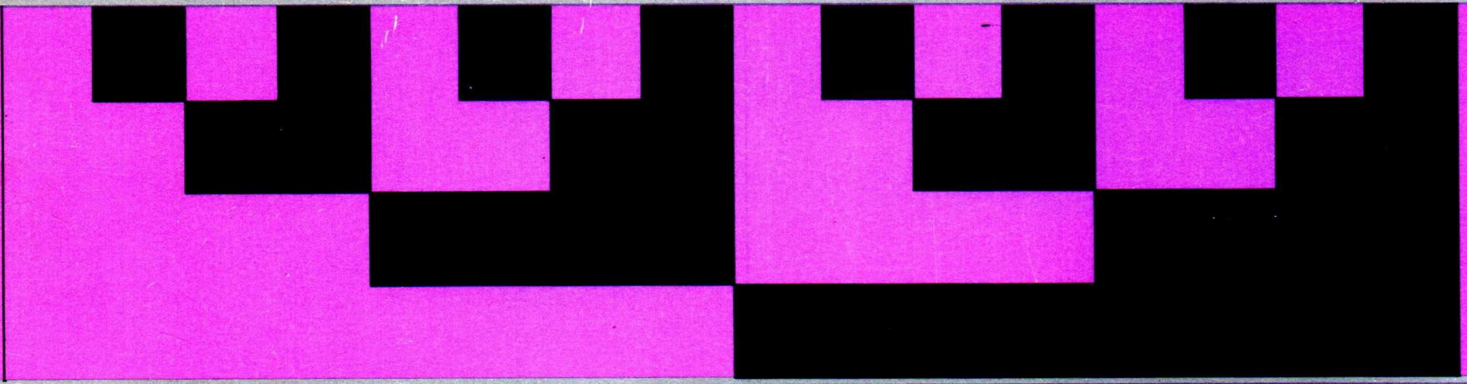
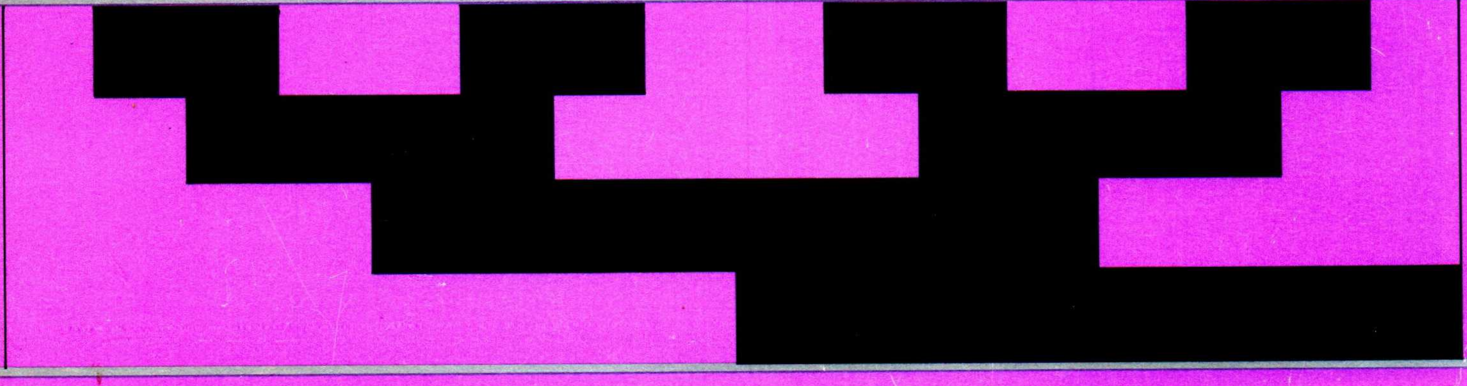
PHILIPS

PHILIPS

digital instrument course

Part 1 BASIC BINARY THEORY AND LOGIC CIRCUITS





Published by

N.V. Philips' Gloeilampenfabrieken
Test and Measuring Department
Eindhoven, The Netherlands

The revised and extended first part of the course-already serialized in Philips Test and Measuring Notes

Editor: T. Sudar

Assistant Editor: R. H. Bathgate

Price $4,—, £ 2,40, Hfl 10,—

Fifth, revised edition

The front cover patterns represent the BCD 1, 2, 4, 8 code (above) and the Gray code (below) for numbers from 1 to 16.

