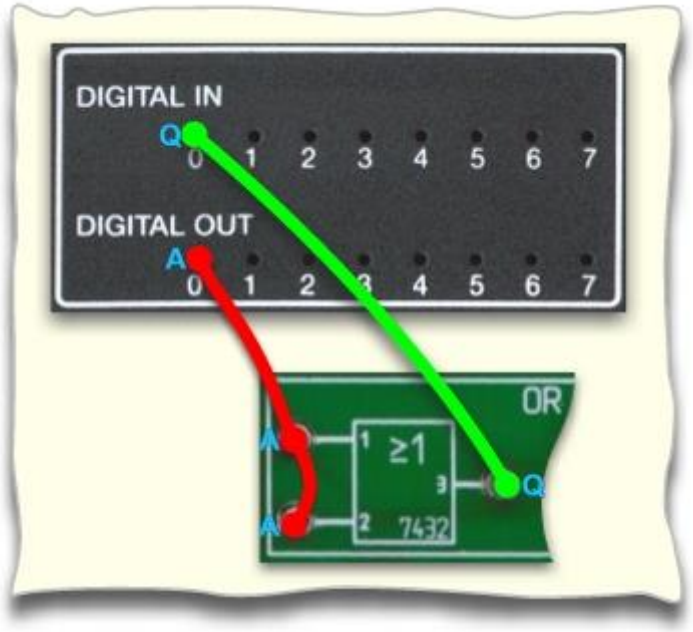
Result: **1 ∨ A =** ☐

How do you interpret the result?

……
……

c) ORing A ∨ A (Tautology)



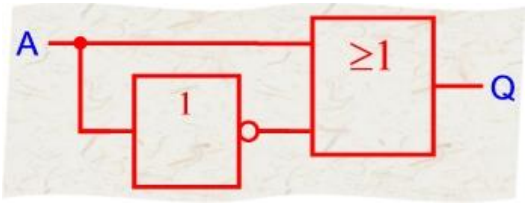


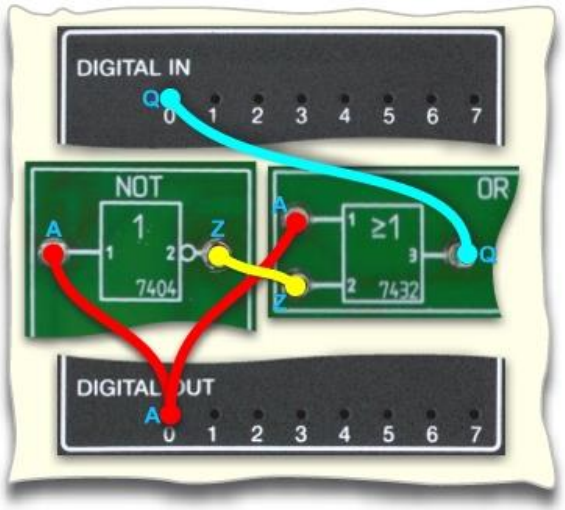
Result: **A ∨ A =** ☐

How do you interpret the result?

……
……
……

d) ORing A ∨ A (Negation law)





Result: **A ∨ A =** ☐

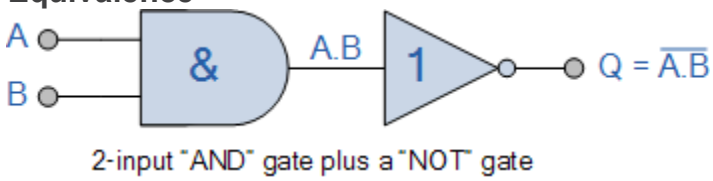
How do you interpret the result?

……
……
……

3.4 NAND Gates

The NAND (Not – AND) gate has an output that is normally at logic level “1” and only goes “LOW” to logic level “0” when **ALL** of its inputs are at logic level “1”. The **Logic NAND Gate** is the reverse or “*Complementary*” form of the AND gate we have seen previously.

Logic NAND Gate Equivalence



2-input “AND” gate plus a “NOT” gate

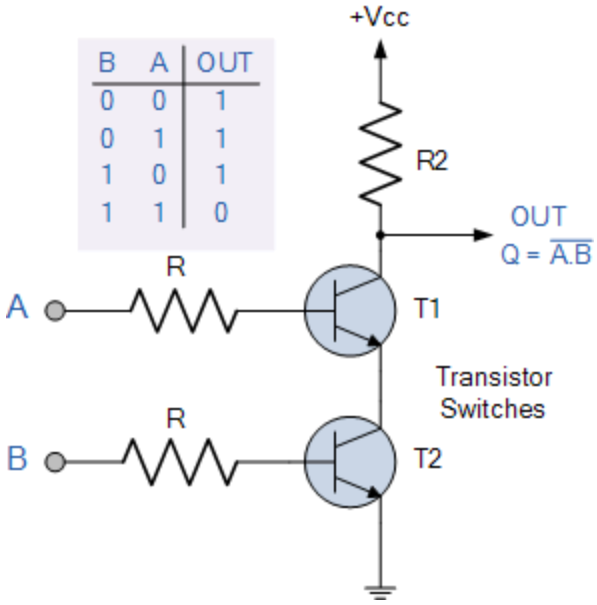
The logic or Boolean expression given for a logic NAND gate is that for *Logical Addition*, which is the opposite to the AND gate, and which it performs on the *complements* of the inputs. The Boolean expression for a logic NAND gate is denoted by a single dot or full stop symbol, (.) with a line or *Overline*, (¯) over the expression to signify the NOT or logical negation of the NAND gate giving us the Boolean expression of: A.B = Q.

Then we can define the operation of a 2-input digital logic NAND gate as being:

“If either A or B are NOT true, then Q is true”

Transistor NAND Gate

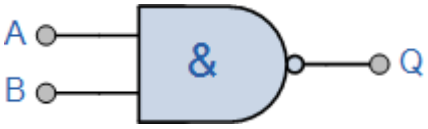
A simple 2-input logic NAND gate can be constructed using RTL Resistor-transistor switches connected together as shown below with the inputs connected directly to the transistor bases. Either transistor must be cut-off “OFF” for an output at Q.



Logic NAND Gates are available using digital circuits to produce the desired logical function and is given a symbol whose shape is that of a standard AND gate with a circle, sometimes called an “inversion bubble” at its output to represent the NOT gate symbol with the logical operation of the NAND gate given as.

The Digital Logic “NAND” Gate

2-input Logic NAND Gate

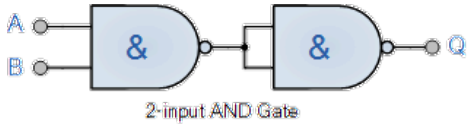
Symbol	Truth Table		
 2-input NAND Gate	B	A	Q
	0	0	1
	0	1	1
	1	0	1
	1	1	0
Boolean Expression **Q = A.B**	Read as A **AND** B gives **NOT** Q		

The “Universal” NAND Gate

The **Logic NAND Gate** is generally classed as a “Universal” gate because it is one of the most commonly used logic gate types. NAND gates can also be used to produce any other type of logic gate function, and in practice the NAND gate forms the basis of most practical logic circuits.

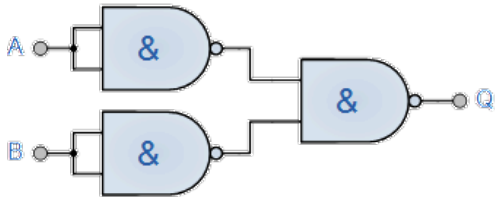
By connecting them together in various combinations the three basic gate types of AND, OR and NOT function can be formed using only NAND gates, for example.

Various Logic Gates using only NAND Gates



2-input AND Gate

NOT Gate



2-input OR Gate

As well as the three common types above, Exclusive-OR, Exclusive-NOR and standard NOR gates can be formed using just individual NAND gates.

Commonly available digital logic NAND gate IC's include:

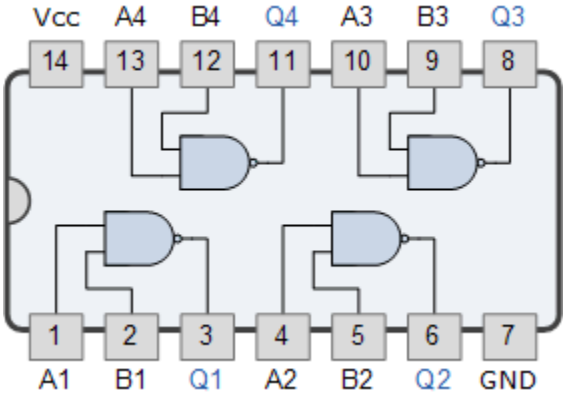
TTL Logic NAND Gates

- 74LS00 Quad 2-input
- 74LS10 Triple 3-input
- 74LS20 Dual 4-input
- 74LS30 Single 8-input

CMOS Logic NAND Gates

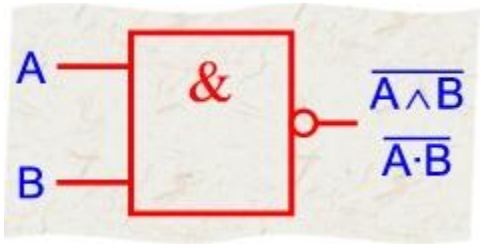
- CD4011 Quad 2-input
- CD4023 Triple 3-input
- CD4012 Dual 4-input

7400 Quad 2-input Logic NAND Gate



In the next tutorial about **Digital Logic Gates**, we will look at the digital logic NOR Gate function as used in both TTL and CMOS logic circuits as well as its Boolean Algebra definition and truth tables.

NAND (Not AND) in the Lab Board



The Boolean AND operation can be written in the forms $Q=A \wedge B$ or $Q = A \cdot B$ The latter will be used on this course.

Exercise 1: Experiment set-up

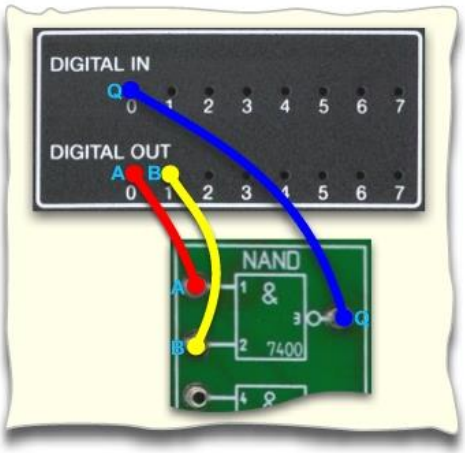


Fig3.4.1 : Experiment set-up - NAND

Procedure:

1.1 Construct the circuit as shown in *Fig.3.4.1*.

1.2 Observe the response of the output to various logic levels at the input and enter them into *Table 1*. Use the "Extended inputs/outputs" VI.

Tip:
You can alter the bit status quickly and easily by using the up and down buttons on the VI.

Table 1

Q_1 Q_0 I_0

B	A	Q
0	0	
0	1	
1	0	
1	1	

NAND

Describe the functionality of the NAND gate:

……
……
……

Exercise 2

2.1 Summaries the functionality of the AND and NAND gates in *Table 2*.

Table 2

B	A	A·B	A·B
0	0		
0	1		
1	0		
1	1		
		↑ AND	↑ NAND

What two gates can be combined to create a NAND gate?

……
……
……

Is the converse also true, i.e. can an AND gate be made by combining a NAND gate with one other gate?

……
……

3.5 Laws of Boolean Algebra

As well as the logic symbols “0” and “1” being used to represent a digital input or output, we can also use them as constants for a permanently “Open” or “Closed” circuit or contact respectively.

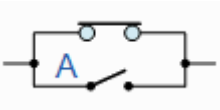
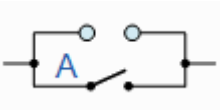
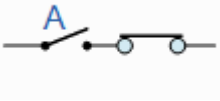
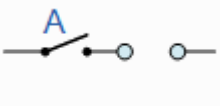
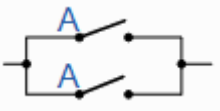
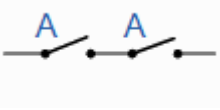
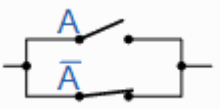
A set of rules or Laws of Boolean Algebra expressions have been invented to help reduce the number of logic gates needed to perform a particular logic operation resulting in a list of functions or theorems known commonly as the **Laws of Boolean Algebra**.

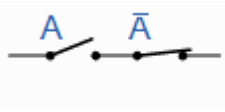
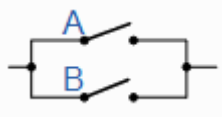
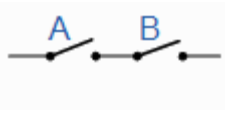
Boolean Algebra is the mathematics we use to analyse digital gates and circuits. We can use these “Laws of Boolean” to both reduce and simplify a complex Boolean expression in an attempt to reduce the number of logic gates required. *Boolean Algebra* is therefore a system of mathematics based on logic that has its own set of rules or laws which are used to define and reduce Boolean expressions.

The variables used in **Boolean Algebra** only have one of two possible values, a logic “0” and a logic “1” but an expression can have an infinite number of variables all labelled individually to represent inputs to the expression, For example, variables A, B, C etc, giving us a logical expression of A + B = C, but each variable can ONLY be a 0 or a 1.

Examples of these individual laws of Boolean, rules and theorems for Boolean Algebra are given in the following table.

Truth Tables for the Laws of Boolean

Boolean Expression	Description	Equivalent Switching Circuit	Boolean Algebra Law or Rule
A + 1 = 1	A in parallel with closed = "CLOSED"		Annulment
A + 0 = A	A in parallel with open = "A"		Identity
A . 1 = A	A in series with closed = "A"		Identity
A . 0 = 0	A in series with open = "OPEN"		Annulment
A + A = A	A in parallel with A = "A"		Idempotent
A . A = A	A in series with A = "A"		Idempotent
NOT A = A	NOT NOT A (double negative) = "A"		Double Negation
A + A = 1	A in parallel with NOT A = "CLOSED"		Complement

A . A = 0	A in series with NOT A = "OPEN"		Complement
A+B = B+A	A in parallel with B = B in parallel with A		Commutative
A.B = B.A	A in series with B = B in series with A		Commutative
A+B = A.B	invert and replace OR with AND		de Morgan's Theorem
A.B = A+B	invert and replace AND with OR		de Morgan's Theorem

The basic **Laws of Boolean Algebra** that relate to the *Commutative Law* allowing a change in position for addition and multiplication, the *Associative Law* allowing the removal of brackets for addition and multiplication, as well as the *Distributive Law* allowing the factoring of an expression, are the same as in ordinary algebra.

Each of the *Boolean Laws* above are given with just a single or two variables, but the number of variables defined by a single law is not limited to this as there can be an infinite number of variables as inputs too the expression. These Boolean laws detailed above can be used to prove any given Boolean expression as well as for simplifying complicated digital circuits.

A brief description of the various **Laws of Boolean** are given below with A representing a variable input.

Description of the Laws of Boolean Algebra

- Annulment Law – A term AND´ed with a “0” equals 0 or OR´ed with a “1” will equal 1
  - A . 0 = 0 A variable AND’ed with 0 is always equal to 0
  - A + 1 = 1 A variable OR’ed with 1 is always equal to 1

- Identity Law – A term OR´ed with a “0” or AND´ed with a “1” will always equal that term
  - A + 0 = A A variable OR’ed with 0 is always equal to the variable
  - A . 1 = A A variable AND’ed with 1 is always equal to the variable

- Idempotent Law – An input that is AND´ed or OR´ed with itself is equal to that input
  - A + A = A A variable OR’ed with itself is always equal to the variable
  - A . A = A A variable AND’ed with itself is always equal to the variable

- Complement Law – A term AND´ed with its complement equals “0” and a term OR´ed with its complement equals “1”
  - A . A = 0 A variable AND’ed with its complement is always equal to 0
  - A + A = 1 A variable OR’ed with its complement is always equal to 1

- Commutative Law – The order of application of two separate terms is not important
  - A . B = B . A The order in which two variables are AND’ed makes no difference
  - A + B = B + A The order in which two variables are OR’ed makes no difference

- Double Negation Law – A term that is inverted twice is equal to the original term
  - $\overline{\overline{A}} = A$ A double complement of a variable is always equal to the variable

- de Morgan´s Theorem – There are two "de Morgan´s" rules or theorems, (1) Two separate terms NOR´ed together is the same as the two terms inverted (Complement) and AND´ed for example: $A+B = \overline{A} \cdot \overline{B}$
- (2) Two separate terms NAND´ed together is the same as the two terms inverted (Complement) and OR´ed for example: $A.B = \overline{A} + \overline{B}$

Other algebraic Laws of Boolean not detailed above include:

- Distributive Law – This law permits the multiplying or factoring out of an expression.
  - $A(B + C) = A.B + A.C$ (OR Distributive Law)
  - $A + (B.C) = (A + B).(A + C)$ (AND Distributive Law)
- Absorptive Law – This law enables a reduction in a complicated expression to a simpler one by absorbing like terms.
  - $A + (A.B) = A$ (OR Absorption Law)
  - $A(A + B) = A$ (AND Absorption Law)
- Associative Law – This law allows the removal of brackets from an expression and regrouping of the variables.
  - $A + (B + C) = (A + B) + C = A + B + C$ (OR Associate Law)
  - $A(B.C) = (A.B)C = A \cdot B \cdot C$ (AND Associate Law)

Boolean Algebra Functions
Using the information above, simple 2-input AND, OR and NOT Gates can be represented by 16 possible functions as shown in the following table.

Function	Description	Expression
1.	NULL	0
2.	IDENTITY	1
3.	Input A	A
4.	Input B	B
5.	NOT A	A
6.	NOT B	B
7.	A AND B (AND)	A . B
8.	A AND NOT B	A . B
9.	NOT A AND B	A . B
10.	NOT AND (NAND)	A . B
11.	A OR B (OR)	A + B
12.	A OR NOT B	A + B
13.	NOT A OR B	A + B
15.	Exclusive-OR	A . B + A . B

16.	Exclusive-NOR	A . B + A . B
14.	NOT OR (NOR)	A + B

Laws of Boolean Algebra Example No1
Using the above laws, simplify the following expression: (A + B)(A + C)

Q = (A + B).(A + C)

A.A + A.C + A.B + B.C – Distributive law

A + A.C + A.B + B.C – Idempotent AND law (A.A = A)

A(1 + C) + A.B + B.C – Distributive law

A.1 + A.B + B.C – Identity OR law (1 + C = 1)

A(1 + B) + B.C – Distributive law

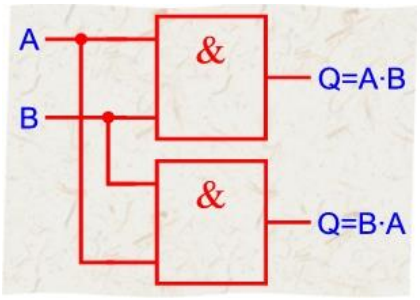
A.1 + B.C – Identity OR law (1 + B = 1)

Q = A + (B.C) – Identity AND law (A.1 = A)

Then the expression: (A + B)(A + C) can be simplified to A + (B.C) as in the Distributive law

Commutative laws experiment

Commutative law for the AND operation



The commutative law for AND operations:

$$A \cdot B = B \cdot A$$

Write out the law in your own words.
Is this law the same as the commutative law for normal multiplication?