DIGITAL INSTRUMENT COURSE

Part 1. Basic binary theory and logic circuits

by A. J. Bouwens

Comparators
Display interface
General industrial
Telecommunications circuits
Voltage regulators
signetics
CH00

Contents list

Part 2 Digital counters and timers.
The second part of the digital instrument course deals with digital frequency counters and timers.
The chapter headings are:
Basic counter circuitry
Modes of operation
Plug-in units and special functions
Accuracy

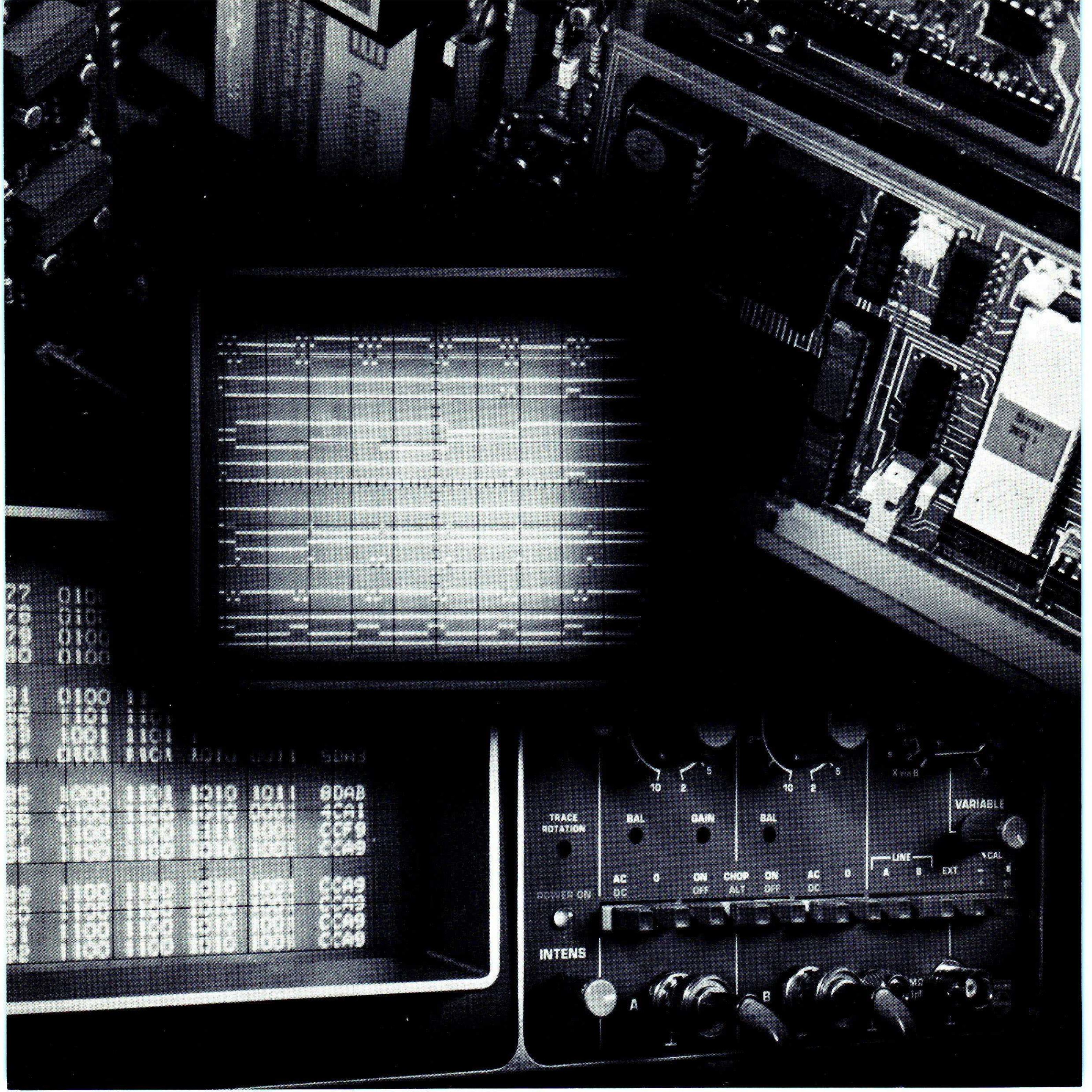
Part 3 Digital voltmeters and multimeters.
The third part of the digital instrument course deals with the principles of operation of DVM's, and highlights two application-oriented techniques involving DVM's.
The chapter headings are:
The operational amplifier
The analog-to-digital converter (ADC)
Automation in voltmeters
Digital multimeter circuits
Accuracy of digital voltmeters
Guarding techniques
AC and RMS measurements

PHILIPS
digital instrument course
Part 3 DIGITAL VOLTMETERS AND MULTIMETERS
22.429
mV
V
kΩ
MΩ
mA
°C

Foreword

Nowadays, an increasing number of electronic measuring instruments are ''going digital'', digital circuitry, digital readout and digital remote control often being combined in the one instrument.
The newcomer to the field of digital techniques is faced with an abundance of articles, courses etc. devoted to the fundamentals of digital circuits. Much less, however, has been written about the use of such circuitry in measuring instruments.
Such information is important, because it enables the user of digital equipment to make the best possible use of the various facilities offered.
In order to satisfy this need, a four-day course in digital instrumentation was given a number of times for the instrument specialists of our European sales organisation. This course was such a success that we thought it might be worth while repeating it in condensed form.
The course is divided into the following parts:
- *Basic binary theory and logic circuits*
- *Digital timers and counters*
- *Digital voltmeters*

This book contains the first part of the course, which was originally serialized in Philips Test and Measuring Notes.


TRACE ROTATION
BAL
GAIN
BAL
VARIABLE
POWER ON
INTENS
AC DC
0
ON OFF
CHOP ALT
ON OFF
AC DC
0
LINE
A
B
EXT
CAL
A
B

Chapter 1

Number systems

Many different number systems are in daily use throughout the world. In electronic digital instruments, the digits are usually represented by different potential levels. If the decimal system were used, the circuits would have to be capable of differentiating accurately between ten levels representing the various digits. This is of course possible with careful circuit design, but the circuits would be rather complex and the chance of errors is quite high.
In digital equipment nowadays, however, the binary system is used practically exclusively, because of its most remarkable feature: simplicity – just two digits. Any number can be expressed in the binary system, which we shall now proceed to examine – after a brief review of various number systems.

Decimal system
The decimal system is probably the most commonly used number system, which is hardly surprising in view of the fact that man has ten fingers (''digits''). Everyone knows what to make of a number like 1972: the right-hand or least significant digit (LSD) stands for a number times 10 to the power of nought 10^0, the next represents a multiple of 10^1, and so on until the left-hand digit, or most significant digit (MSD) is reached, which in the above example stands for 10^3. This number can therefore be expressed as:

$$1972 = 1 \times 10^3 + 9 \times 10^2 + 7 \times 10^1 + 2 \times 10^0 =$$
$$1000 \quad + 900 \quad + 70 \quad + 2$$

Any integer (whole number) can be expressed as shown above, while fractional quantities can be expressed in much the same way using for example:

$$19.72 = 1 \times 10^1 + 9 \times 10^0 + 7 \times 10^{-1} + 2 \times 10^{-2}$$
$$= 10 \quad + 9 \quad + 7/10 \quad + 2/100$$

The advantage of the decimal system, compared e.g. with Roman numerals, is that because it has a positional notation based on powers of ten, arithmetical operations are considerably simplified. For example, a shift to the left multiplies by ten, while a shift to the right divides by ten:

1972 shift to left	19720
1972 shift to right	197,2

In general, a number to base r can be expressed as follows:

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

The coefficient ''a'' can range in value from 0 to r − 1. In the binary system (r = 2) we thus have two possible values of ''a'', viz 0 and 1; in the octal system (r = 8) we have eight possible values, from 0 to 7; in the decimal system (r = 10) ten from 0 to 9 and in the hexadecimal (r = 16) system sixteen from 0 to F; See table 1.1 (as there are only ten symbols in our notation for numerical values, we borrow the first six letters of the alphabeth for the extra six symbols needed in the hexadecimal system.

Decimal (Base 10)			Binary (Base 2)							Octal (Base 8)			Hexa-decimal (Base 16)	
10^2	10^1	10^0	2^6	2^5	2^4	2^3	2^2	2^1	2^0	8^2	8^1	8^0	16^1	16^0
	0	0				0	0	0	0		0	0		0
	0	1				0	0	0	1		0	1		1
	0	2				0	0	1	0		0	2		2
	0	3				0	0	1	1		0	3		3
	0	4				0	1	0	0		0	4		4
	0	5				0	1	0	1		0	5		5
	0	6				0	1	1	0		0	6		6
	0	7				0	1	1	1		0	7		7
	0	8				1	0	0	0		1	0		8
	0	9				1	0	0	1		1	1		9
	1	0				1	0	1	0		1	2		A
	1	1				1	0	1	1		1	3		B
	1	2				1	1	0	0		1	4		C
	1	3				1	1	0	1		1	5		D
	1	4				1	1	1	0		1	6		E
	1	5				1	1	1	1		1	7		F
1	0	0	1	1	0	0	1	0	0	1	4	4	6	4
1	2	7	1	1	1	1	1	1	1	1	7	7	7	F

Table 1.1 Numbers to different bases

Binary system

In order to simplify circuit design and improve reliability, the binary number system is generally employed in digital equipment.

As the binary system is based on powers of two there are only two digits namely 0 and 1. The instrument now only has to recognise two separate states which can be "no pulse" (0) and "pulse" (1), "low level" (0) or "high level" (1); "no contact" (0) or "contact closure" (1), "no current" (0) or "current" (1), and so on.

A positional notation is again used, as shown in Table 1.1, where the binary equivalents of a number of decimal integers are given.

As may be seen from the table, the decimal number 100 already requires as many as 7 binary digits.

The zeros and ones in the binary notation are generally called "bits"; this is a contraction of "binary digits".

It may be seen that the least significant digit (LSD) is still on the right while as we move from right to left each digit represents the next higher power of two. If successive negative indices are used, fractional quantities can also be represented; the digits to the right of the binary point now move in a descending scale, each being half the value of its predecessor.