……
……

Exercise 1:

Experiment set-up

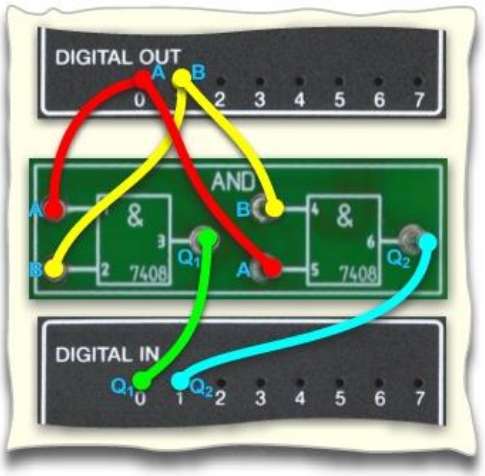


Fig.3.5.1 : Experiment set-up - Commutative law, AND

Procedure:

1.1. Construct the circuit as shown in *Fig.3.5.1*.

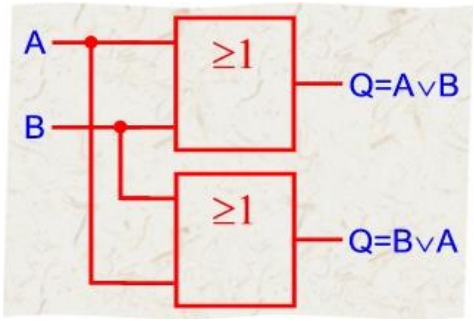
1.2. Observe the response of the output to various logic levels at the input and enter them into *Table 1*. Use the "Extended inputs/outputs" VI.

Table 1

Q_1	Q_0	I_1	I_0
B	**A**	**Q_2**	**Q_1**
0	0		
0	1		
1	0		
1	1		
		↑ B·A	↑ A·B

1.3. Confirm the validity of the commutative law by means of the values in the table.

Commutative law for the OR operation



The commutative law for OR operations is:

$$A \vee B = B \vee A$$

Exercise 2:

Experiment set-up

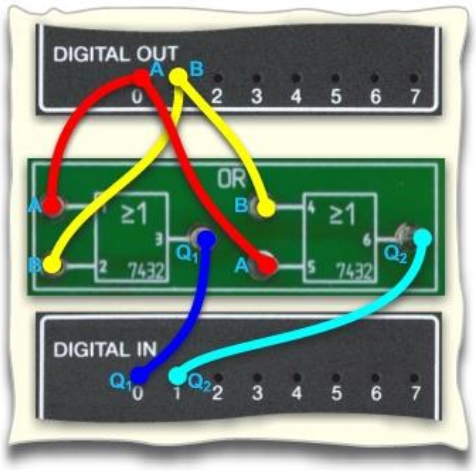


Fig.3.5.2 : Experiment set-up - Commutative law, OR

Procedure:

2.1 Construct the circuit as shown in *Fig3.5.2*.

2.2 Observe the response of the output to various logic levels at the input and enter them into *Table 2*. Use the "Extended inputs/outputs" VI.

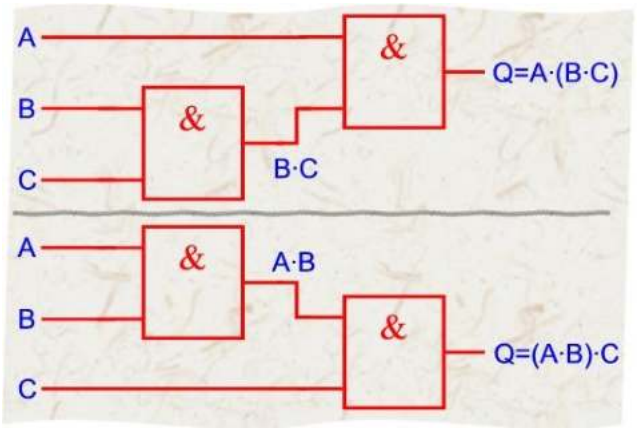
Table 2

Q_1	Q_0	I_1	I_0
B	**A**	**Q_2**	**Q_1**
0	0		
0	1		
1	0		
1	1		
		↑ B ∨ A	↑ A ∨ B

2.3 Confirm the validity of the commutative law by means of the values in the table.

Associative laws Experiment

Associative law for the AND operation



The associative law for AND operations is:

$$A \cdot B \cdot C = A \cdot (B \cdot C) = (A \cdot B) \cdot C$$

Write out the law in your own words.
Is this law the same as the associative law for normal multiplication?

……
……

Exercise 1:

Experiment set-up

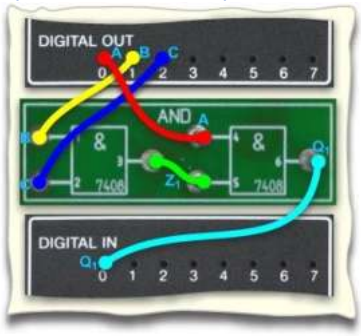


Fig.3.5.3 : Experiment set-up - Associative law A·(B·C)

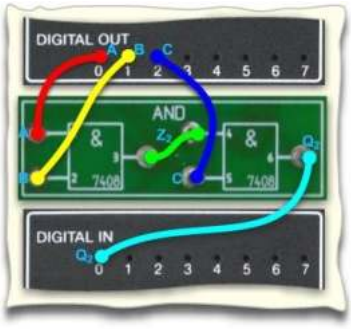


Fig.3.5.4 : Experiment set-up - Associative law (A·B)·C

Procedure:

1.1 Construct the first circuit as shown in *Fig.3.5.3*

1.2 Observe the response of the output to various logic levels at the input and enter them into *Table 1*. Use the "Extended inputs/outputs" VI.

1.3 Construct the second circuit as shown in *Fig.3.5.4*.

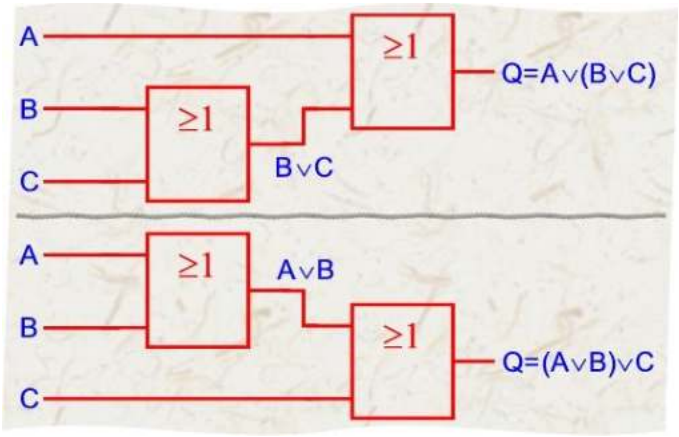
1.4 Repeat step 1.2 for this circuit.

Table 1

			re	re
			Fig.3.5.3	**Fig.3.5.4**
Q_2	Q_1	Q_0	I_0	I_0
C	**B**	**A**	Q_1	Q_2
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		
			↑	↑
			A·(B·C)	(A·B)·C

1.5 Confirm the validity of the associative law by means of the values in the table.

Associative law for the OR operation



The associative law for OR operations is:

$$A \vee B \vee C = A \vee (B \vee C) = (A \vee B) \vee C$$

Write out the law in your own words.
Is this law the same as the associative law for normal addition?

……
……

Exercise 2:

Experiment set-up

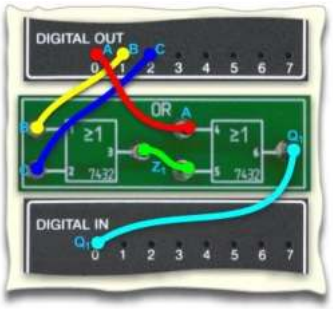


Fig.3.5.5 : Experiment set-up - Associative law A∨(B∨C)

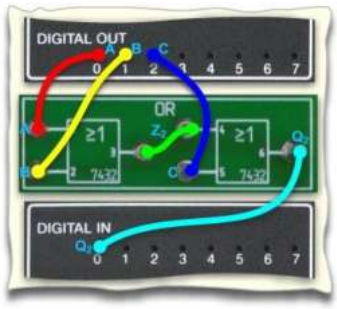


Fig.3.5.6 : Experiment set-up - Associative law (A∨B)∨C

Procedure:

2.1 Construct the first circuit as shown in *Fig. 3.5.5*.
2.2 Observe the response of the output to various logic levels at the input and enter them into *Table 2*. Use the "Extended inputs/outputs" VI.
2.3 Construct the second circuit as shown in *Fig.3.5.6*.
2.4 Repeat step 2.2 for this circuit.

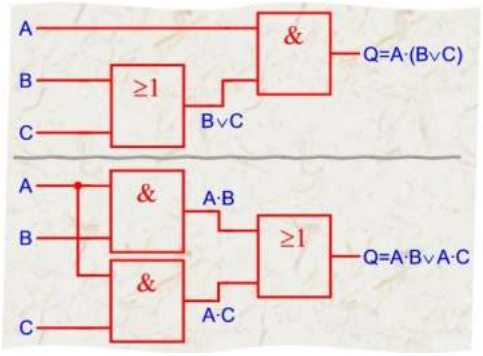
Table 2

			Fig.3.5.5	**Fig.3.5.6**
Q_2	Q_1	Q_0	I_0	I_0
C	**B**	**A**	**Q_1**	**Q_2**
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		
			↑ A∨(B∨C)	↑ (A∨B)∨C

2.5 Confirm the validity of the associative law according to the values in the table.

Distributive laws Experiment

First distributive law



The first distributive law is:

$$A \cdot (B \vee C) = (A \cdot B) \vee (A \cdot C)$$

Note on the sequence of operations:
Similar to ordinary algebra where multiplication and division take precedence over addition and subtraction, here AND operations have priority over OR operations.

Write out the law in your own words. Is this law the same as the distributive law in normal algebra?

……
……

Exercise 1:Experiment set-up

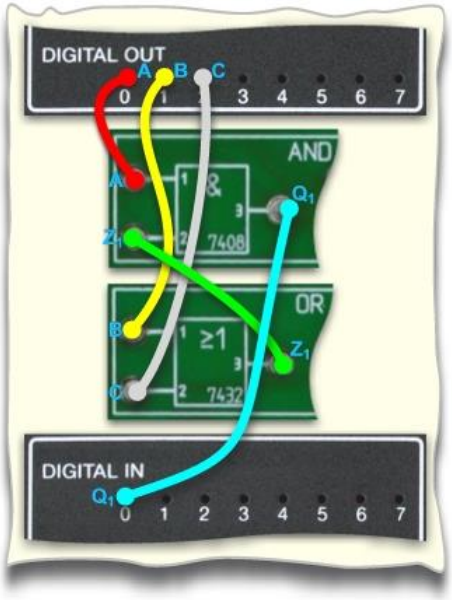


Fig.3.5.7 : Experiment set-up - 1st distributive law
$Q_1 = A \cdot (B \vee C)$

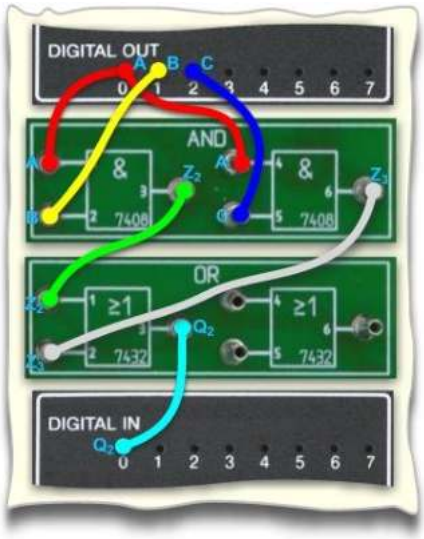


Fig.3.5.8 : Experiment set-up, 1st distributive law
$Q_2 = (A \cdot B) \vee (A \cdot C)$

Procedure:

1.1 Construct the first circuit as shown in Fig.3.5.7.
1.2 Observe the response of the output to various logic levels at the input and enter them into Table 1. Use the "Extended inputs/outputs" VI.
1.3 Construct the second circuit as shown in Fig.3.5.8 .
1.4 Repeat step 1.2 for this circuit.

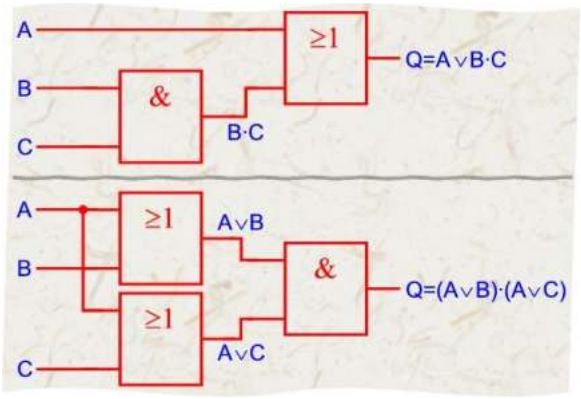
Table 1

			Fig.3.5.7	Fig.3.5.8
Q_2	Q_1	Q_0	I_0	I_0
C	**B**	**A**	**Q_1**	**Q_2**
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

$Q_1 = A \cdot (B \vee C)$
$Q_2 = (A \cdot B) \vee (A \cdot C)$

1.5 Confirm the validity of the distributive law by means of the values in the table.

2nd distributive law



The 2nd distributive law is:

$$A \vee (B \cdot C) = (A \vee B) \cdot (A \vee C)$$

Write out the law in your own words. Is this law the same as the distributive law in normal algebra?

……

……

……

Exercise 2:

Experiment set-up

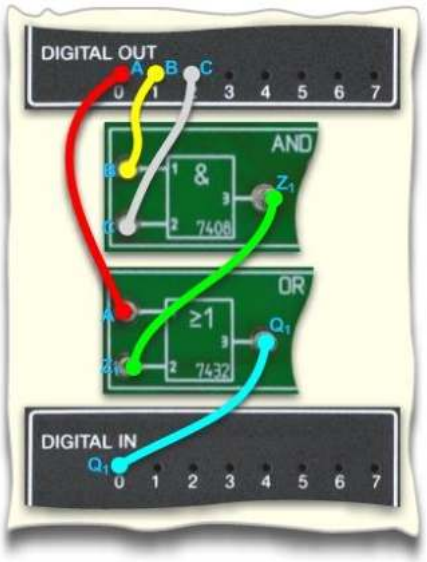


Fig.3.5.9 : Experiment set-up, 2nd distributive law
$Q_1 = A \vee (B \cdot C)$

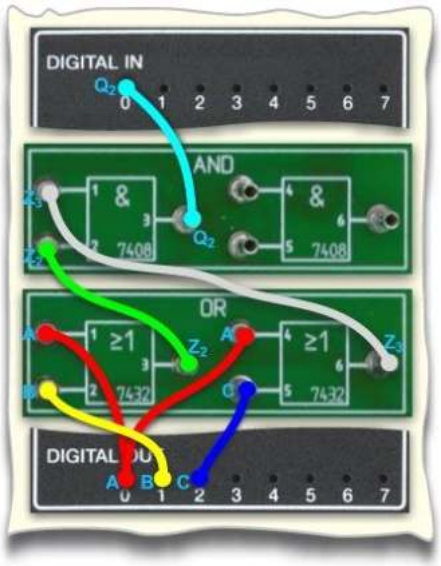


Fig.3.5.10 : Experiment set-up, 2nd distributive law
$Q_2 = (A \vee B) \cdot (A \vee C)$

Procedure

2.1 Construct the circuit as shown in Fig. 2.1.
2.2 Observe the response of the output to various logic levels at the input and enter them into Table 2. Use the "Extended inputs/outputs" VI.
2.3 Construct the 2nd circuit as shown in Fig.2.2 .
2.4 Repeat step 2.2 for this circuit.

Table 2

			Fig.3.5.9	Fig.3.5.10
Q_2	Q_1	Q_0	I_0	I_0
C	**B**	**A**	**Q_1**	**Q_2**
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

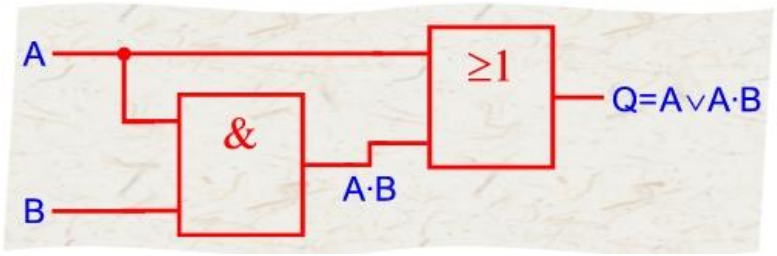
$$Q_1 = A \vee (B \cdot C)$$
$$Q_2 = (A \vee B) \cdot (A \vee C)$$

2.5 Confirm the validity of the 2nd distributive law by means of the values in the table.

Absorption experiment

In the following exercises, we will specifically investigate AND/OR/NOT sequences with two input variables that can be simplified.

Exercise 1 $A \vee (A \cdot B) = ?$



Experiment set-up

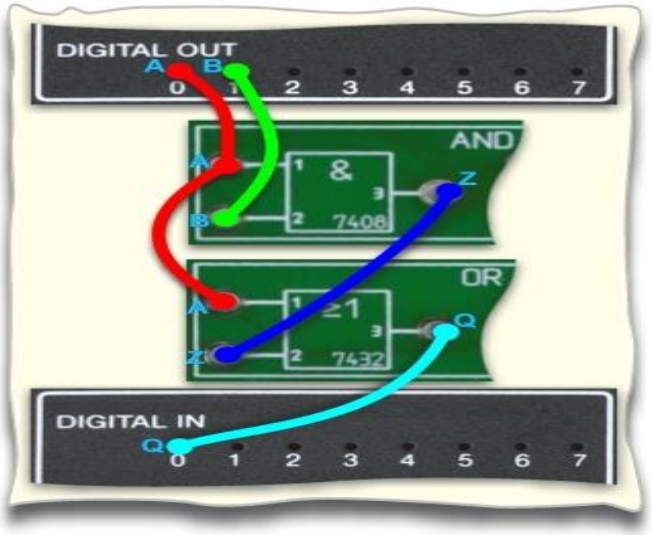


Fig.3.5.11 : Experiment set-up - $Q = A \vee (A \cdot B)$

Procedure:

1.1 Construct the first circuit as shown in *Fig.3.5.11*

1.2 Observe the response of the output to various logic levels at the input and enter them into *Table 1*. Use the "Extended inputs/outputs" VI.

1.3 Copy the values from your table into *Karnaugh map 1*.

1.4 Confirm the result you obtained from the table using the Karnaugh map.

Table 1

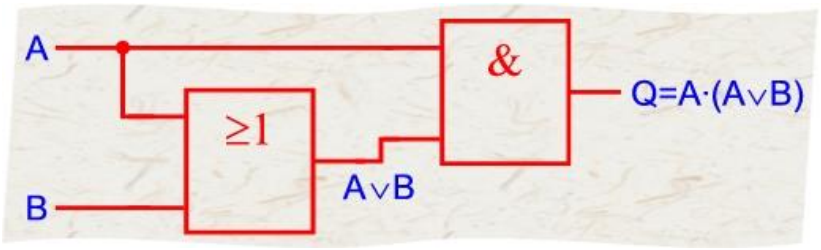
Q_1	Q_0	I_0
B	A	Q
0	0	
0	1	
1	0	
1	1	

Karnaugh map 1

A	A
		B
		B

$Q = A \vee (A \cdot B) =$	

Exercise 2 $A \cdot (A \vee B) = ?$



Experiment set-up

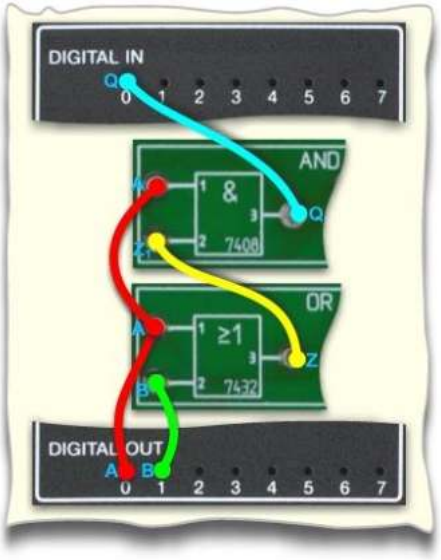


Fig.3.5.12 : Experiment set-up - $Q = A \cdot (A \vee B)$

Procedure:

2.1 Construct the first circuit as shown in *Fig.3.5.12*.

2.2 Observe the response of the output to various logic levels at the input and enter them into *Table 2*. Use the "Extended inputs/outputs" VI.