Thus, the binary number.

$$1.1101 = 1\times 2^{0} + 1\times 2^{-1} + 1\times 2^{-2} + 0\times 2^{-3} + 1\times 2^{-4}$$

can be written in decimal notation:

$$1+\tfrac{1}{2}+\tfrac{1}{4}+0+\tfrac{1}{16} = 1.8125$$

It should be noted that decimal integers can be expressed precisely in binary form, but the conversion of a decimal fraction into a binary number will generally involve some approximation. The actual error will depend on the number of binary digits used, and is usually very small in practice. However, this does not mean that the binary system is less accurate than the decimal system; one simply needs more digits to represent a quantity with the required precision.

Octal system

A number system widely used in computer techniques is the octal system.

One of the particularly useful properties of the octal system is the simple conversion technique from binary to octal or vice versa.

The octal system comprises the following characters: 0, 1, 2, 3, 4, 5, 6, and 7. An octal value greater than 7 must be expressed by several digits, e.g.

$$765_{8}{}^{*} = 7\times 8^{2} + 6\times 8^{1} + 5\times 8^{0} = 7\times 64 + 6\times 8 + 5\times 1 = 501_{10}$$

As the base of the octal system is $8 = 2^{3}$, it is very easy to convert octal numbers into binary:

Binary 101011 can be split up in groups of bits 101/011 and directly transformed into octal: 5 3

using the following conversion table:

000 = 0	(octal)	100 = 4	(octal)
001 = 1	,,	101 = 5	,,
010 = 2	,,	110 = 6	,,
011 = 3	,,	111 = 7	,,

Check: 101 011 binary is

$1\times 2^{5} + 1\times 2^{3} + 1\times 2^{1} + 1\times 2^{0} = 43_{10}$

53 octal is

$5\times 8^{1} + 3\times 8^{0} = 43_{10}$

Hexadecimal system

Another number system – also used in computer techniques is the hexadecimal system. This number system has $r=16$ as its base; the characters used are given in Table 1.1. Just as with the octal system, it is easy to convert from the hexadecimal system to the binary and vice-versa since $2^{4}=16$ and hence each hexadecimal digit corresponds to 4 binary digits. Table 1.1 can be used as a conversion table; for example:

$225_{10} = 1110\,0001_{2} = E1_{16} = 341_{8}$

Conversion of number systems

Binary/octal to decimal

— Multiply the most significant (octal) digit by 8.

— Add to this product the value of the next significant digit and multiply by 8.

— Repeat this up to the least significant digit (LSD).

— Add the value of the LSD to the last product.

— The result of this is the required decimal number.

Example

```
Binary      111/011/101/001
Octal         7   3   5   1
Conversion  [(7 × 8 + 3) 8 + 5] 8 + 1
              7 × 8 + 3 = 59
                          8 ×
                     ________
                    472 + 5 = 477
                              8 ×
                          ________
                          3816 + 1 = 3817₁₀
```

* Note: The notation e.g. 765_8 is used here to mean the number 765 written in the octal notation.

This procedure is – of course also valid not only for integers but also for fractions, e.g.:

Binary 111/011 .101/001

Octal 7 / 3 /. 5 / 1

Conversion $(7 \times 8 + 3) + (1 \times 8^{-1} + 5)8^{-1} =$

59 . 641

Similarly a number to base 2 can be converted to do decimal equivalent by multiplying each digit by r (for the integer part) and dividing by r for the fractional part, and adding.

Decimal to octal/binary

1. Divide the decimal number by 8, and write down the remainder (r_1)
2. Divide the quotient of the preceding stage by 8 again and write down the remainder (r_2)
3. Repeat step 2 until the quotient is 0.
   The required number is then $r_n\ r_{n-1} \ldots\ldots r_2 r_1$

Example: Decimal 3817_{10}

Conversion 3817 : 8 = 477 : 8 = 59 : 8 = 7 : 8 = 0

```
              32        40       56     0
              —         —        —      —
               61        77       3 = r3  7 = r4
               56        72
               —         —
                57        5 = r2
                56
                —
                 1 = r1
```

Resulting in 7351 (octal) or 111/011/101/001 (binary).

Decimal fractions can be converted into binary/octal/hexadecimal fractions by a similar method, except that here one multiplies instead of dividing, and takes the integral part of the product instead of the remainder.

Example: 59.641_{10}

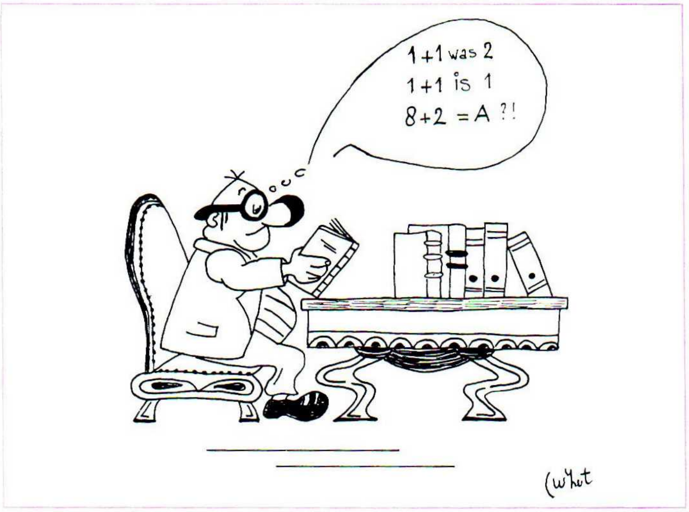
59 : 8 = 7 : 8 = 0

```
56    0
—     —
 3    7
```

$0.641 \times 8 = 5.128$
$0.128 \times 8 = 1.024$
$0.024 \times 8 = 0.192$
$0.192 \times 8 = 1.536$

Resulting in 73.5101 (octal); 3B.A41 (hexadecimal) or 111/011.101/001/000/001 (binary)

In general, the conversion of a decimal number to base r is done in the same way except that r is used instead of 8 (16).



Arithmetic processes in the binary number system

Arithmetic processes in the binary number system follow exactly the same basic rules as in the decimal number system.

Since the binary system is based on powers of two, a shift to the left multiplies by two and a shift to the right divides by two. Consider for example the binary number $1001 = 9_{10}$.

With a shift to the left, this becomes:

$$10010 = 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$$
$$= 16_{10} \quad + 0 \quad + 0 \quad + 2_{10} \quad + 0 \quad = 18_{10}$$

i.e. the number has been doubled

A shift to the right gives:

$$100.1 = 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1}$$
$$= 4 \quad + 0 \quad + 0 \quad + \tfrac{1}{2} \quad = 4\tfrac{1}{2}{}_{10}$$

i.e. the number has been halved.

The normal arithmetic processes can be carried out on numbers in any scale, but the rules in the binary system are extremely simple. For instance, there are only three rules for addition:

$0 + 0 = 0$
$1 + 0 = 1$
$1 + 1 =$ two $= 0$ and 1 to carry

and three rules for multiplication

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

Addition

$00111 = 7_{10}$
$00101 = 5_{10}$
——— + ———
$01100 = 12_{10}$(with carries to the second, third and fourth digits, reading from right to left)

Subtraction

In subtraction, it may be necessary to "borrow" from the preceding column. For example:

$01100 = 12_{10}$
$00111 = 7_{10}$
——— − ———
$00101 = 5_{10}$ (with "borrows" from the second, third and fourth columns)

Multiplication

Binary multiplication is very simple:

$111 = 7_{10}$
$101 = 5_{10}$
——— = ———

```
   111
  000 .
 111 . .
```

——— − ———
$100011 = 35_{10}$ (with carries to the first 4 digits)

Division

Binary division is the same as decimal division except for the intermediate multiplications and subtractions, which must (of course) be carried out in the binary system.

```
11/10101/111
   11
   ——
   100
    11
   ——
    011
     11
   ——
      0     (21 : 3 = 7)
```

Complements

In arithmic operations it is often more convenient to subtract by adding the complements of the number in question.
There are two types of complements in each base r system: the r's complement and the (r–1)'s complement.

The r's complement

The r's complement of a positive number N, the integral part of which has n digits, is defined as $r^n - N$.

Some examples:
The 10's complement of 1279_{10} is $10^4 - 1279 = 8721_{10}$
The 8's complement of 127_8 is $10_8^3 - 127_8 = 651_8$
The 2's complement of 101_2 is $10_2^3 - 101_2 = 011_2$

The r's complement of fractions is determined in the same way $(r^0 - N)$:
The r's complement of 0.1279_{10} is $1 - 0.1279 = 0.8721_{10}$
The r's complement of 0.127_8 is $1 - 0.127_8 = 0.651_8$
The r's complement of 0.101_2 is $1 - 0.101_2 = 0.011_2$

The (r − 1)'s complement

The definition of the (r − 1)'s complement is somewhat more complex. For a positive number N to base r with n digits in the integral part and m in the fractional part, the (r − 1)'s complement is defined as:

$r^n - r^{-m} - N$

Some examples will clarify this:
The 9's complement of 1279_{10} is $10^4 - 1 - 1279 = 8720_{10}$
(No fraction part, so $m = 0$ and $r^{-m} = 1$)
The 9's complement of 0.1279_{10} is
$1 - 10^{-4} - 0.1279 = 0.8720$
(No integer part, so $n = 0$ and $r^n = 1$)
The 9's complement of 12.79_{10} is $10^2 - 10^{-2} - 12.79 = 87.20$

In the octal system.
The 7's complement of 127_8 is $10_8^3 - 1 - 127_8 = 650_8$
The 7's complement of 0.127_8 is $1 - 10_8^{-3} - 0.127_8 = 0.65_8$
The 7's complement of 1.27_8 is $10_8 - 10_8^{-2} - 1.27_8 = 6.50_8$

and in the binary system:
The 1's complement of 101_2 is $10_2^3 - 1 - 101 = 010_2$
The 1's complement of 0.101_2 is $1_2 - 10_2^{-3} - 0.101_2 = 0.010_2$
The 1's complement of 1.01_2 is $10_2 - 10_2^{-2} - 1.01_2 = 0.10_2$

It follows from the above that the r's complement can be obtained from the (r − 1)'s complement by addition of 1 to the least significant digit. In the binary system, the 1's (r − 1) complement is easily obtained by simply inverting all bits. The 2's complement is then obtained by adding a 1; e.g.:
The 2's complement of 11011011 is obtained from the one's complement 00100100 by adding **one** to give 00100101.