2.3 Copy the values from your table into *Karnaugh map 2*.

2.4 Confirm the result you obtained from the table using the Karnaugh map.

Table 2

Q_1 Q_0 I_0

B	A	Q
0	0	
0	1	
1	0	
1	1	

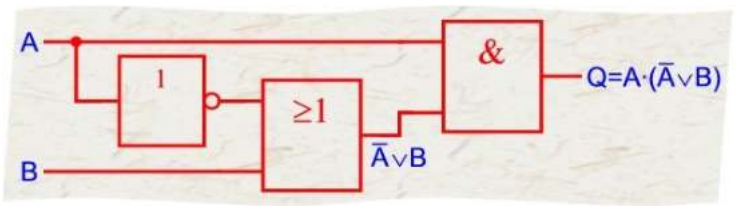
Karnaugh map 2

A	A	
		B
		B

$Q = A \cdot (A \vee B) =$

Exercise 3

$A \cdot (A \vee B) = ?$



Experiment set-up

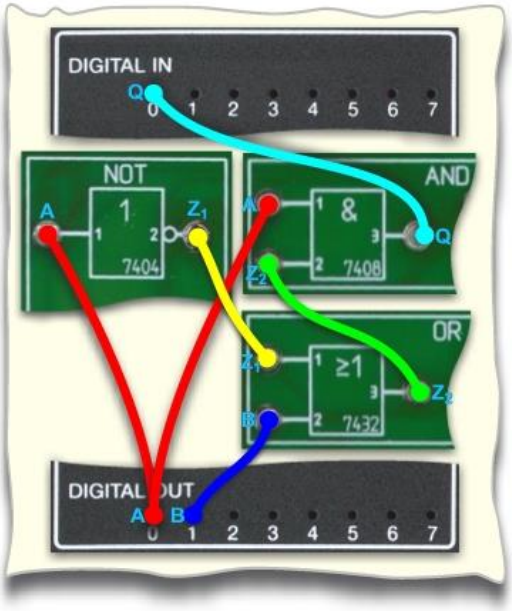


Fig.3.5.13 : Experiment set-up - $Q = A \cdot (A \vee B)$

Procedure:

3.1 Construct the first circuit as shown in *Fig.3.5.13*.

3.2 Observe the response of the output to various logic levels at the input and enter them into *Table 3*. Use the "Extended inputs/outputs" VI.

3.3 Copy the values from your table into *Karnaugh map 3*.

3.4 Confirm the result you obtained from the table using the Karnaugh map.

Table 3

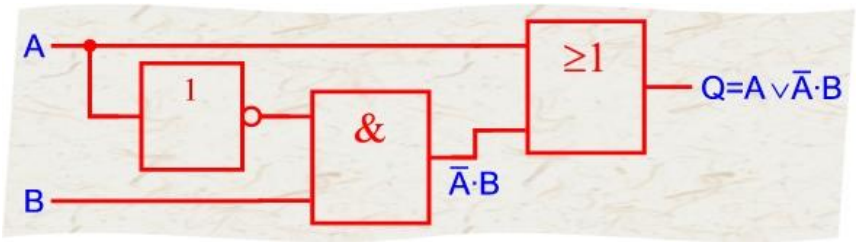
Q_1	Q_0	I_0
B	**A**	**Q**
0	0	
0	1	
1	0	
1	1	

Karnaugh map 3

A	A	
		B
		B

$Q = A \cdot (A \vee B) =$	

Exercise 4 $A \vee (A \cdot B) = ?$



Experiment set-up

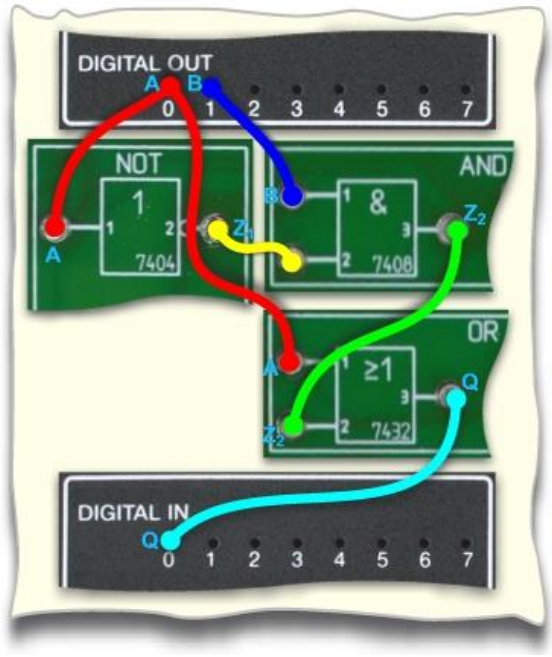


Fig.3.5.14 : Experiment set-up - $Q = A \vee (A \cdot B)$

Procedure:

4.1 Construct the first circuit as shown in *Fig.3.5.14*.

4.2 Observe the response of the output to various logic levels at the input and enter them into *Table 4*. Use the "Extended inputs/outputs" VI.

4.3 Copy the values from your table into *Karnaugh map 4*.

4.4 Confirm the result you obtained from the table using the Karnaugh map.

Table 4

Q_1	Q_0	I_0
B	**A**	**Q**
0	0	
0	1	
1	0	
1	1	

Karnaugh map 4

A	A	
		B
		B

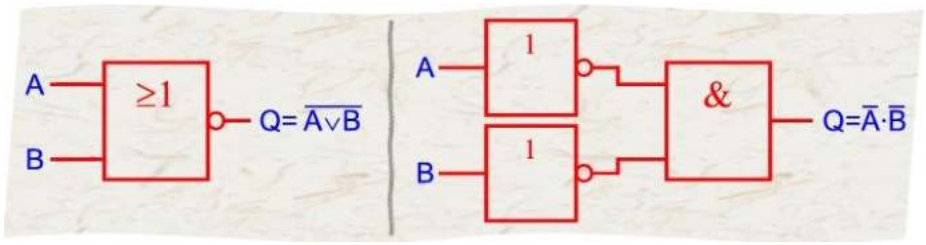
$Q = A \vee (A \cdot B) =$	+

Note: If you need to type in a symbol for OR, it is easiest to use the "+" symbol.

Negation laws (De Morgan's laws) Experiment

These two experiments are intended to provide experimental confirmation of the two laws of De Morgan.

Negation law 1: $\overline{A \vee B} = \overline{A} \cdot \overline{B}$



Experiment set-up

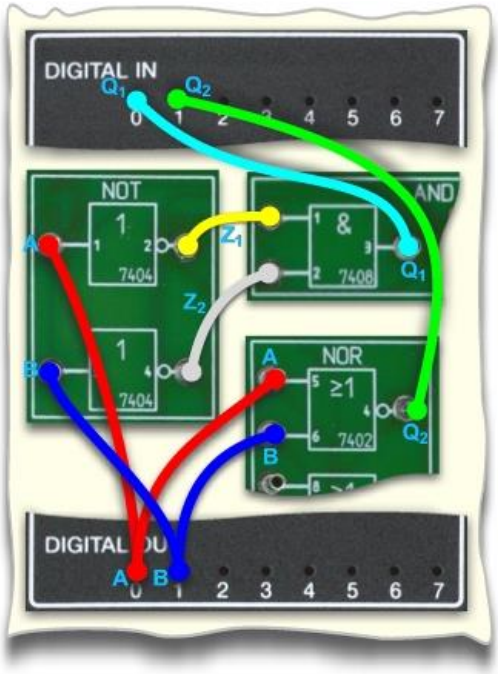


Fig.3.5.15: Experiment set-up -
$Q_1 = A \cdot B$ $Q_2 = \overline{A \vee B}$

In this experiment, both sides of the equation are experimentally reproduced and can be directly compared with one another.

Procedure:

1.1 Construct the first circuit as shown in *Fig.3.5.15*.

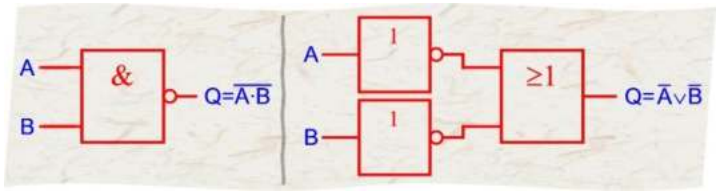
1.2 Observe the response of the output to various logic levels at the input and enter them into *Table 1*. Use the "Extended inputs/outputs" VI.

Table 1

Q_1	Q_0	I_1	I_0
B	**A**	**Q_2**	**Q_1**
0	0		
0	1		
1	0		
1	1		
		↑	↑
		$\overline{A \vee B}$	$A \cdot B$

1.3 Confirm the validity of the negation law from the table.

Negation law 2: $A \cdot B = \overline{A \vee B}$



Experiment set-up

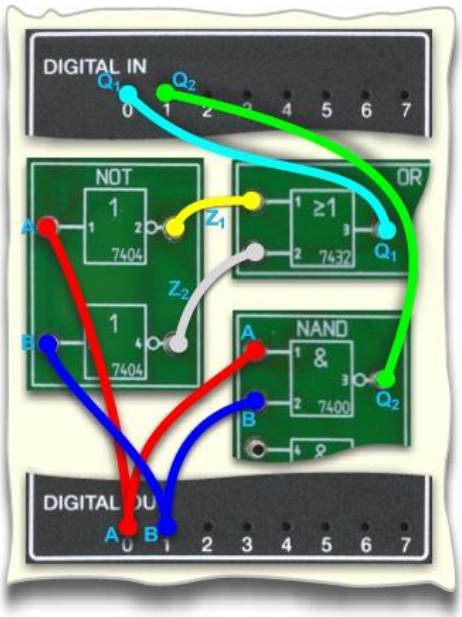


Fig.3.5.16: Experiment set-up -
$Q_1 = A \vee B$ $Q_2 = A \cdot B$

In this experiment, both sides of the equation are experimentally reproduced and can be directly compared with one another.

Procedure:

2.1 Construct the first circuit as shown in *Fig.3.5.16*

2.2 Observe the response of the output to various logic levels at the input and enter them into *Table 2*. Use the "Extended inputs/outputs" VI.

Table 2

Q_1	Q_0	I_1	I_0
B	**A**	Q_2	Q_1
0	0		
0	1		
1	0		
1	1		
		↑	↑
		$A \cdot B$	$A \vee B$

2.3 Confirm the validity of the negation law from the table.

3.6 Exclusive-OR Gate

In the previous experiments , we saw that by using the three principal gates, the AND Gate, the OR Gate and the NOT Gate, we can build many other types of logic gate functions, such as a NAND Gate and a NOR Gate or any other type of digital logic function we can imagine.

But there are two other types of digital logic gates which although they are not a basic gate as they are constructed by combining together other logic gates, their output Boolean function is important enough to be considered as complete logic gates. These two "hybrid" logic gates are called the **Exclusive-OR (Ex-OR) Gate** and its complement the **Exclusive-NOR (Ex-NOR) Gate**.

Previously, we saw that for a 2-input OR gate, if A = "1", **OR** B = "1", **OR BOTH** A + B = "1" then the output from the digital gate must also be at a logic level "1" and because of this, this type of logic gate is known as an Inclusive-OR function. The logic gate gets its name from the fact that it *includes* the case of Q = "1" when both A and B = "1".

If however, an logic output "1" is obtained when **ONLY** A = "1" or when **ONLY** B = "1" but **NOT** both together at the same time, giving the binary inputs of "01" or "10", then the output will be "1". This type of gate is known as an Exclusive-OR function or more commonly an Ex-Or function for short. This is because its boolean expression *excludes* the "**OR BOTH**" case of Q = "1" when both A and B = "1".

In other words the output of an Exclusive-OR gate **ONLY** goes "HIGH" when its two input terminals are at "**DIFFERENT**" logic levels with respect to each other.

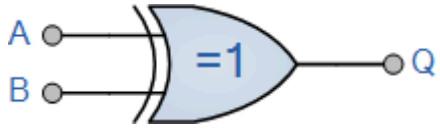
An odd number of logic "1's" on its inputs gives a logic "1" at the output. These two inputs can be at logic level "1" or at logic level "0" giving us the Boolean expression of: $Q = (A \oplus B) = A.B + A.B$

The **Exclusive-OR Gate** function, or **Ex-OR** for short, is achieved by combining standard logic gates together to form more complex gate functions that are used extensively in building arithmetic logic circuits, computational logic comparators and error detection circuits.

The two-input "Exclusive-OR" gate is basically a modulo two adder, since it gives the sum of two binary numbers and as a result are more complex in design than other basic types of logic gate. The truth table, logic symbol and implementation of a 2-input Exclusive-OR gate is shown below.

The Digital Logic "Exclusive-OR" Gate

2-input Ex-OR Gate

Symbol	Truth Table		
 2-input Ex-OR Gate	B	A	Q
	0	0	0
	0	1	1
	1	0	1
	1	1	0
Boolean Expression Q = A ⊕ B	A **OR** B but NOT **BOTH** gives Q		

Giving the Boolean expression of: Q = AB + AB

The truth table above shows that the output of an Exclusive-OR gate ONLY goes "HIGH" when both of its two input terminals are at "DIFFERENT" logic levels with respect to each other. If these two inputs, A and B are both at logic level "1" or both at logic level "0" the output is a "0" making the gate an "odd but not the even gate". In other words, the output is "1" when there are an odd number of 1's in the inputs.

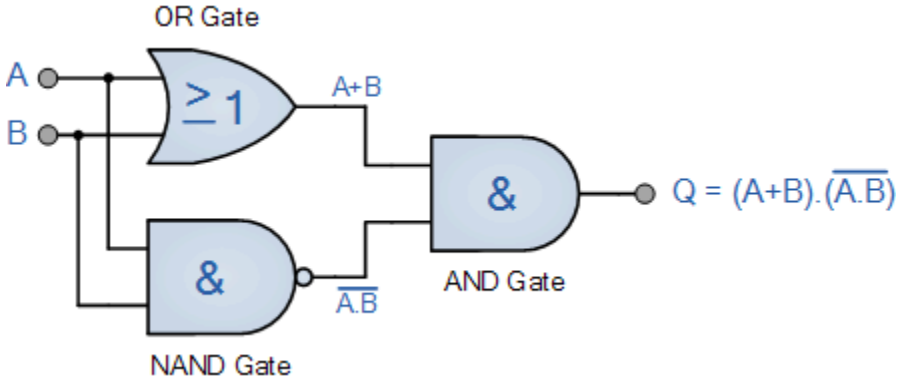
This ability of the *Exclusive-OR gate* to compare two logic levels and produce an output value dependent upon the input condition is very useful in computational logic circuits as it gives us the following Boolean expression of:

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

The logic function implemented by a 2-input Ex-OR is given as either: "A OR B but NOT both" will give an output at Q. In general, an Ex-OR gate will give an output value of logic "1" ONLY when there are an **ODD** number of 1's on the inputs to the gate, if the two numbers are equal, the output is "0".

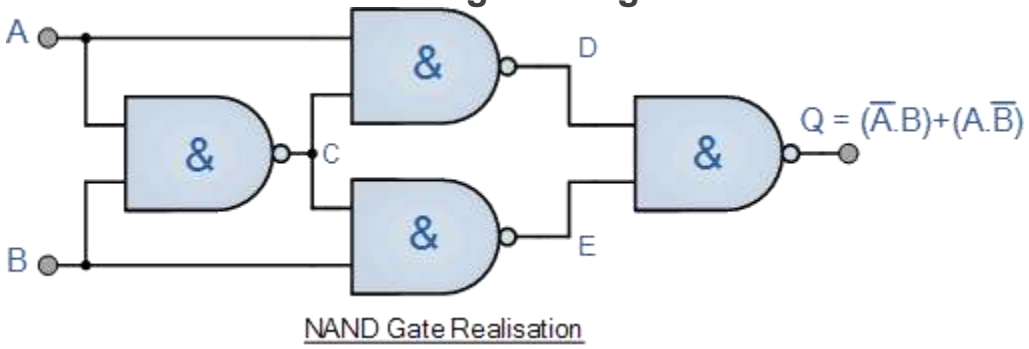
Then an Ex-OR function with more than two inputs is called an "odd function" or modulo-2-sum (Mod-2-SUM), not an Ex-OR. This description can be expanded to apply to any number of individual inputs as shown below for a 3-input Ex-OR gate.

Ex-OR Gate Equivalent Circuit



One of the main disadvantages of implementing the Ex-OR function above is that it contains three different types logic gates OR, NAND and finally AND within its design. One easier way of producing the Ex-OR function from a single gate is to use our old favorite the NAND gate as shown below.

Ex-OR Function Realization using NAND gates



Exclusive-OR Gates are used mainly to build circuits that perform arithmetic operations and calculations especially **Adders** and **Half-Adders** as they can provide a “carry-bit” function or as a controlled inverter, where one input passes the binary data and the other input is supplied with a control signal.

Commonly available digital logic Exclusive-OR gate IC's include:

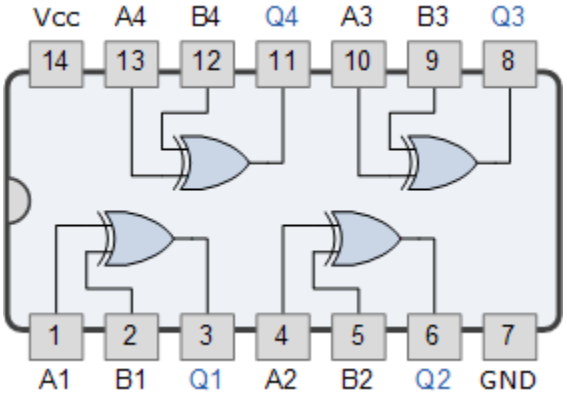
TTL Logic Ex-OR Gates

- 74LS86 Quad 2-input

CMOS Logic Ex-OR Gates

- CD4030 Quad 2-input

7486 Quad 2-input Exclusive-OR Gate

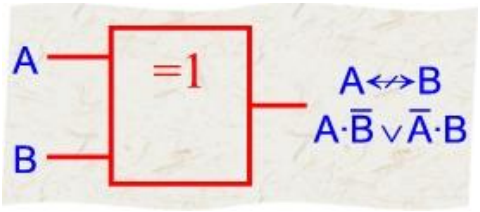


The **Exclusive-OR** logic function is a very useful circuit that can be used in many different types of computational circuits. Although not a basic logic gate in its own right, its usefulness and versatility has turned it into a standard logical function complete with its own Boolean expression, operator and symbol. The *Exclusive-OR Gate* is widely available as a standard quad two-input 74LS86 TTL gate or the 4030B CMOS package.

One of its most commonly used applications is as a basic logic comparator which produces a logic "1" output when its two input bits are not equal. Because of this, the exclusive-OR gate has an inequality status being known as an odd function. In order to compare numbers that contain two or more bits, additional exclusive-OR gates are needed with the 74LS85 logic comparator being 4-bits wide.

In the next tutorial about **Digital Logic Gates**, we will look at the digital logic *Exclusive-NOR* gate known commonly as the Ex-NOR Gate function as used in both TTL and CMOS logic circuits as well as its Boolean Algebra definition and truth tables.