Other examples

Binary	1's complement	2's complement
101	010	011
0.101	0.010	0.011
1.01	0.10	0.11

Octal	7's complement	8's complement
127	650	651
0.127	0.650	0.651
1.27	6.50	6.51

Subtraction with r's complement
The subtraction of two positive numbers $(M-N)_r$ can be realized as follows: First add the r's complement of the subtrahend N to the minuend M. When there is a carry, discard it; when there is no carry, take the r's complement of the number obtained and give it a negative sign.

Some examples will clarify this:

```
M = 1972                                    1972
N = 1279 10's complement 8721               8721+
                                           ------
                                         (1)0693
               carry is 1, answer is     +  693
                                           ======

M = 1279                                    1279
N = 1972 10's complement 8028               8028+
                                           ------
                                            9307
  no carry, so take complement; answer   -  693
                                           ======
```

And in binary notation

```
N = 1100                                    1100
N = 0111 2's complement                     1001+
                                           ------
                                         (1)0101
               carry is 1; answer is     +  101
                                           ======

M = 0111                                    0111
N = 1100 2's complement                     0100+
                                           ------
                                            1011
                 no carry; answer is     -  101
                                           ======
```

Subtraction with (r − 1)'s complement
This is carried out in the same way as described above except for the use of the "end-around carry" as explained below.
The subtraction of two positive numbers $(M-N)_r$ is in the following ways with (r − 1)'s complement.
First add the (r − 1)'s complement of the subtrahend N to the minuend M. If there is a carry add it to the least significant digit and if there is no carry take the (r − 1)'s complement of the number obtained and give it a negative sign.

Examples:

```
M = 19.72                                   19.72
N = 12.79   9's complement                  87.20+
                                           ------
                                           106.92
                          add carry:        └──→1+
                                           ------
                                             6.93
                                           ======

M = 12.79                                   12.79
N = 19.72   9's complement                  80.27+
                                           ------
                                            93.06
          no carry, take complement       - 6.93
                                           ======
```

and in binary notation:

```
M = 1100                                    1100
N = 0111   1's complement                   1000+
                                           ------
                                           10100
                          add carry:        └──→1+
                                           ------
                                             101
                                           ======

M = 0111                                    0111
N = 1100   1's complement                   0011+
                                           ------
                                            1010
          no carry, take complement       -  101
                                           ======
```

Both the 2's and 1's complements are used in digital techniques. Each has its advantages and disadvantages. The 1's complement is easy to obtain (by simply inverting all bits) but the arithmetics is somewhat more complicated (because of the end-around carry), while the opposite is true of the 2's complement. This is of course also true of the 9's and 10's complements in the decimal system.

Coding
A binary representation would be impossible as part of a human language (imagine saying one-zero-one-one-one instead of 23!) But the binary notation is the natural language of the two-state components of the electronic digital instruments. Its circuits represent the binary digits (bits) as **pulses** or **no pulses, high** or **low voltages**, etc. Special binary codes have therefore been devised to simplify the reading and handling of large numbers.
It is often convenient to code each digit of a decimal number separately, using 4 bits per decimal digit. The fact that 4 bits of binary notation represent the decimal numerals (0...9) makes it advantageous to combine the two systems. Such a code is usually known as the **binary coded decimal** system (BCD). This type of code is far easier for us to deal with because we think decimally.

The easiest method of choosing a BCD code is to use the first 10 binary numbers (0-9) and reject the remaining six (10-15, the illegitimate codes). This code is called the natural BCD (NBCD) or 8421 code. It has the advantage that normal binary techniques can be used.
The NBCD code is one of a group of codes that assign "weights" to each of the bits so each decimal digit will be equal to the sum of these weights. For example, decimal 974 in NBCD is:
1001 – 0111 – 0100
(9) – (7) – (4)

2421	4321	5221	5421	6321	7421
3321	4421	5311	6221	6421	8421
4311	5211	5321	6311	7321	

Table 1.2 Four-bit positive weighted BCD codes.

There are exactly 17 four-bit positive-weighted codes (tabulated in Table 1.2.), but apart from the 8421 code all weighted codes have at least one decimal digit that can be represented in more than one way. If we would like to encode e.g. 974 in the 4421 code one gets the following possibilities:
1101 – 1011 – 1000
or 1101 – 0111 – 0100
(9) – (7) – (4).
which gives 4 different ways of writing this number.