EXOR (Antivalence) experiment



Circuit diagram symbol for Exclusive OR (abbr.: EXOR, XOR)

The symbol for EXOR is that shown in the illustration. The operation is termed "Exclusive OR" or "Antivalence".

The logical function is given as $Q = A \cdot B \vee A \cdot B$. This statement will be tested in the following experiment.

[**The exclusive OR might be thought of as the "John-Wayne OR":**
"... Stranger, this town ain't big enough for the both of us ...
one of us is gonna have to leave, either YOU or ME"]

Exercise 1:

Experiment set-up

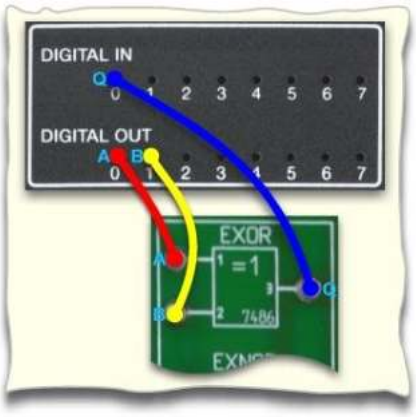


Fig.3.6.1 : Experiment set-up - EXOR

Procedure:

1.1 Construct the circuit as shown in *Fig.3.6.1*
1.2 Observe the response of the output to various logic levels at the input and enter them into *Table 1*. Use the "Extended inputs/outputs" VI.

Tip:
You can alter the bit status quickly and easily by using the up and down buttons on the VI.

Table 1

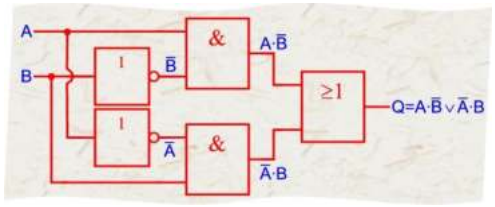
Q_1	Q_0	I_0
B	**A**	**Q**
0	0	
0	1	
1	0	
1	1	

EXOR

Describe how an EXOR gate works.
Why is it also known as an "anti-valence" (inequality) gate?

……
……

Exercise 2:



This is how an EXOR gate is reproduced using OR/AND/NOT gates.

Experiment set-up

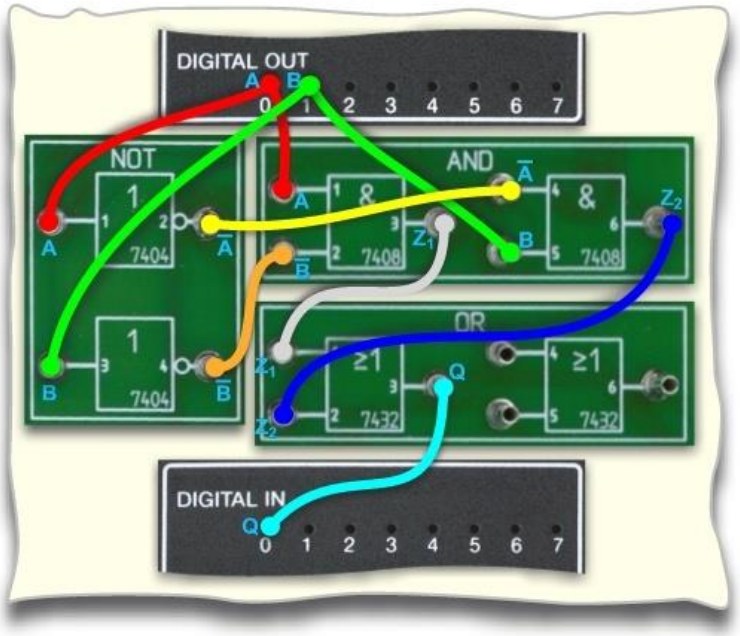


Fig.3.6.2: Experiment set-up - Q = A·B ∨ A·B

Procedure:

2.1 Construct the circuit according to *Fig.3.6.2*.

2.2 Observe the response of the output to various logic levels at the input and enter them into *Table 2*. Use the "Extended inputs/outputs" VI.

Table 2

Q_1	Q_0	I_0
B	**A**	**Q**
0	0	
0	1	
1	0	
1	1	

↑

$A \cdot B \vee A \cdot B$

2.3 Confirm from the values in Tables 1 and 2 that the EXOR function can be represented by the function $Q = A \cdot B \vee A \cdot B$.

3.7 Exclusive-NOR Gate

Basically the “Exclusive-NOR” gate is a combination of the Exclusive-OR gate and the NOT gate but has a truth table similar to the standard NOR gate in that it has an output that is normally at logic level “1” and goes “LOW” to logic level “0” when **ANY** of its inputs are at logic level “1”.

However, an output “1” is only obtained if **BOTH** of its inputs are at the same logic level, either binary “1” or “0”. For example, “00” or “11”. This input combination would then give us the Boolean expression of: $Q = (A \oplus B) = A.B + A.B$

Then the output of a digital logic Exclusive-NOR gate **ONLY** goes “HIGH” when its two input terminals, A and B are at the “**SAME**” logic level which can be either at a logic level “1” or at a logic level “0”. In other words, an even number of logic “1’s” on its inputs gives a logic “1” at the output, otherwise is at logic level “0”.

Then this type of gate gives and output “1” when its inputs are “*logically equal*” or “*equivalent*” to each other, which is why an **Exclusive-NOR** gate is sometimes called an **Equivalence Gate**.

The logic symbol for an Exclusive-NOR gate is simply an Exclusive-OR gate with a circle or “inversion bubble”, (o) at its output to represent the NOT function. Then the **Logic Exclusive-NOR Gate** is the reverse or “*Complementary*” form of the Exclusive-OR gate, $(A \oplus B)$ we have seen previously.

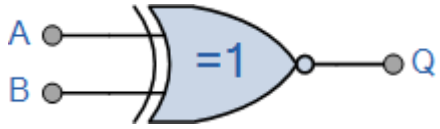
Ex-NOR Gate Equivalent

2-input "Ex-OR" gate plus a "NOT" gate

The **Exclusive-NOR Gate**, also written as: “Ex-NOR” or “XNOR”, function is achieved by combining standard gates together to form more complex gate functions and an example of a 2-input Exclusive-NOR gate is given below.

The Digital Logic "Ex-NOR" Gate

2-input Ex-NOR Gate

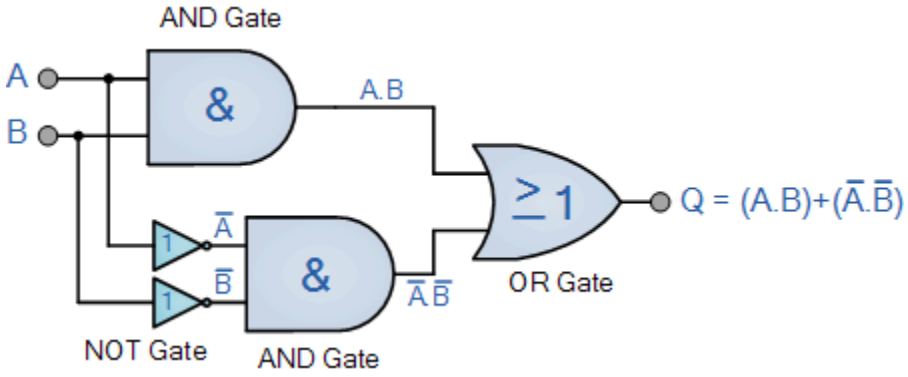
Symbol	Truth Table		
 2-input Ex-NOR Gate	B	A	Q
	0	0	1
	0	1	0
	1	0	0
	1	1	1
Boolean Expression Q = A ⊕ B	Read if A **AND** B the **SAME** gives Q		

Giving the Boolean expression of: Q = $\bar{A}B + A\bar{B}$

The logic function implemented by a 2-input Ex-NOR gate is given as "when both A AND B are the SAME" will give an output at Q. In general, an Exclusive-NOR gate will give an output value of logic "1" ONLY when there are an **EVEN** number of 1's on the inputs to the gate (the inverse of the Ex-OR gate) except when all its inputs are "LOW".

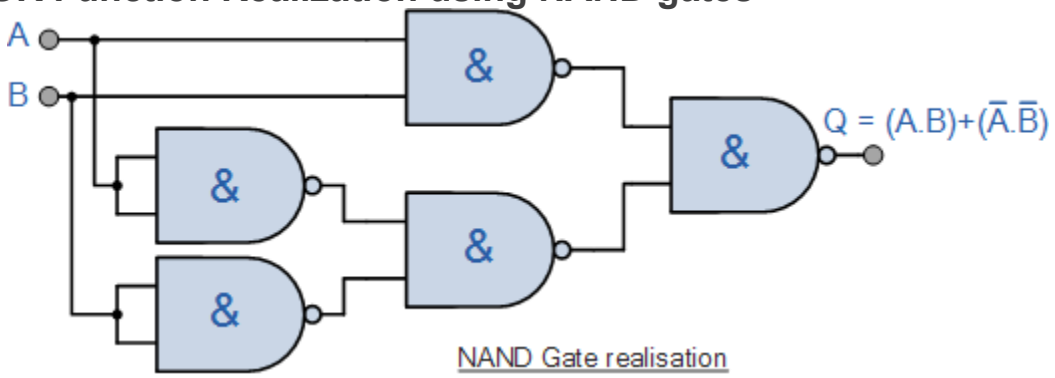
Then an Ex-NOR function with more than two inputs is called an "even function" or modulo-2-sum (Mod-2-SUM), not an Ex-NOR. This description can be expanded to apply to any number of individual inputs as shown below for a 3-input Exclusive-NOR gate.

Ex-NOR Gate Equivalent Circuit



One of the main disadvantages of implementing the Ex-NOR function above is that it contains three different types logic gates the AND, NOT and finally an OR gate within its basic design. One easier way of producing the Ex-NOR function from a single gate type is to use NAND gates as shown below.

Ex-NOR Function Realization using NAND gates



NAND Gate realisation

Ex-NOR gates are used mainly in electronic circuits that perform arithmetic operations and data checking such as *Adders*, *Sub tractors* or *Parity Checkers*, etc. As the Ex-NOR gate gives an output of logic level "1" whenever its two inputs are equal it can be used to compare the magnitude of two binary digits or numbers and so Ex-NOR gates are used in Digital Comparator circuits.

Commonly available digital logic Exclusive-NOR gate IC's include:

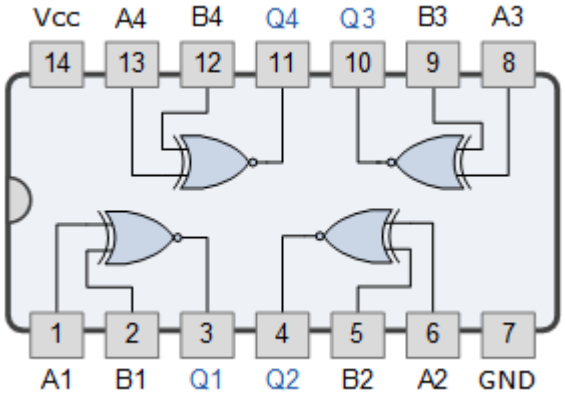
TTL Logic Ex-NOR Gates

- 74LS266 Quad 2-input

CMOS Logic Ex-NOR Gates

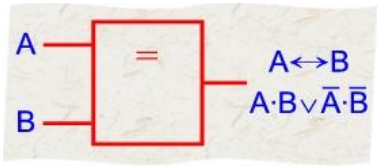
- CD4077 Quad 2-input

74266 Quad 2-input Ex-NOR Gate

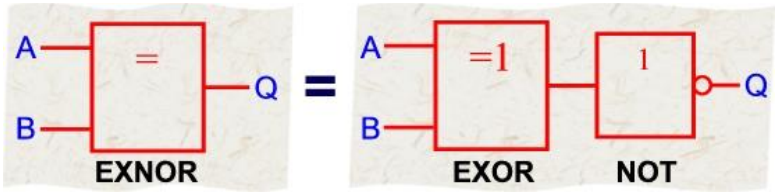


In the next tutorial about **Digital Logic Gates**, we will look at the digital Tri-state Buffer also called the non-inverting buffer as used in both TTL and CMOS logic circuits as well as its Boolean Algebra definition and truth table.

EXNOR (Equivalence) Experiment



Circuit diagram symbol of an EXNOR gate



An EXNOR operation simply corresponds to an inverted EXOR

The symbol for EXNOR is that shown in the illustration. The operation is termed "Exclusive NOR" or "Equivalence".

The logical function is given as $Q = A \cdot B \vee A \cdot B$. This statement will be tested in the following experiment.

Exercise 1:Experiment set-up

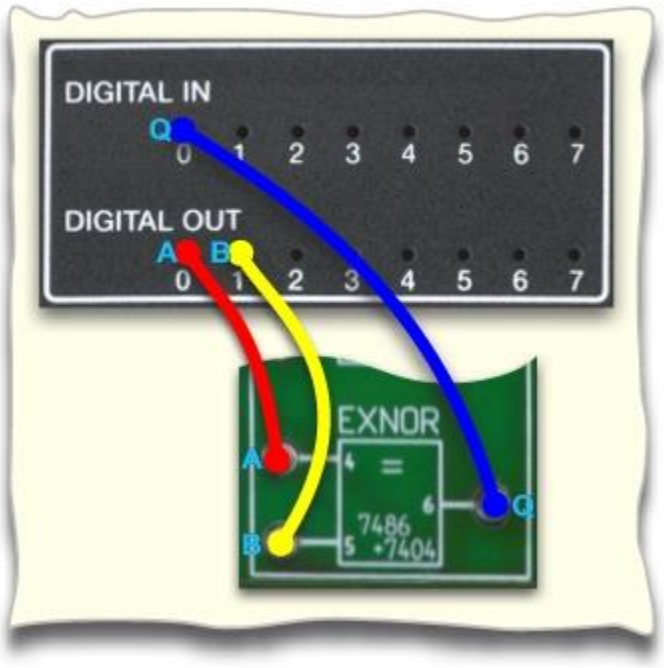


Fig.3.7.1 : Experiment set-up - EXNOR

Procedure:

1.1 Construct the circuit as shown in *Fig.3.7.1*.

1.2 Observe the response of the output to various logic levels at the input and enter them into *Table 1*. Use the "Extended inputs/outputs" VI.

Tip:
You can alter the bit status quickly and easily by using the up and down buttons on the VI.

Table 1

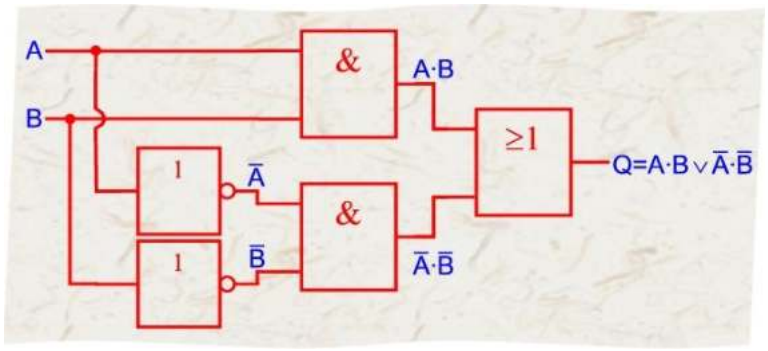
Q_1	Q_0	I_0
B	**A**	**Q**
0	0	
0	1	
1	0	
1	1	

EXNOR

Describe how an EXNOR gate works
Why is it also known as an "equivalence" (equality) gate?

……
……

Exercise 2:



This is how an EXNOR gate is reproduced using OR/AND/NOT gates.

Experiment set-up

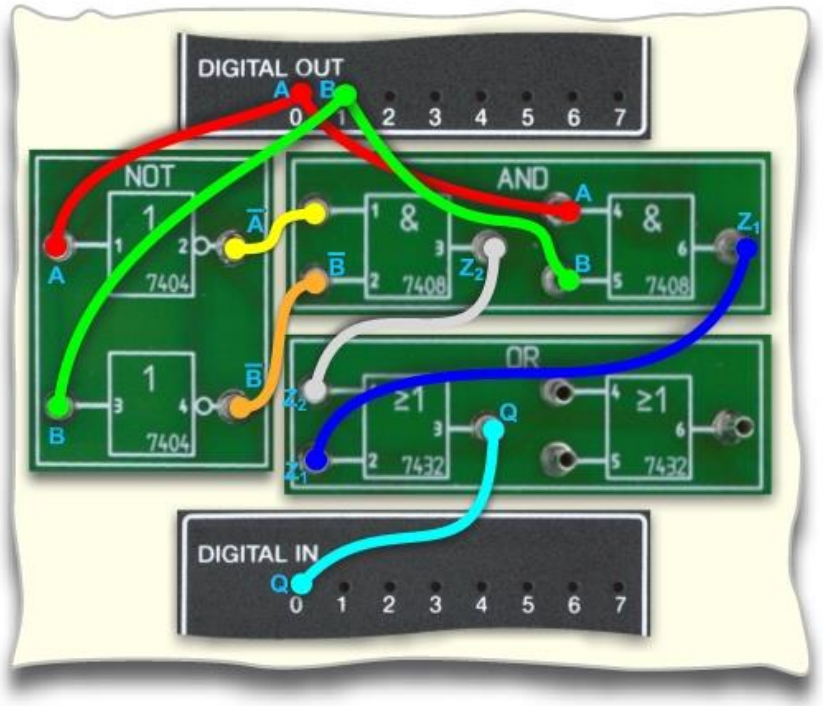


Fig.3.7.2 : Experiment set-up - $Q = A \cdot B \vee A \cdot B$

Procedure:

2.1 Construct the circuit according to *Fig.3.7.2*.

2.2 Observe the response of the output to various logic levels at the input and enter them into *Table 2*. Use the "Extended inputs/outputs" VI.

Table 2

Q_1	Q_0	I_0
B	**A**	**Q**
0	0	
0	1	
1	0	
1	1	

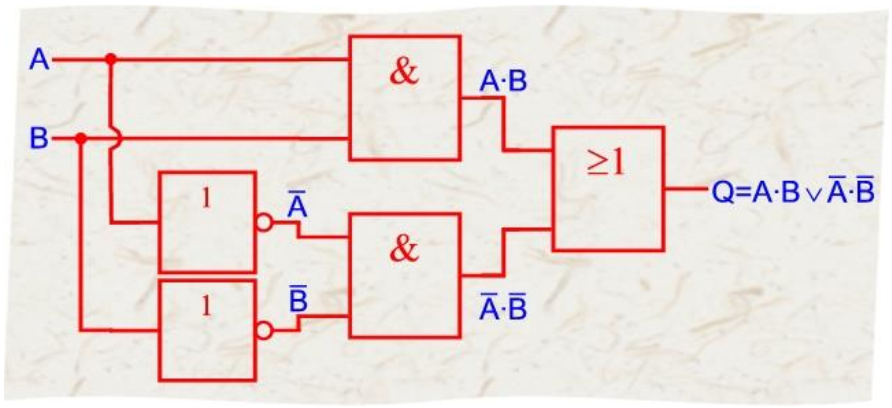
↑

$A \cdot B \vee A \cdot B$

2.3 Confirm from the values in Tables 1 and 2 that the EXNOR function can be represented by the function $Q = A \cdot B \vee \overline{A} \cdot \overline{B}$.

Question:
Can you think of a way to achieve the same result using only three gates?
You can try out your suggestion on the experiment card.

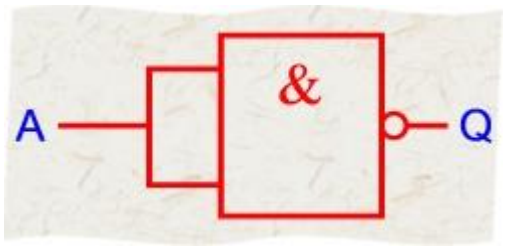
……
……
……



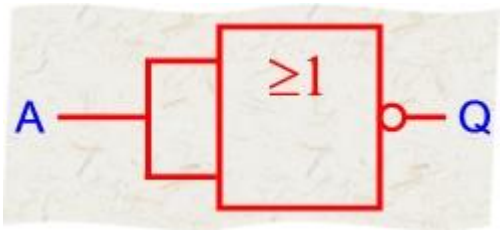
3.8 Realizing gates by combining NAND / NOR gates

Due to the nature of their manufacture, NAND and NOR gates are by far the easiest and cheapest to make as components. For this reason, it is often more economical to realize other gates by combining NAND or NOR components.

3.8.1 NOT function using NAND/NOR gates



NOT function realized using a NAND gate



NOT function realized using a NOR gate

The NOT function can be realized using a NAND or a NOR gate.

Complete the mathematical equations:

Since $\square \cdot \square = \square$ then: $Q = \overline{\square \cdot \square} = \overline{\square}$

and

since $\square \vee \square = \square$ then: $Q = \overline{\square \vee \square} = \overline{\square}$

Exercise 1:

Experiment set-up

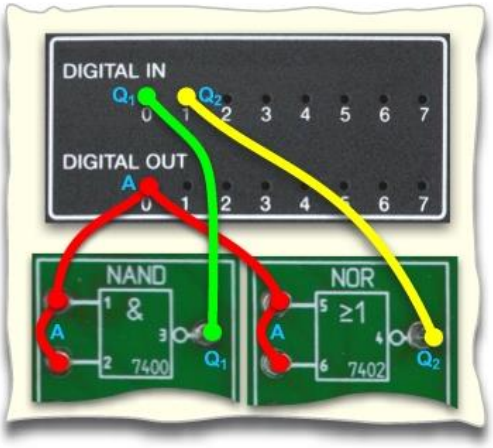


Fig.3.8.1.1: Experiment set-up -
NOT function using NAND and NOR gates

Procedure:

1.1 Construct the circuit as shown in *Fig.3.8.1.1*.

1.2 Observe the response of the output to various logic levels at the input and enter them into *Table 1*. Use the "Extended inputs/outputs" VI.

Table 1

Q_0	I_1	I_0
A	Q_2	Q_1
0		
1		

NOT function using NOR gate

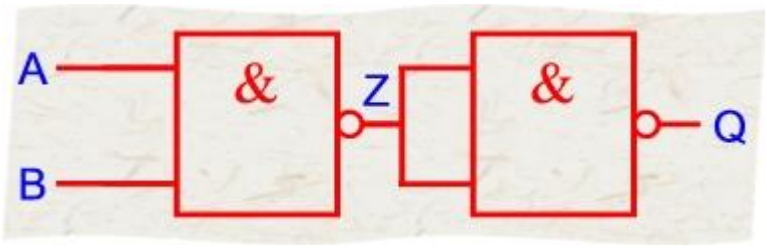
NOT function using NAND gate

1.3 Confirm based on the table that the expected inversion takes place.

3.8.2 AND function using NAND/NOR gates

AND function using NAND gates

Since a NAND function is just an inverted AND, an AND gate can be implemented by the addition of an inverter after a NAND gate. This inverter can also be implemented using a NAND gate.



Complete the formulae:

$Z = \overline{\square}$; $Q = \overline{\square} = \overline{\overline{\square}} = \square$

Exercise 1:

Experiment set-up

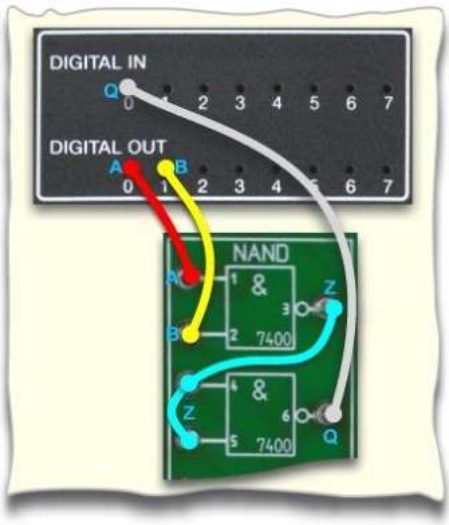


Fig.3.8.2.1 : Experiment set-up -
AND function realized using NAND gates

Procedure:

1.1 Construct the circuit as shown in *Fig.3.8.2.1*

1.2 Observe the response of the output to various logic levels at the input and enter them into *Table 1*. Use the "Extended inputs/outputs" VI.

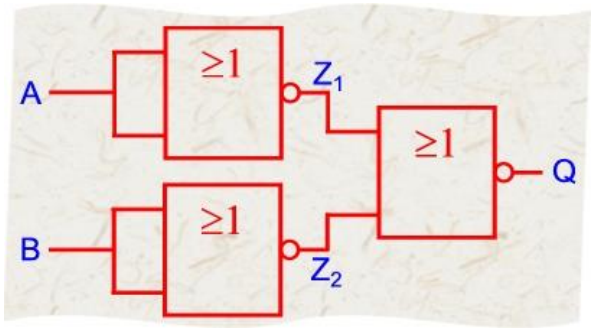
Table 1

Q_1	Q_0	I_0
B	**A**	**Q**
0	0	
0	1	
1	0	
1	1	

1.3 Confirm from the values in the table that the AND function has been successfully realized.

AND function using NOR gates

According to De Morgan, a NOR can be regarded as an AND gate with inverted inputs. Thus, if each individual input to a NOR gate is inverted beforehand, what results is an AND gate.



Complete these formulae:

$Z_1 = \overline{\square}$; $Z_2 = \overline{\square}$; $Q = \overline{\square \vee \square} = \overline{\square} \cdot \overline{\square}$

$Q = \overline{\overline{\square}} \cdot \overline{\overline{\square}} = \square$

" . "

Exercise 2:

Experiment set-up

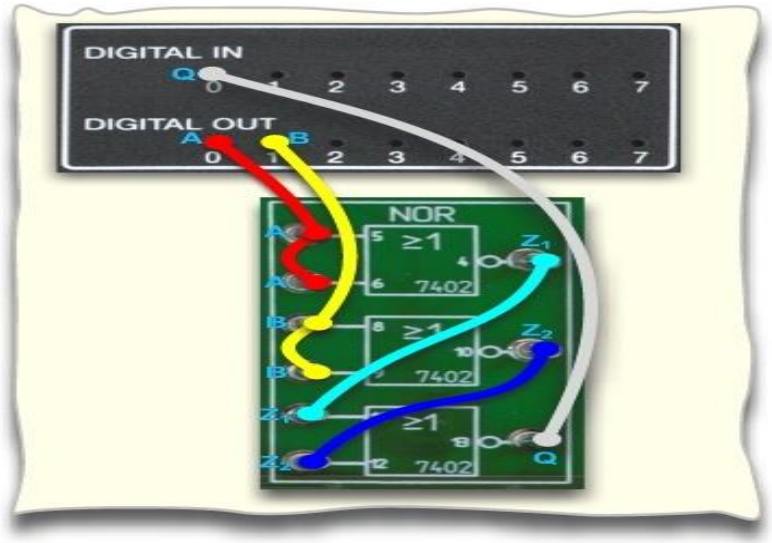


Fig.3.8.2.2 : Experiment set-up -
AND function using NOR gates

Procedure:

2.1 Construct the circuit as shown in *Fig.3.8.2.2*.

2.2 Observe the response of the output to various logic levels at the input and enter them into *Table 2*. Use the "Extended inputs/outputs" VI.

Table 2

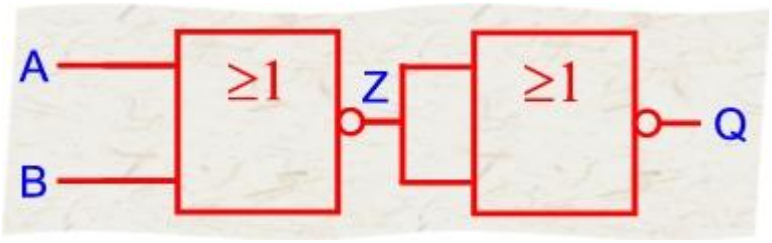
Q_1	Q_0	I_0
B	**A**	**Q**
0	0	
0	1	
1	0	
1	1	

2.3 Confirm from the values in the table that the AND function has been realized.

3.8.3 OR function realized using NAND / NOR gates

OR function using NOR gates

Since the OR function is simply the inversion of a NOR, it can be realised by placing an inverter (NOT) after a NOR gate. The inverter can also be implemented using a NOR gate.



Complete these formulae:

$Z = \overline{\square + \square}$; $Q = \overline{\square} = \overline{\overline{\square \; \square}} = \square$

Exercise 1:

Experiment set-up

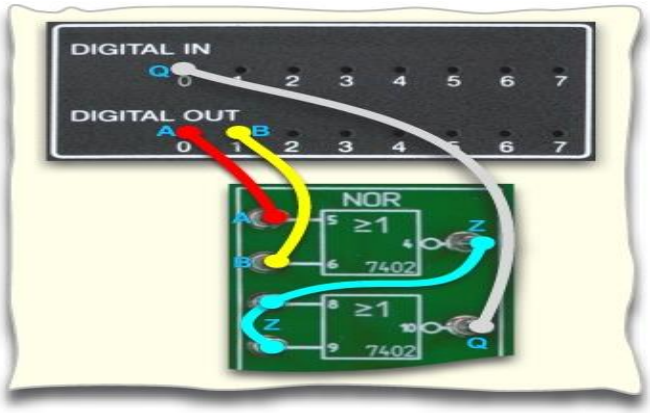


Fig.3.8.3.1 : Experiment set-up -
OR function realized using NOR gates

Procedure:

1.1 Construct the circuit as shown in *Fig.3.8.3.1*.
1.2 Observe the response of the output to various logic levels at the input and enter them into *Table 1*. Use the "Extended inputs/outputs" VI.

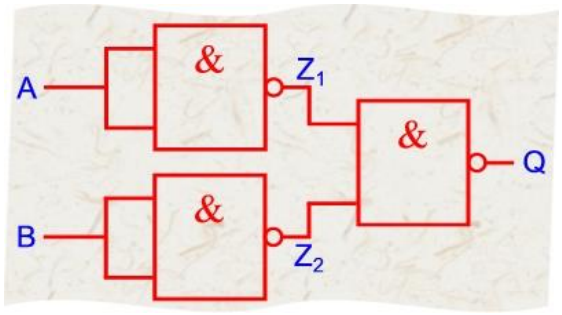
Table 1

Q_1	Q_0	I_0
B	**A**	**Q**
0	0	
0	1	
1	0	
1	1	

1.3 Confirm based on the table that the OR functionality has been successfully realized.

OR using NAND gates

According to De Morgan's laws a NAND operation is identical to an OR with the inputs inverted. Thus, an OR function can simply be realized by inverting each of the inputs to a NAND gate.



Complete these formulae:

$Z_1 = \overline{\square}$; $Z_2 = \overline{\square}$; $Q = \overline{\square \cdot \square} = \overline{\square} \vee \overline{\square}$

$Q = \overline{\overline{\square}} \vee \overline{\overline{\square}} = \square^{+}$ "$\vee$" → "+"

Exercise 2:

Experiment set-up

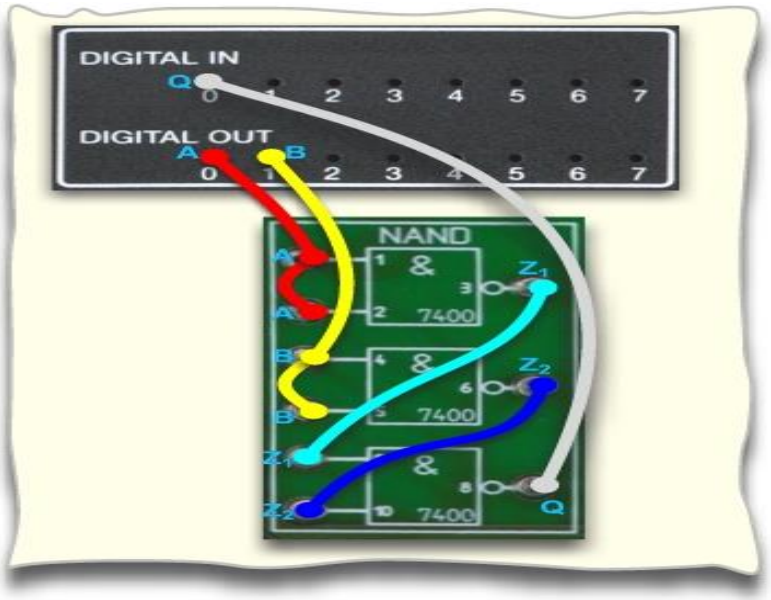


Fig.3.8.3.2 : Experiment set-up -
OR function realized using NAND gates

Procedure:

2.1 Construct the circuit as shown in *Fig.3.8.3.2*.
2.2 Observe the response of the output to various logic levels at the input and enter them into *Table 2*. Use the "Extended inputs/outputs" VI.

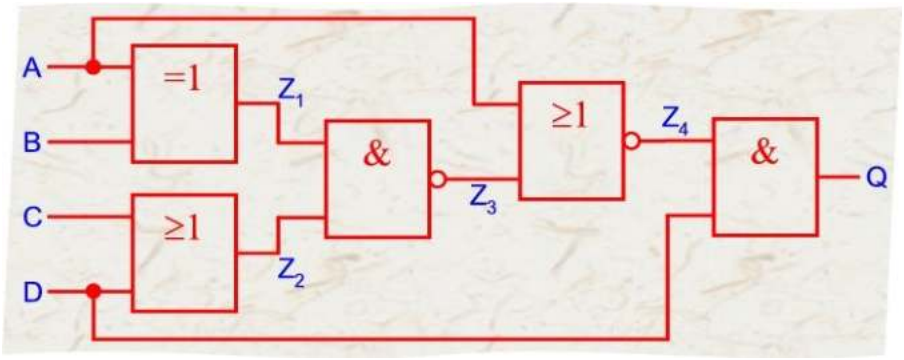
Table 2

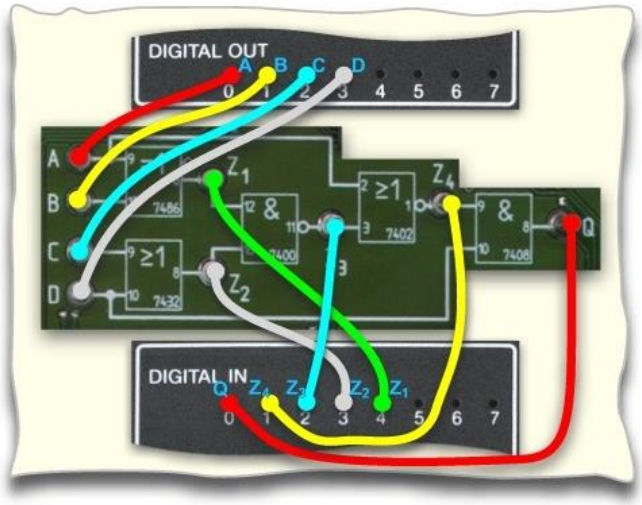
Q_1	Q_0	I_0
B	**A**	**Q**
0	0	
0	1	
1	0	
1	1	

2.3 Confirm based on the table that the OR functionality has been successfully realized

Fault simulation

Fault 1





Exercise

Identify the fault in the circuit. Describe the fault and suggest a remedy:

Procedure:
1.1 Construct the initial circuit.
1.2 Observe the response of the output to various logic levels at the input and enter them into *Table 1*. Use the "Extended inputs/outputs" VI.

Tip:
You can alter the bit status quickly and easily by using the up and down buttons on the VI.

1.3 From *Table 1*, transfer the appropriate values to the 5 smaller tables. This should enable you to recognize the functioning of the individual gates. If certain combinations of bits do not occur, enter a "?".

Table 1

	Q_3	Q_2	Q_1	Q_0	I_4	I_3	I_2	I_1	I_0
	D	C	B	A	Z_1	Z_2	Z_3	Z_4	Q
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					

Q_1	Q_0	I_4
B	A	Z_1
0	0	
0	1	
1	0	
1	1	

Q_3	Q_2	I_3
D	C	Z_2
0	0	
0	1	
1	0	
1	1	

I_4	I_3	I_2
Z_1	Z_2	Z_3
0	0	
0	1	
1	0	
1	1	

Q_0	I_2	I_1
A	Z_3	Z_4
0	0	
0	1	
1	0	
1	1	

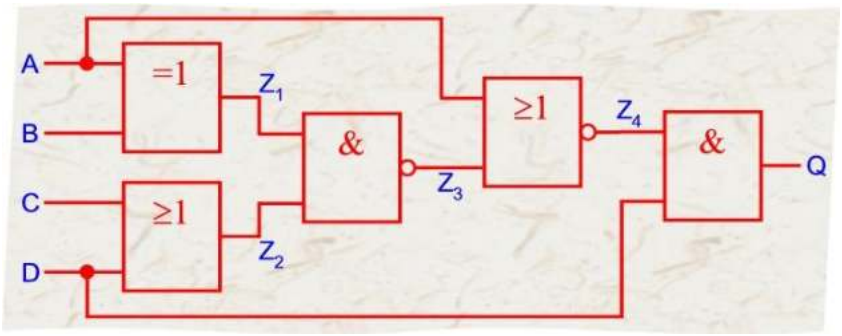
Q_3	I_1	I_0
D	Z_4	Q
0	0	
0	1	
1	0	
1	1	

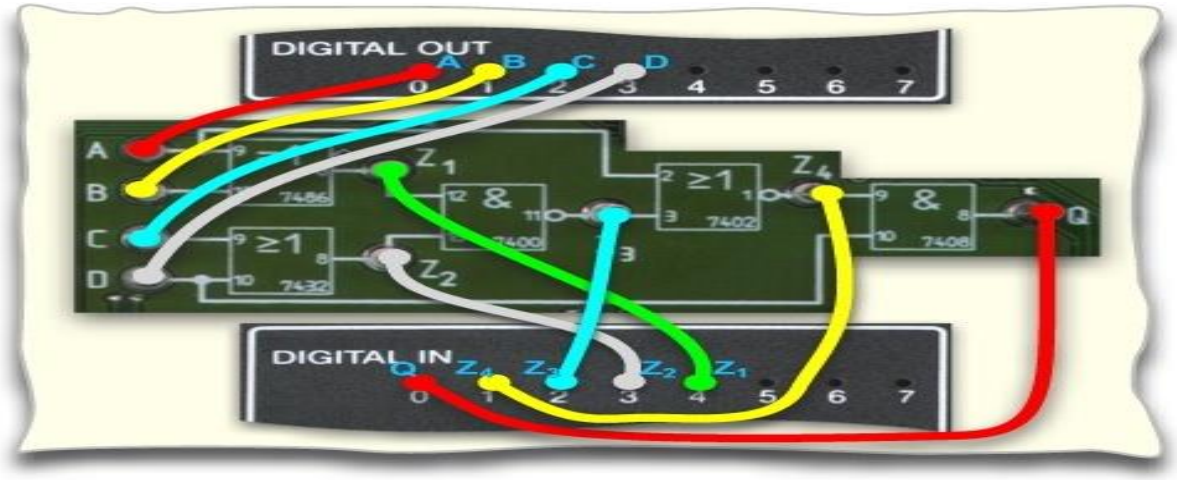
What is the fault?

……

How might such a fault occur?