A disadvantage of the 8421 code is that it is not self-complementing. A self-complementing BCD code has the property that inversion of each bit ("1" exchanged for "0" and *vice versa*) makes each decimal digit its own 9's complement. E.g. the 2421 code is self-complementing. If we encode 974 in this code we get:
1111 – 0111 – 0100
(9) – (7) (4)
After inversion
0000 – 1000 – 1011
(0) (2) (5)

Other examples of self-complementing BCD codes are the 3321, 4311 and 5211 codes. Unfortunately they lead to rather cumbersome circuitry. There is however a simple alternative available in the excess – 3 (XS-3) code. When the sum of the weights of the bits of a BCD code exceeds the value of the decimal digit that it represents, the code is called excess weighted. In the XS-3 code the excess weight is three. If now a binary 3 (0011) is added to each decimal representation in the 8421 code, this new code becomes self-complementing. As can be seen from Table 1.3., this code uses the middle of the first 16 binary numbers.

	8421	2421	5421	5311	Excess-3	Gray	XS-3 Gray
0	0000	0000	0000	0000	0011	0000	0010
1	0001	0001	0001	0001	0100	0001	0110
2	0010	0010	0010	0011	0101	0011	0111
3	0011	0011	0011	0100	0110	0010	0101
4	0100	0100	0100	0101	0111	0110	0100
5	0101	1011	1000	1000	1000	0111	1100
6	0110	1100	1001	1001	1001	0101	1101
7	0111	1101	1010	1011	1010	0100	1111
8	1000	1110	1011	1100	1011	1100	1110
9	1001	1111	1100	1101	1100	1101	1010

Table 1.3 Various BCD codes

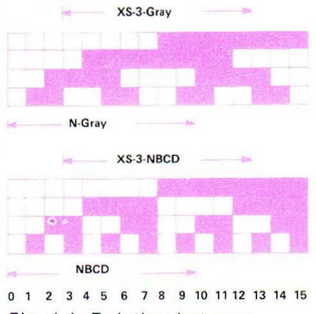


Fig. 1.1. Relation between NBCD and Gray codes

	Bi-quinary		Qui-binary	
	50	43210	86420	10
0	01	00001	00001	01
1	01	00010	00001	10
2	01	00100	00010	01
3	01	01000	00010	10
4	01	10000	00100	01
5	10	00001	00100	10
6	10	00010	01000	01
7	10	00100	01000	10
8	10	01000	10000	01
9	10	10000	10000	10

Table 1.4 2-out of-7 codes

When converting information from analog to digital form or *vice versa*, difficulties very often arise when a step between two code groups requires a change of more than one bit. Thus, when two or more digits are changed, perfect adjustment of circuitry is necessary to prevent the generation of false code combinations owing to lack of simultaneous operation.

For instance, when 7 is changed to 8 in NBCD, all 4 bits have to change. Special codes (cyclic or progressive codes) have therefore been developed which have the property that each code group differs in only one bit from its predecessor. They are cyclic in the sense that when the last decimal digit (15 in case of a 4-bit code) changes to 16 (0) only 1 bit has to change too, see Fig. 1.1.

The most commonly used cyclic code is the Gray code. If we only use the first 10 combinations of the Gray codes (in the interests of BCD coding), we get the "natural" Gray code. The main advantage of the Gray code is lost here, however, because we now need to change three bits to get from 9 to 0. By shifting the Gray code three positions, a code analogous to the binary XS-3 is obtained, which is called the Excess – 3 Gray code (XS-3 Gray). This code not only overcomes the 9-to-0 transition problem but is also self-complementing. (Only the most significant bit has to be inverted to obtain the complement.)

The Gray code is much older than our digital instruments; it was invented in 1880 by a French engineer Emile Baudot for mechanical applications and also represented a major advance in telegraphy.

There are many more codes in use to-day, much too many to deal with in detail here. We will conclude this section on numerical codes with a brief discussion of two more important codes, the 2-out of-7 codes. These codes have been frequently used in digital instruments. There are two versions in use: the biquinary or 50-43210 code and the quibinary or 86420-10 code. These codes have the advantage of constant loading and easy error detection because there are always exactly two and not more or less than two. Both codes are true-weighting and are tabulated in Table 1.4.

The above codes are all numerical codes. The BCD code does not have enough bits for the transmission of other characters (e.g. letters, special symbols, etc.). In this case we make use of an alpha-numerical code, with more than four bits to represent the figures, letters or symbols. Well known examples are the **telex code** (5 bits), the ISO-7 bit code (ASCII code), the extended BCD code (8 bits) and the punched card code; see Tables 1.5, 1.6 and 1.7.