Fault 2





Exercise

Identify the fault in the circuit. Describe the fault and suggest a remedy:

2.1 Construct the initial circuit.

2.2 Observe the response of the output to various logic levels at the input and enter them into *Table 2*. Use the "Extended inputs/outputs" VI.

Tip:
You can alter the bit status quickly and easily by using the up and down buttons on the VI.

2.3 From *Table 2*, transfer the appropriate values to the 5 smaller tables. This should enable you to recognize the functioning of the individual gates. If certain combinations of bits do not occur, enter a "?".

Table 2

	Q_3	Q_2	Q_1	Q_0	I_4	I_3	I_2	I_1	I_0
	D	**C**	**B**	**A**	**Z_1**	**Z_2**	**Z_3**	**Z_4**	**Q**
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					

Q_1	Q_0	I_4
B	**A**	**Z_1**
0	0	
0	1	
1	0	
1	1	

Q_3	Q_2	I_3
D	**C**	**Z_2**
0	0	
0	1	
1	0	
1	1	

I_4	I_3	I_2
Z_1	**Z_2**	**Z_3**
0	0	
0	1	
1	0	
1	1	

Q_0	I_2	I_1
A	**Z_3**	**Z_4**
0	0	
0	1	
1	0	
1	1	

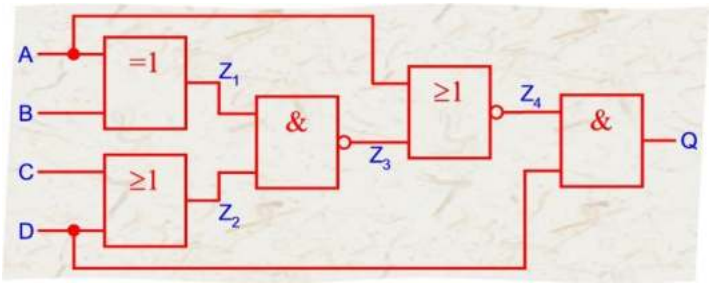
Q_3	I_1	I_0
D	**Z_4**	**Q**
0	0	
0	1	
1	0	
1	1	

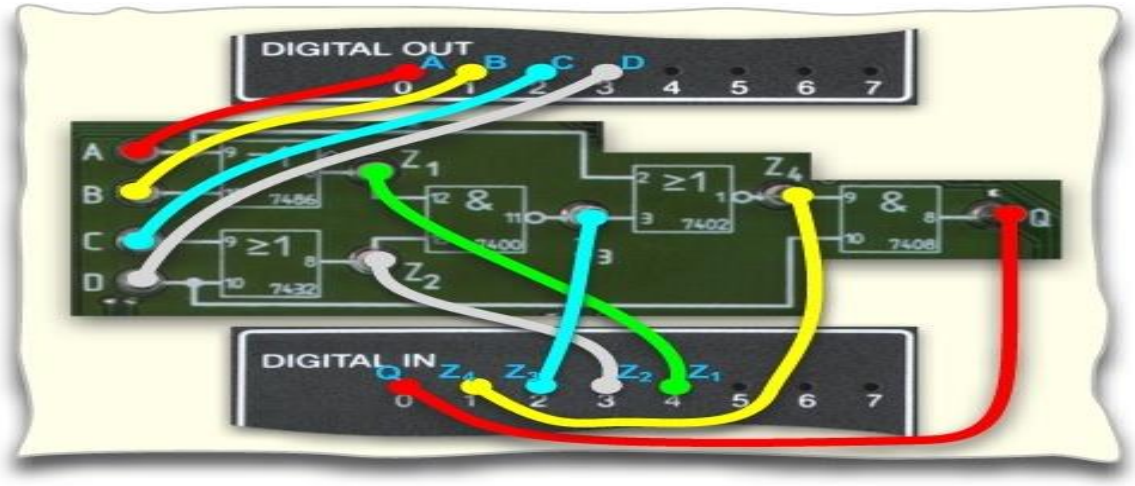
What is the fault?

……

What might cause such a fault to occur?

Fault 3





Exercise

Identify the fault in the circuit. Describe the fault and suggest a remedy:

3.1 Construct the first circuit.

3.2 Observe the response of the output to various logic levels at the input and enter them into *Table 1*. Use the "Extended input and output" VI.

Tip:
You can alter the bit status quickly and easily by using the up and down buttons on the VI.

2.3 From *Table 3*, transfer the appropriate values to the 5 smaller tables. This should enable you to recognize the functioning of the individual gates. If certain combinations of bits do not occur, enter a "?".

Table 3

	Q_3	Q_2	Q_1	Q_0	I_4	I_3	I_2	I_1	I_0
	D	**C**	**B**	**A**	**Z_1**	**Z_2**	**Z_3**	**Z_4**	**Q**
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					

Q_1	Q_0	I_4
B	**A**	**Z_1**
0	0	
0	1	
1	0	
1	1	

Q_3	Q_2	I_3
D	**C**	**Z_2**
0	0	
0	1	
1	0	
1	1	

I_4	I_3	I_2
Z_1	**Z_2**	**Z_3**
0	0	
0	1	
1	0	
1	1	

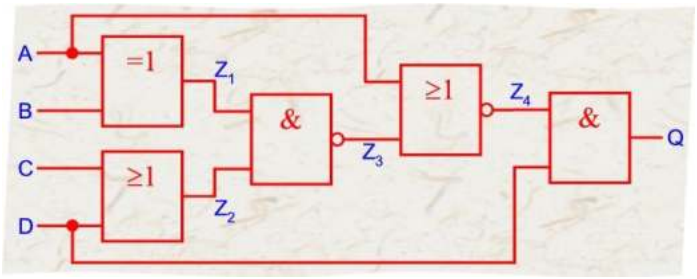
Q_0	I_2	I_1
A	**Z_3**	**Z_4**
0	0	
0	1	
1	0	
1	1	

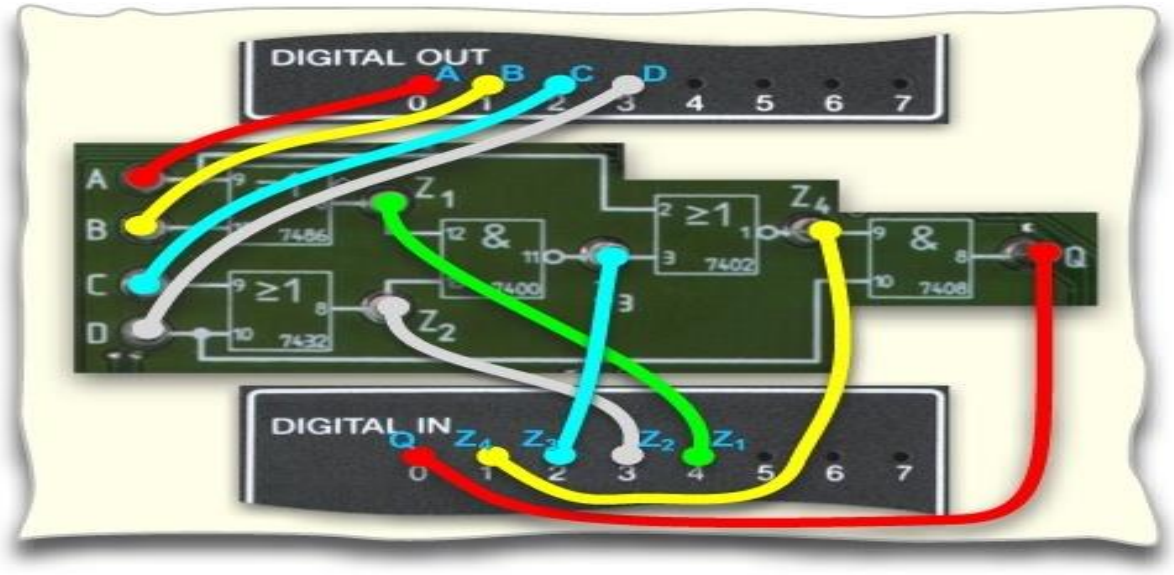
Q_3	I_1	I_0
D	**Z_4**	**Q**
0	0	
0	1	
1	0	
1	1	

What is the fault?

What might cause such a fault to occur?

Fault 4





Exercise

Identify the fault in the circuit. Describe the fault and suggest a remedy:

4.1 Construct the first circuit.

4.2 Observe the response of the output to various logic levels at the input and enter them into *Table 4*. Use the "Extended inputs/outputs" VI.

Tip:
You can alter the bit status quickly and easily by using the up and down buttons on the VI.

3.3 From *Table 4*, transfer the appropriate values to the 5 smaller tables. This should enable you to recognize the functioning of the individual gates. If certain combinations of bits do not occur, enter a "?".

Table 4

	Q_3	Q_2	Q_1	Q_0	I_4	I_3	I_2	I_1	I_0
	D	**C**	**B**	**A**	**Z_1**	**Z_2**	**Z_3**	**Z_4**	**Q**
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					

Q_1	Q_0	I_4
B	**A**	**Z_1**
0	0	
0	1	
1	0	
1	1	

Q_3	Q_2	I_3
D	**C**	**Z_2**
0	0	
0	1	
1	0	
1	1	

I_4	I_3	I_2
Z_1	**Z_2**	**Z_3**
0	0	
0	1	
1	0	
1	1	

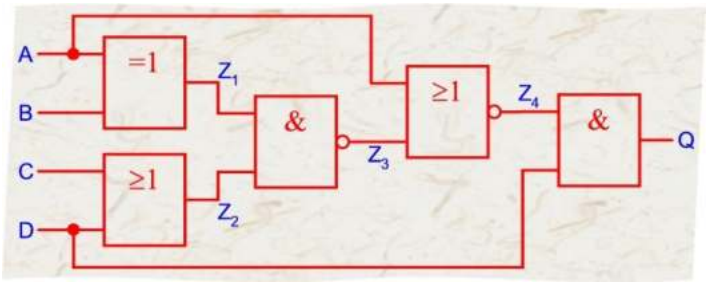
Q_0	I_2	I_1
A	**Z_3**	**Z_4**
0	0	
0	1	
1	0	
1	1	

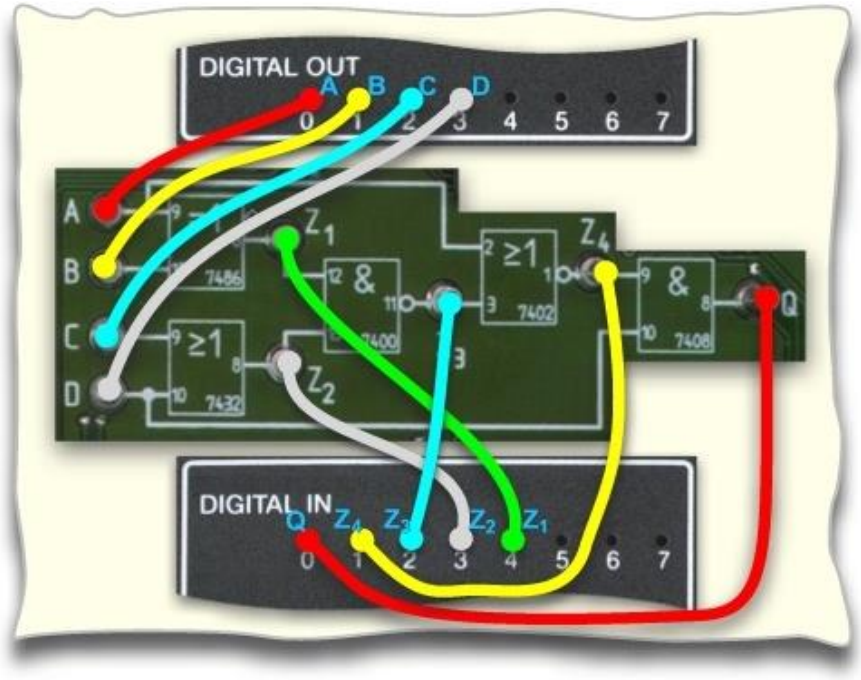
Q_3	I_1	I_0
D	**Z_4**	**Q**
0	0	
0	1	
1	0	
1	1	

What is the fault?

What might cause such a fault to occur?

Fault 5





Exercise

Identify the fault in the circuit. Describe the fault and suggest a remedy:

5.1 Construct the initial circuit.

5.2 Observe the response of the output to various logic levels at the input and enter them into *Table 5*. Use the "Extended input and output" VI.

Tip:
You can alter the bit status quickly and easily by using the up and down buttons on the VI.

5.3 From *Table 5*, transfer the appropriate values to the 5 smaller tables. This should enable you to recognize the functioning of the individual gates. If certain combinations of bits do not occur, enter a "?".

Table 5

	Q_3 D	Q_2 C	Q_1 B	Q_0 A	I_4 Z_1	I_3 Z_2	I_2 Z_3	I_1 Z_4	I_0 Q
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					

Q_1 Q_0 I_4

B	A	Z_1
0	0	
0	1	
1	0	
1	1	

Q_3 Q_2 I_3

D	C	Z_2
0	0	
0	1	
1	0	
1	1	

I_4 I_3 I_2

Z_1	Z_2	Z_3
0	0	
0	1	
1	0	
1	1	

Q_0 I_2 I_1

A	Z_3	Z_4
0	0	
0	1	
1	0	
1	1	

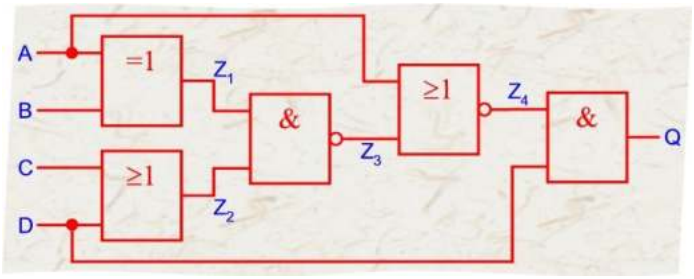
Q_3 I_1 I_0

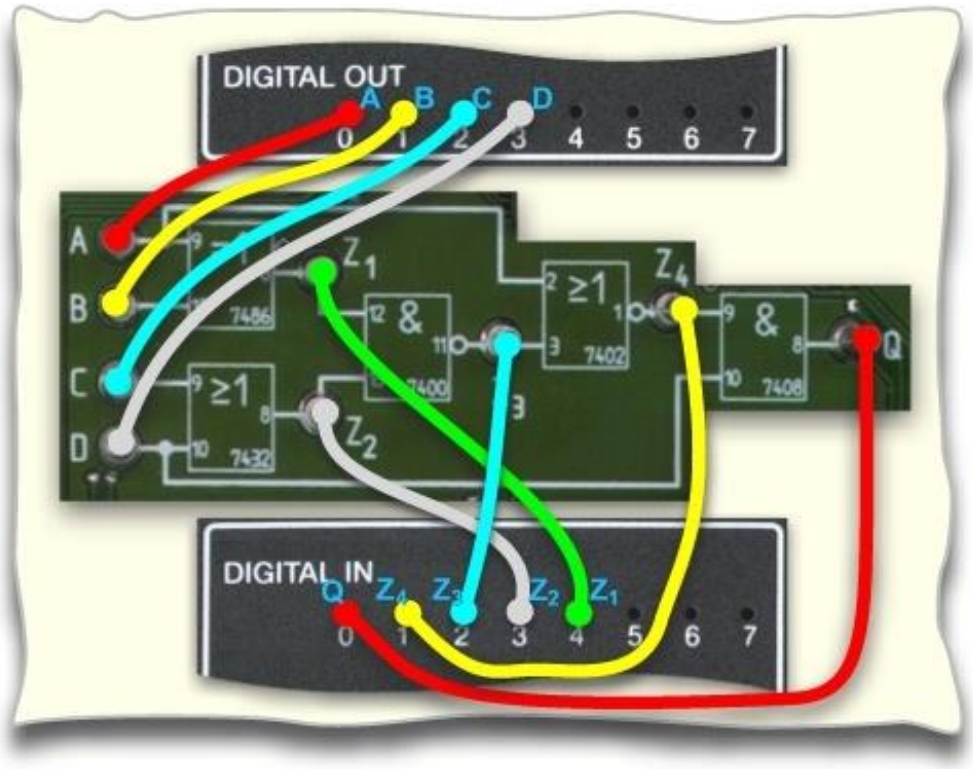
D	Z_4	Q
0	0	
0	1	
1	0	
1	1	

What is the fault?

What might cause such a fault to occur?

Fault 6





Exercise

Identify the fault in the circuit. Describe the fault and suggest a remedy:

6.1 Construct the initial circuit.

6.2 Observe the response of the output to various logic levels at the input and enter them into *Table 6*. Use the "Extended input and output" VI.

Tip:
You can alter the bit status quickly and easily by using the up and down buttons on the VI.

6.3 From *Table 5*, transfer the appropriate values to the 5 smaller tables. This should enable you to recognize the functioning of the individual gates. If certain combinations of bits do not occur, enter a "?".

Table 6

	Q_3	Q_2	Q_1	Q_0	I_4	I_3	I_2	I_1	I_0
	D	**C**	**B**	**A**	**Z_1**	**Z_2**	**Z_3**	**Z_4**	**Q**
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					

Q_1	Q_0	I_4
B	**A**	**Z_1**
0	0	
0	1	
1	0	
1	1	

Q_3	Q_2	I_3
D	**C**	**Z_2**
0	0	
0	1	
1	0	
1	1	

I_4	I_3	I_2
Z_1	**Z_2**	**Z_3**
0	0	
0	1	
1	0	
1	1	

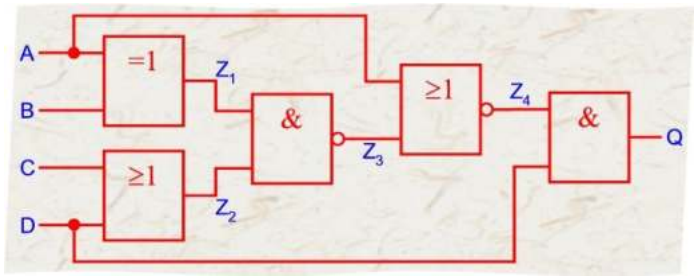
Q_0	I_2	I_1
A	**Z_3**	**Z_4**
0	0	
0	1	
1	0	
1	1	

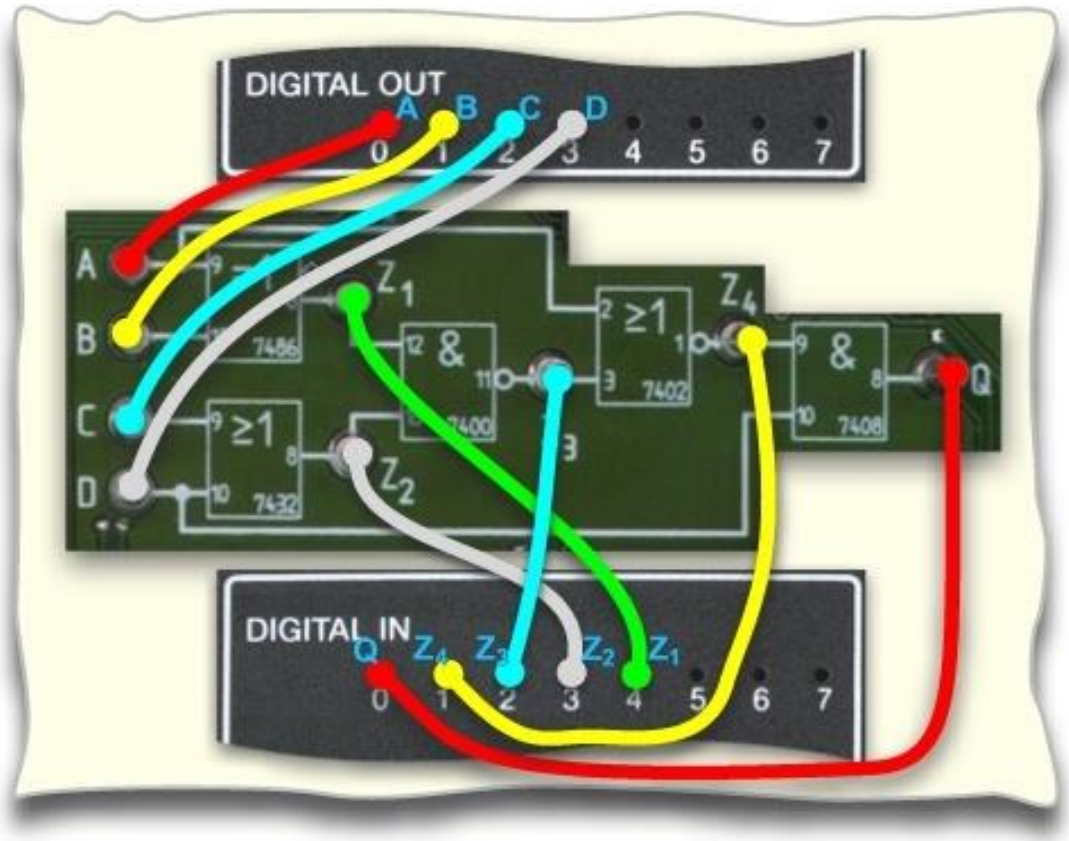
Q_3	I_1	I_0
D	**Z_4**	**Q**
0	0	
0	1	
1	0	
1	1	

What is the fault?

What might cause such a fault to occur?

Fault 7





Exercise

Identify the fault in the circuit. Describe the fault and suggest a remedy:

7.1 Construct the initial circuit.

7.2 Observe the response of the output to various logic levels at the input and enter them into *Table 7*. Use the "Extended input and output" VI.

Tip:
You can alter the bit status quickly and easily by using the up and down buttons on the VI.

7.3 From *Table 7*, transfer the appropriate values to the 5 smaller tables. This should enable you to recognize the functioning of the individual gates. If certain combinations of bits do not occur, enter a "?".

Table 7

	Q_3	Q_2	Q_1	Q_0	I_4	I_3	I_2	I_1	I_0
	D	**C**	**B**	**A**	**Z_1**	**Z_2**	**Z_3**	**Z_4**	**Q**
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					

Q_1	Q_0	I_4
B	**A**	**Z_1**
0	0	
0	1	
1	0	
1	1	

Q_3	Q_2	I_3
D	**C**	**Z_2**
0	0	
0	1	
1	0	
1	1	

I_4	I_3	I_2
Z_1	**Z_2**	**Z_3**
0	0	
0	1	
1	0	
1	1	

Q_0	I_2	I_1
A	**Z_3**	**Z_4**
0	0	
0	1	
1	0	
1	1	

Q_3	I_1	I_0
D	**Z_4**	**Q**
0	0	
0	1	
1	0	
1	1	

What is the fault?

What might cause such a fault to occur?

REVIEW **QUESTIONS**

Q.1: Fill in space:

1.1……is the gate which gives the complement of its input.

1.2. The output of ....gate is only Q when all its inputs are zeroes.

1.3. The output of ....gate is only 1 when all its inputs are ones.

1.4. The output of ..... gate is only Q when all its inputs are ones.

1.5. The output of ....gate is only 1 when all its inputs are zeroes.

1.6. The output of .... gate is only 1 when its inputs are different.

Q.2: Circle the correct answer:

2.1. A NOT gate is used to:
(a) buffer its input. (b) amplify its input.
(c) complement its input. (d) phase delay its input.

2.2. The output of an OR gate is HIGH:
(a) all of the time (b) when any input is LOW.
(c) when any input is HIGH. (d) when all inputs are LOW

2.3. The output of an AND gate is HIGH:
(a) all of the time. (b) when any input is LOW.
(c) when any input is HIGH. (d) when all inputs are HIGH.

2.4. A 2-input OR gate has its output logic state level 0. The inputs A & B are at logic states:
(a) A = 0, B = 0 (b) A = 0, B = 1
(c) A = 1, B = 0 (d) A = 1, B = 1

2.5. A 2-input AND gate has its output state level 1. The inputs A & B are at logic states:
(a) A = 0, B = 0 (b) A = 0, B = 1
(c) A = 1, B = (d) A = 1, B = 1

2.6. The output NOR gate is 1 (HIGH) when:
(a) When one of the inputs at logic level 1.
(b) When all inputs are at logic level I.
(c) When all inputs are at logic level 0
(d) When one of the inputs at logic level 0

2.7. The output of a NAND gate is 0 (LOW) when:
(a) When one of the inputs at logic level 1.
(b) When all inputs are at logic level l.
(c) When all inputs are at logic level 0.
(d) When one of the inputs at logic level 0.

2.8. A NOR gate with inverted output performs as the logic function of:

(a) AND gate.
(b) OR gate.
(c) NAND gate.
(d) NOT (inverter) gate.

2.9. A NAND gate can be converted to an inverter by:

(a) connecting inverters to the inputs.
(b) connecting one inverter to one of the inputs.
(c) connecting the inputs together.
(d) connecting one inverter to the output.

2.10. The output of XOR gate is 0 (LOW) when:

(a) one of the inputs at logic level 1.
(b) all inputs are at logic level 1.
(c) all inputs are at logic level 0.
(d) one of the inputs at logic level 0.
(e) both (b) and (c).

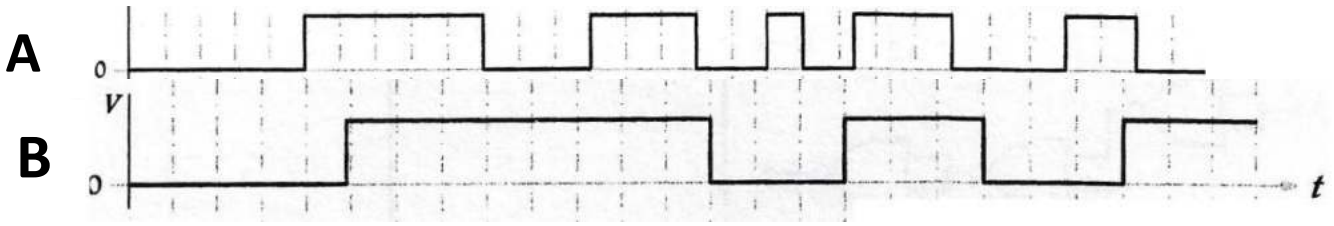
2.11. An Exclusive-OR gate provides a high output indication:

(a) for condition of equality. (b) all of the time.
(c) for condition of inequality. (d) none of the above

2.12. An Exclusive-NOR gate provides a low output indication:

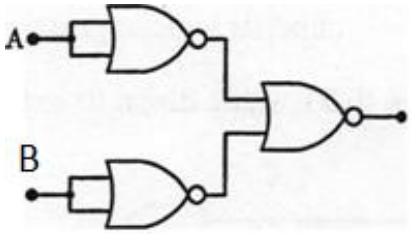
(a) for condition of equality. (b) all of the time.
(c) for condition of inequality. (d) none of the above.

Q.3: For the two waveforms A & B shown in Figure below, if they applied to the inputs of AND, NAND, OR; NOR and XOR gate; draw the output waveform from each gate.



(d) none of the above.

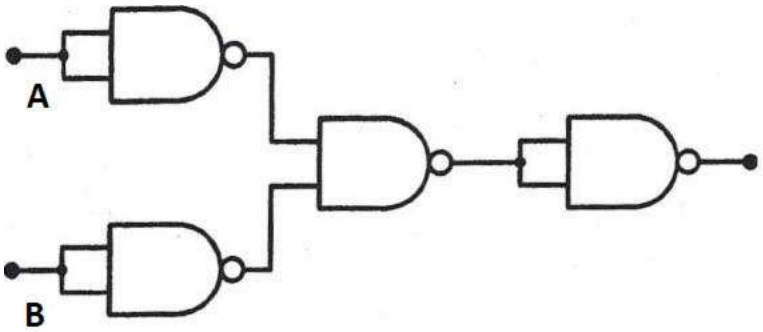
Q.4: (a) For the circuit in Figure below, write down its logic function equation.



F= ....

(b)develop truth table for the circuit to find out what logic gate does it perform ?

Q.5: (a) For the circuit in Figure below, write down its logic function equation.



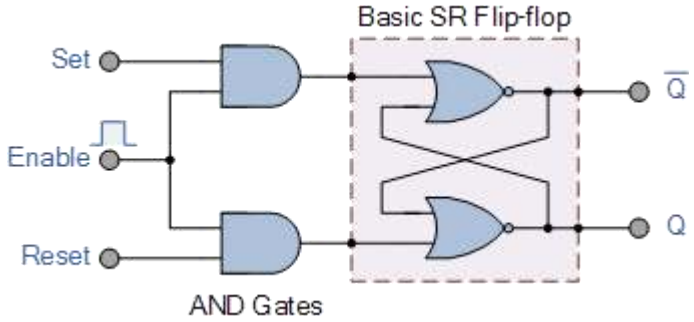
F=…….

(b) develop truth table for the circuit to find out what logic gate does it perform ?

Chapter 4
JK-flip flop

Conversion of Flip-flops

Flip-flops are the basic building blocks of Sequential Circuits which can be converted from one form to another capable of storing a single bit of data



The basic S-R NAND flip-flop circuit has many advantages and uses in sequential logic circuits but it suffers from two basic switching problems.

1. the Set = 0 and Reset = 0 condition (S = R = 0) must always be avoided
2. if Set or Reset change state while the enable (EN) input is high the correct latching action may not occur

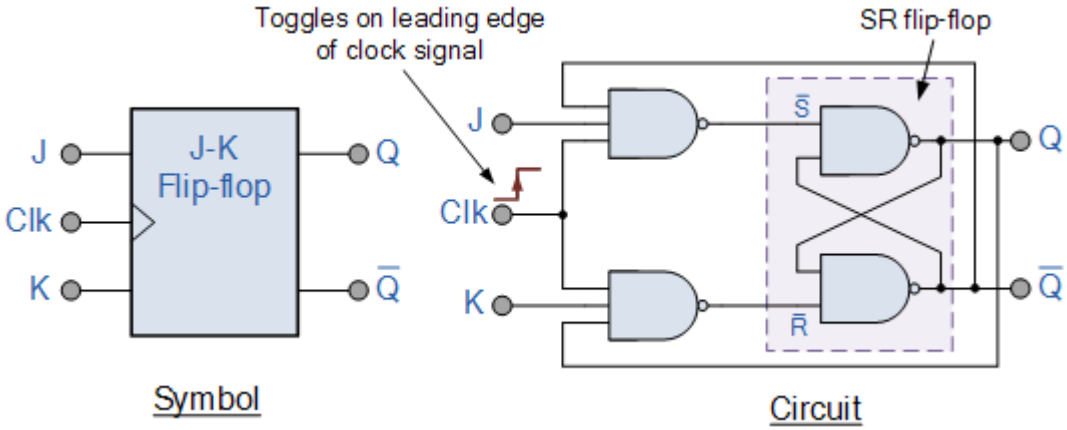
Then to overcome these two fundamental design problems with the SR flip-flop design, the **JK flip Flop** was developed.

This simple **JK flip Flop** is the most widely used of all the flip-flop designs and is considered to be a universal flip-flop circuit. The two inputs labelled “J” and “K” are not shortened abbreviated letters of other words, such as “S” for Set and “R” for Reset, but are themselves autonomous letters chosen by its inventor Jack Kilby to distinguish the flip-flop design from other types.

The sequential operation of the JK flip flop is the same as for the previous SR flip-flop with the same “Set” and “Reset” inputs. The difference this time is that the “JK flip flop” has no invalid or forbidden input states of the SR Latch even when S and R are both at logic “1”.

The **JK flip flop** is basically a gated SR flip-flop with the addition of a clock input circuitry that prevents the illegal or invalid output condition that can occur when both inputs S and R are equal to logic level “1”. Due to this additional clocked input, a JK flip-flop has four possible input combinations, “logic 1”, “logic 0”, “no change” and “toggle”. The symbol for a JK flip flop is similar to that of an *SR Bistable Latch* as seen in the previous tutorial except for the addition of a clock input.

The Basic JK Flip-flop



Both the S and the R inputs of the previous SR bistable have now been replaced by two inputs called the J and K inputs, respectively after its inventor ***Jack Kilby***. Then this equates to: J = S and K = R.

The two 2-input AND gates of the gated SR bistable have now been replaced by two 3-input NAND gates with the third input of each gate connected to the outputs at Q and Q. This cross coupling of the SR flip-flop allows the previously invalid condition of S = “1” and R = “1” state to be used to produce a “toggle action” as the two inputs are now interlocked.

If the circuit is now “SET” the J input is inhibited by the “0” status of Q through the lower NAND gate. If the circuit is “RESET” the K input is inhibited by the “0” status of Q through the upper NAND gate. As Q and Q are always different, we can use them to control the input. When both inputs J and K are equal to logic “1”, the JK flip flop toggles as shown in the following truth table.

The Truth Table for the JK Function

	Clock	Input		Output		Description
	Clk	J	K	Q	Q	
same as for the SR Latch	X	0	0	1	0	Memory no change
	X	0	0	0	1	
	¯↓_	0	1	1	0	Reset Q » 0
	X	0	1	0	1	
	¯↓_	1	0	0	1	Set Q » 1
	X	1	0	1	0	
toggle action	¯↓_	1	1	0	1	Toggle
	¯↓_	1	1	1	0	

Then the JK flip-flop is basically an SR flip flop with feedback which enables only one of its two input terminals, either SET or RESET to be active at any one time thereby eliminating the invalid condition seen previously in the SR flip flop circuit.

Also when both the J and the K inputs are at logic level “1” at the same time, and the clock input is pulsed “HIGH”, the circuit will “toggle” from its SET state to a RESET state, or visa-versa. This results in the JK flip flop acting more like a T-type toggle flip-flop when both terminals are “HIGH”.

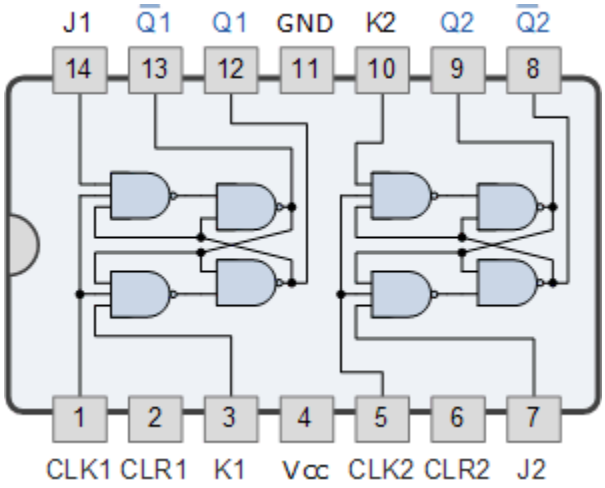
Although this circuit is an improvement on the clocked SR flip-flop it still suffers from timing problems called “race” if the output Q changes state before the timing pulse of the clock input has time to go “OFF”. To avoid this the timing pulse period (T) must be kept as short as possible (high frequency). As this is sometimes not possible with modern TTL IC’s the much-improved **Master-Slave JK Flip-flop** was developed.

Master-Slave JK Flip-flop

The master-slave flip-flop eliminates all the timing problems by using two SR flip-flops connected in a series configuration. One flip-flop act as the “Master” circuit, which triggers on the leading edge of the clock pulse while the other acts as the “Slave” circuit, which triggers on the falling edge of the clock pulse. This results in the two sections; the master section and the slave section being enabled during opposite half-cycles of the clock signal.

The TTL 74LS73 is a Dual JK flip-flop IC, which contains two individual JK type bi-stable’s within a single chip enabling single or master-slave toggle flip-flops to be made. Other JK flip flop IC’s include the 74LS107 Dual JK flip-flop with clear, the 74LS109 Dual positive-edge triggered JK flip flop and the 74LS112 Dual negative-edge triggered flip-flop with both preset and clear inputs.

Dual JK Flip-flop 74LS73



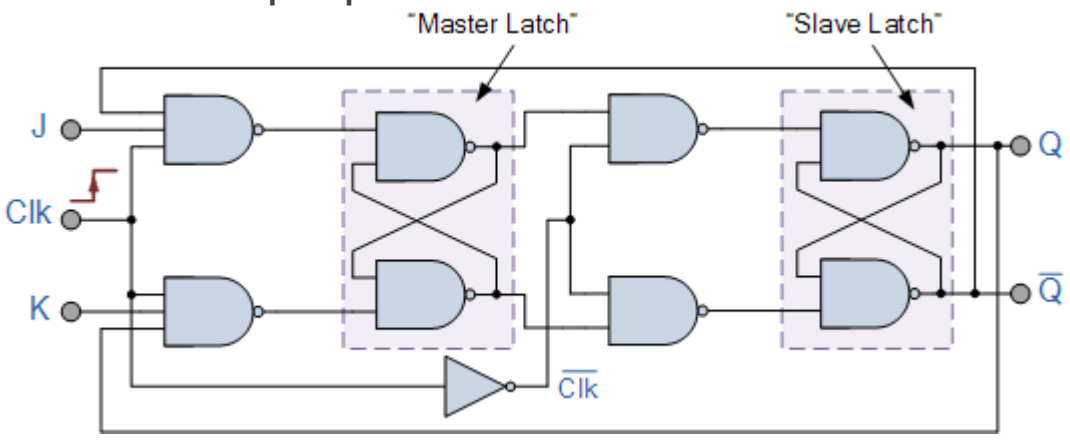
Other Popular JK Flip-flop ICs

Device Number	Subfamily	Device Description
74LS73	LS TTL	Dual JK-type Flip Flops with Clear
74LS76	LS TTL	Dual JK-type Flip Flops with Preset and Clear
74LS107	LS TTL	Dual JK-type Flip Flops with Clear
4027B	Standard CMOS	Dual JK-type Flip Flop

The Master-Slave JK Flip-flop

The **Master-Slave Flip-Flop** is basically two gated SR flip-flops connected together in a series configuration with the slave having an inverted clock pulse. The outputs from Q and Q from the “Slave” flip-flop are fed back to the inputs of the “Master” with the outputs of the “Master” flip flop being connected to the two inputs of the “Slave” flip flop. This feedback configuration from the slave’s output to the master’s input gives the characteristic toggle of the JK flip flop as shown below.

The Master-Slave JK Flip Flop



The input signals J and K are connected to the gated “master” SR flip flop which “locks” the input condition while the clock (Clk) input is “HIGH” at logic level “1”. As the clock input of the “slave” flip flop is the inverse (complement) of the “master” clock input, the “slave” SR flip flop does not toggle. The outputs from the “master” flip flop are only “seen” by the gated “slave” flip flop when the clock input goes “LOW” to logic level “0”.

When the clock is “LOW”, the outputs from the “master” flip flop are latched and any additional changes to its inputs are ignored. The gated “slave” flip flop now responds to the state of its inputs passed over by the “master” section.

Then on the “Low-to-High” transition of the clock pulse the inputs of the “master” flip flop are fed through to the gated inputs of the “slave” flip flop and on the “High-to-Low” transition the same inputs are reflected on the output of the “slave” making this type of flip flop edge or pulse-triggered.

Then, the circuit accepts input data when the clock signal is “HIGH”, and passes the data to the output on the falling-edge of the clock signal. In other words, the **Master-Slave JK Flip flop** is a “Synchronous” device as it only passes data with the timing of the clock signal.

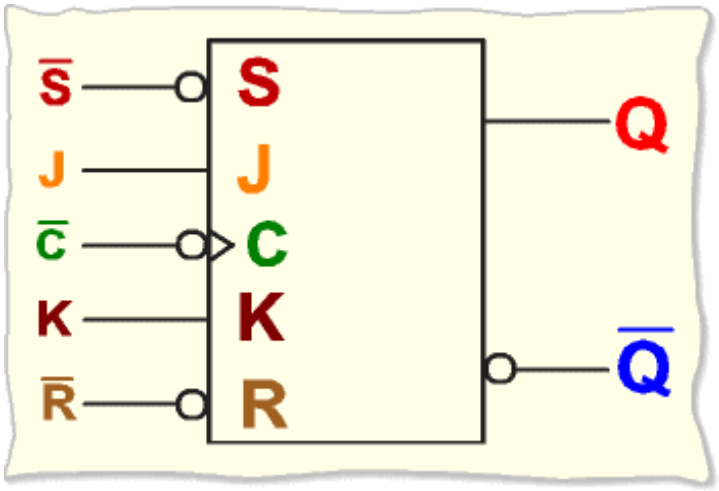
In the next tutorial about **Sequential Logic Circuits**, we will look at *Multi-vibrators* that are used as waveform generators to produce the clock signals to switch sequential circuits.

FLIP-FLOPS in the lab

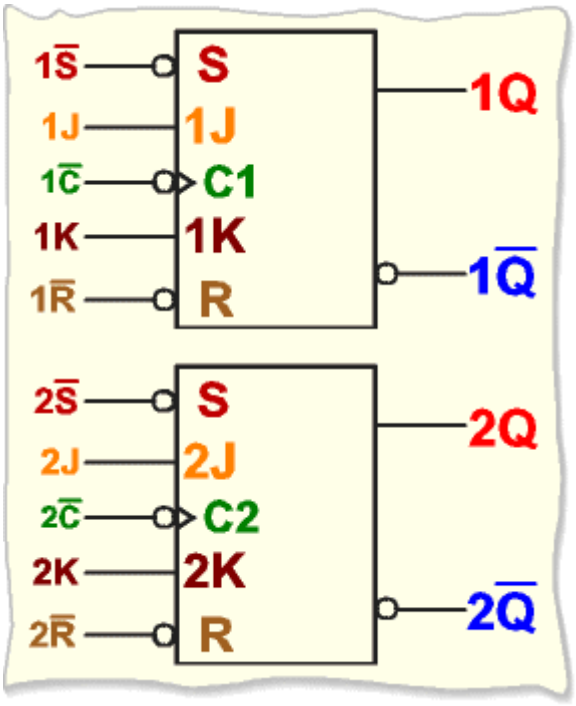
The **JK flip-flop** used here is a single-edge triggered flip-flop.
(There are also double-edge-triggered flip-flops - the so-called JK master-slave flip-flops [JKMS flip-flop])

The designations "**J**" for the set and "**K**" for the reset inputs have been chosen arbitrarily (consequently they do' reflect their actual function as they do in the RS flip-flop). From time to time **J** is said to mean "Jump" and **K** "Kill" which more closely resembles the idea of Set and Reset functions.

Besides having the set (**J**), reset (**K**) and clock input (**C**), the **74 HC 112** used here has two additional inputs. These have one **static** set (**S**) and one static reset input (**R**).



These static (asynchronous) inputs **S** and **R**, as well as the clock input **C** are inverted inputs (active LOW), i.e. they are activated by a LOW-level signal at the input. The clock input responds to the negative or falling edge of a clock pulse (change of the clock signal from HIGH to LOW).



If, as is the case with the 74HC112, we are dealing with a multiple (here 2-fold) flip-flop IC, numbers are also added to the designations to ensure pins are identified uniquely.

Static set and reset inputs:
Irrespective of the inputs **J**, **K** and **C** the following truth table applies:

Function	INPUTS		OUTPUTS	
	S	R	Q	Q
Asynchronous set	L	H	H	L
Asynchronous reset	H	L	L	H
Indeterminate	L	L	H	L

When S=LOW the flip-flop is set and when R=LOW it is reset.

The state where S=LOW and R=LOW simultaneously is to be avoided, because in this state, both inputs simultaneously switch from LOW to HIGH, and thus the output state of Q and Q cannot be predicted.

The response of the IC in this "disallowed" state is manufacturer-specific and can vary.
With ICs manufactured by Texas Instruments, Hitachi or STMicroelectronics, for example, if S=LOW and R=LOW the output state is assumed to be Q=HIGH and Q=HIGH.

Dynamic setting and resetting:
For the case where the static set and reset inputs **S** and **R** are set to HIGH (**S** and **R** thus being LOW), the following truth table applies:

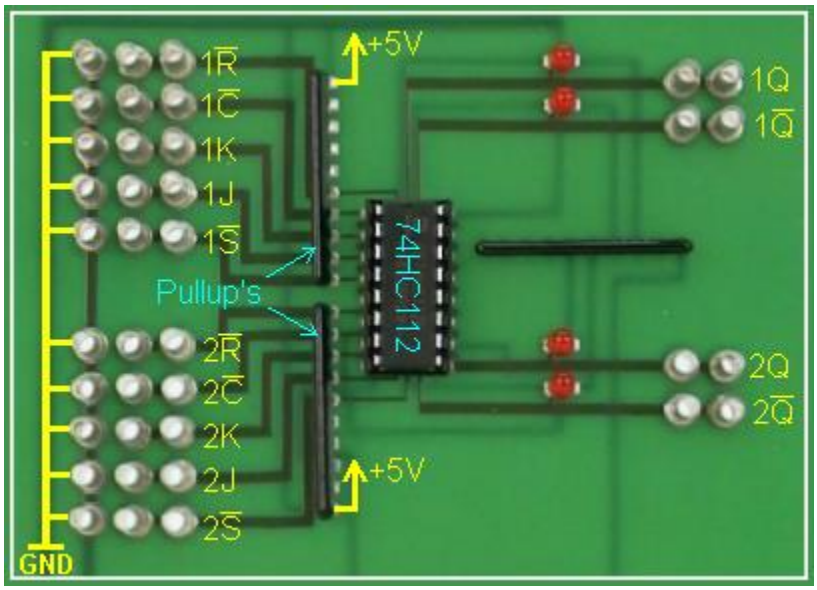
Function	INPUTS			OUTPUTS	
	C	J	K	Q	Q
Invert (Toggle)	↓	H	H	q	q
Reset (load "0")	↓	L	H	L	H
Set (load "1")	↓	H	L	H	L
Change (save)	↓	L	L	q	q

↓ : Falling edge, clock pulse C goes from HIGH to LOW

q : Value present at output Q before current clock pulse C

q : Value at output Q before the current clock pulse C

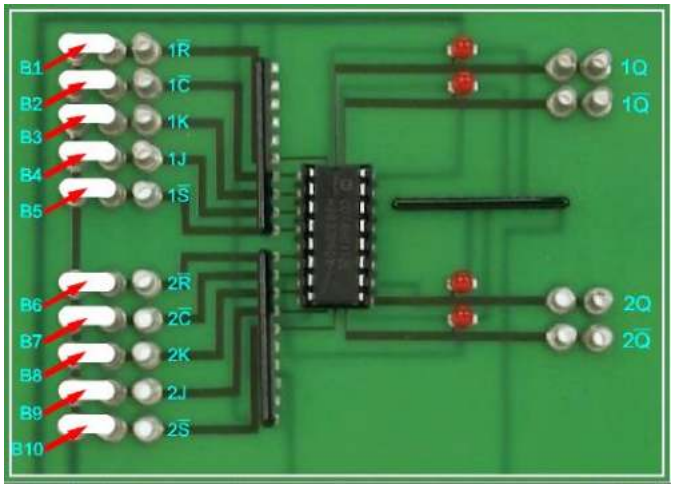
Circuit



The figure shows the arrangement of contacts for the two flip-flops. The inputs are provided with pull-up resistors to pull the inputs up to 5V (HIGH). The inputs can be set to 0V (LOW) by inserting jumpers **B1** to **B10** (see below)

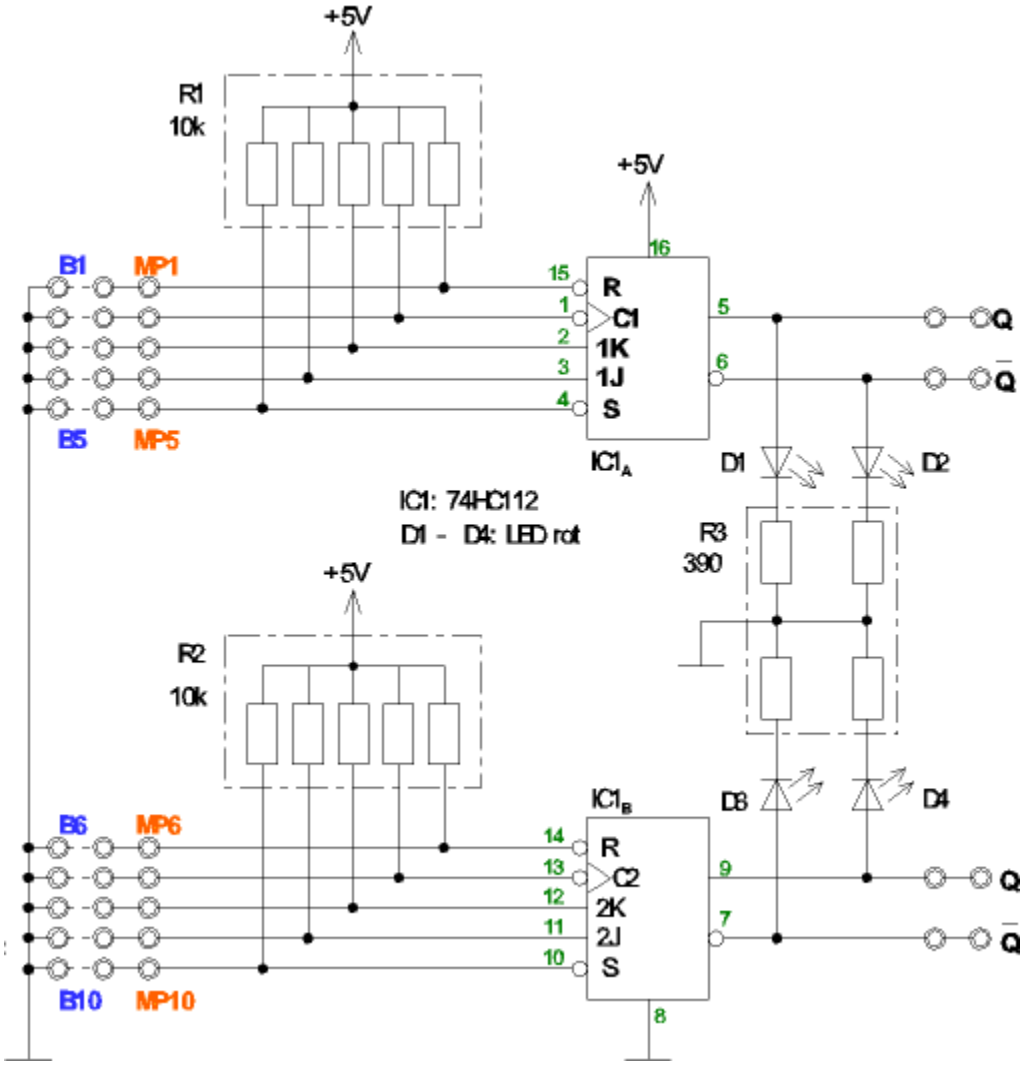
Without jumpers
R and **S** are inactive, **J** and **K** are active and **C** is HIGH.

With jumpers
it follows that **R** and **S** are active, **J** and **K** are inactive and **C** is LOW.



If you click on the figure, it will open a new window, to which you can easily switch so that you always have ready access to the contact designations.

Here is the schematic diagram of the flip-flop circuit implemented on the card.



If you click on the figure, it will open a new window, to which you can easily switch so that you always have a quick overview of how the circuit operates.

Static response of a JK flip-flop:

Procedure

1. Insert the JK Flip-Flop experiment card into the Experimenter **without** any connection plugs. The board is thus in its initial state.
   Now connect the sockets on the experiment card to the sockets of the Uni*Tr@in*-I Interface as specified in the list of connections to the right.

List of connections

From	*To*
Digital In 6	Terminal 1R
Digital In 5	Terminal 1C
Digital In 4	Terminal 1K
Digital In 3	Terminal 1J
Digital In 2	Terminal 1S
Dig-In 1	Terminal 1Q
Dig-In 0	Terminal 1Q

2. Open the virtual instrument ***Digital Inputs*** and transfer the values measured in the following exercises into the Table. (Enter an **H** when a HIGH signal level is detected and an **L** for LOW levels). Also, be sure to enter whether the LEDs D1 and D2 light up or not.

1R	1C	1K	1J	1S	1Q	1Q

3. **Initial state:**
   Enter the values into column **a)**.
4. **Static set:**
   Insert jumper **B5**. Enter the values into column **b)** .
   Now remove jumper **B5** and observe whether any change can be observed at the outputs.
5. **Static reset:**
   Insert jumper **B1**. Enter the values into column **c)**
   Now remove jumper **B1** and observe whether any change can be observed at the outputs.
6. **Simultaneous static activation of set & reset:**
   Now insert jumpers **B1**and **B5**. Enter the values in column **d)**.

S
E
T
↓

R
E
S

7. Fill in the gaps with the words: ***active, high, inactive, low or not***

E
T

↓

Meas. point	a) without jumpers	b) with B5	c) with B1	d) with B1&B5
1R				
1C	--------	------- -	------- -	--------
1K				
1J				
1S				
Q				
Q				
Diode 1				
Diode 2				

L (Logic 0 LOW)
H (Logic 1 HIGH)

In the **initial state**:
Q = [ ] and Q = [ ]

Without jumpers:

All inputs are at [ ] level.
The internal set (S) and reset inputs (R) are set to [ ] and are [ ].

With B5:

If **B5** is bridged, **S** is [ ]
so the set function is [ ].
Thus, the output **Q** is [ ]. That means that **Q** is [ ]

With B1:

If **B1** is bridged, **R** is [ ] so that the static reset function is [ ]. Thus, the output **Q** is [ ]. Consequently, **Q** is [ ]

With B1 and B5:

LEDs:

LED D1 lights up when Q is [ ].
D2 does [ ] light up in this case.

The LED D2 lights up if Q is [ ].
D1 does [ ] light up in this case.

Dynamic response of JK flip-flops

Investigating single pulses:

In the following experiments, during the clock pulse keep the static inputs **R** and **S**on **High** (jumpers **B1** and **B5** **not** connected) (the functions they represent are therefore inactive).
The static inputs are set as required in advance of the clock pulse to produce a particular initial state (set or reset the flip-flop).

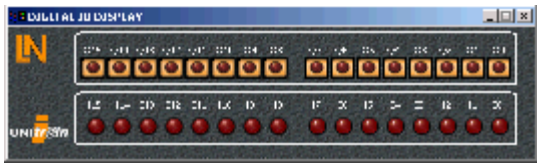
Procedure

1. **J = High ; K = Low**
   Insert jumper **B3** only.
   Connect the terminal sockets of the experiment card with the sockets of the Uni*Tr@in*-I Interface in accordance with the list of connections.

Connection lists

From	*To*
Digital In 6	Terminal 1R
Digital In 5	Terminal 1C
Digital In 4	Terminal 1K
Digital In 3	Terminal 1J
Digital In 2	Terminal 1S
Dig-In 1	Terminal 1Q
Digital In 0	Terminal 1Q
Digital Out 0	Terminal 1C

2. Now go to the menu *Instruments* and open the virtual instrument
   - Digital inputs and outputs



Settings

Digital outputs	Q_0 is activated by individual clock pulses

3. If the flip-flop is to be set (Q=H, LED D1 lights up) then briefly insert the jumper **B1** to reset the flip-flop.
4. Enter the states of the inputs and outputs into the Table in column **1a)**.
5. Using Q_0 generate a single clock pulse. Now enter the resulting state in column **1b)**. Generate an another clock signal and note down the resulting state in column **1c)**.
6. **J = Low ; K = High**
   Remove the connection plug **B3** and only plug in connection plug **B4**.
7. The flip-flop should be set (Q=H, LED D1 lights up). If it is not to be set, briefly insert connection plug **B5** to reset the flip-flop.
8. Enter the states of the inputs and outputs into the Table in column **2a)**.
9. Using Q_0 generate a single clock pulse. Now enter the resulting state in column **2b)**. Generate another clock signal and note down the state in column **2c)**.

v-T1 : before first clock pulse
n-T1 : after first clock pulse (= before second clock pulse)
n-T2 : after second clock pulse

(1): reset the flip-flop before applying the clock pulse
(2): set the flip-flop before applying the clock pulse

	(1) ↓			(2) ↓		
	1a)	1b)	1c)	2a)	2b)	2c)
	Pulse 1 - Pulse 2			Pulse 1 - Pulse 2		
	v-T_1	n-T_1	n-T_2	v-T_1	n-T_1	n-T_2
Meas. point	with B3			with B4		
1R	☐	☐	☐	☐	☐	☐
1C	--------	--------	--------	--------	--------	--------
1K	☐	☐	☐	☐	☐	☐
1J	☐	☐	☐	☐	☐	☐
1S	☐	☐	☐	☐	☐	☐
Q	☐	☐	☐	☐	☐	☐
Q	☐	☐	☐	☐	☐	☐
Diode 1	○	○	○	○	○	○
Diode 2	○	○	○	○	○	○

10. **J = Low ; K = Low**
    Insert jumpers **B3 & B4** only.
11. If the flip-flop is set, briefly insert jumper **B1** to reset it. Enter the input and output states into the Table in column **1a)**.
12. Using Q_0 generate a single clock pulse. Now enter the state in column **3b)**.
13. Briefly insert jumper **B5** to set the flip-flop. Enter the input and output states into the Table in column **3c)**.
14. Generate a clock pulse and note down the state in column **3d)**.
15. **J = High ; K = High**
    Remove all jumpers.
16. Briefly insert jumper **B1** to reset the flip-flop. Enter the input and output states into the Table in column **4a)**.
17. Using Q_0 generate a single clock pulse. Enter the resulting state in column **4b)**. Generate two additional clock pulses one after another and note down the resulting status after each pulse in column **4c)** and **4d)**.

(1): reset the flip-flop before applying the clock pulse
(2): set the flip-flop before applying the clock pulse

	(1) ↓ 3a)	3b)	(2) ↓ 3c)	3d)	(1) ↓ 4a)	4b)	4c)	4d)
	Pulse 1		Pulse 2		Pulse 1 - Pulse 2 - Pulse 3			
	v-T_1	n-T_1	v-T_2	n-T_2	v-T_1	n-T_1	n-T_2	n-T_3
Meas. point	With B3 & B4				without jumpers			
1R	☐	☐	☐	☐	☐	☐	☐	☐
1C	------	------	------	------	------	------	------	------
1K	☐	☐	☐	☐	☐	☐	☐	☐
1J	☐	☐	☐	☐	☐	☐	☐	☐
1S	☐	☐	☐	☐	☐	☐	☐	☐
Q	☐	☐	☐	☐	☐	☐	☐	☐
Q	☐	☐	☐	☐	☐	☐	☐	☐
Diode 1	○	○	○	○	○	○	○	○
Diode 2	○	○	○	○	○	○	○	○

References :

1-digital techniques , Eng.M.Elshishtawi(1997), fourth edition Nov-2014

2-**L@Bsoft course "*Digital Technology*",**LUCAS-NÜLLE GmbH

3- Electronics Tutorials website-courses online