		Bits		B7	0	0	0	0	1	1	1	1
				B6	0	0	1	1	0	0	1	1
				B5	0	1	0	1	0	1	0	1
b_4	b_3	b_2	b_1									
0	0	0	0		NUL	DLE	SP	0		P		p
0	0	0	1		SOH	DC1	!	1	A	Q	a	q
0	0	1	0		STX	DC2	"	2	B	R	b	r
0	0	1	1		ETX	DC3	=	3	C	S	c	s
0	1	0	0		EOT	DC4	¤	4	D	T	d	t
0	1	0	1		ENQ	NAK	%	5	E	U	e	u
0	1	1	0		ACK	SYN	&	6	F	V	f	v
0	1	1	1		BEL	ETB	'	7	G	W	g	w
1	0	0	0		BS	CAN	(	8	H	X	h	x
1	0	0	1		HT	EM	)	9	I	Y	i	y
1	0	1	0		LF	SUB	*	:	J	Z	j	z
1	0	1	1		VT	ESC	+	;	K	[	k	{
1	1	0	0		FF	FS	,	<	L	\	l	|
1	1	0	1		CR	GS	–	=	M	]	m	}
1	1	1	0		S0	RS	.	>	N	ˆ	n	–
1	1	1	1		S1	US	/	?	O	_	o	DEL

Table 1.5 The ISO code. ISO WORD

b_7	b_6	b_5	b_4	b_3	b_2	b_1

		Bits		b8	0				0				1				1			
				b7	0				1				0				1			
				b6	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
				b5	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
b_4	b_3	b_2	b_1																	
0	0	0	0		NUL	DLE	DS		SP	&	–									0
0	0	0	1		SOH	DC1	SOS				/		a	j	~		A	J		1
0	0	1	0		STX	DC2	FS						b	k	s		B	K	S	2
0	0	1	1		ETX	DC3							c	l	t		C	L	T	3
0	1	0	0		PF	RES	BYP	PN					d	m	u		D	M	U	4
0	1	0	1		HT	NL	LF	RS					e	n	v		E	N	V	5
0	1	1	0		LC	BS	EOB	UC					f	o	w		F	O	W	6
0	1	1	1		DEL	IDL	PRE	EOT					g	p	x		G	P	X	7
1	0	0	0			CAN							h	q	y		H	Q	Y	8
1	0	0	1		RLF	EM						`	i	r	z		I	R	Z	9
1	0	1	0		SMM	CC	SM		ç	!	:	:								LVM
1	0	1	1		VT	CU1	CU2	CU3	.	$	,	#								
1	1	0	0		FF	IFS		DC4	<	*	%	@					⑀		⑁	
1	1	0	1		CR	IGS	ENQ	NAK	(	)	–	'								
1	1	1	0		SO	IRS	ACK		+	;	>	=					⑂			
1	1	1	1		SI	IUS	BEL	SUB	ı		?	"								

Table 1.6 The EXTENDED BCD code. Extended binary coded decimal interchange code. EBCDIC WORD

b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1

Each group of pulses representing a figure or another character is called a "word".
If such a word has to be transmitted via one single wire, the pulses and no pulses will follow each other in time. This is called BIT SERIAL CODING (Fig. 1.2.). If we had 8 lines available for the transmission of such a word, this complete word could be transmitted in one pulse interval. This is called BIT PARALLEL CODING (Fig. 1.3.).

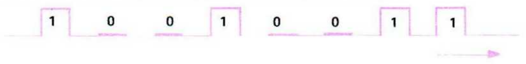


Fig. 1.2 Bit serial coding of the decimal 39 (if the BCD code is used) or 147 (with the pure binary code).

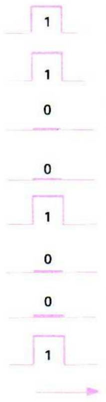


Fig. 1.3 Bit parallel coding of the same number as in Fig. 1.2.

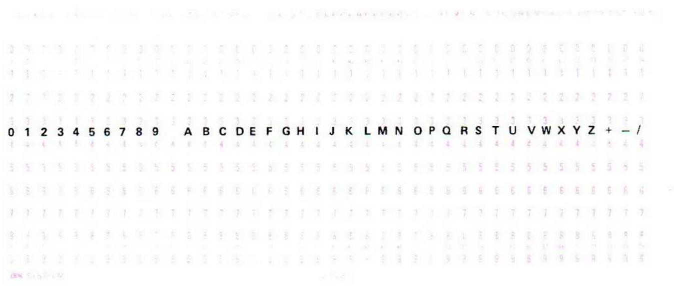

0 1 2 3 4 5 6 7 8 9 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z + – /

Table 1.7 Punched-card or Hollerith code.

Questions:

Q.1.1. How many digits in binary notation are required for the decimal number 17?

A	4	A	
B	5	B	
C	7	C	

Q.1.2. When an even decimal number is converted into the binary number system the least significant digit LSD is

A	1	A	
B	0	B	
C	1 or 0	C	

Q.1.3. Number 85 in standard BCD is

A	1000-1100	A	
B	1101-1010	B	
C	1000-1010	C	

Q.1.4. 0110–0001–1001 in standard BCD is the decimal number

A	615	A	
B	916	B	
C	619	C	

Q.1.5. The (r – 1)'s complement of the (r – 1)'s complement of an integer N is:

A	The (r – 2)'s complement	A	
B	The original number (N)	B	
C	The r's complement	C	

Q.1.6. The 10's complement of 715_8 is:

A	63	A	
B	539	B	
C	285	C	

Q.1.7. The sum of weights in a self-complementing BCD code must be:

A	8	A	
B	9	A	
C	10	C	

Q.1.8. How many different 4-bit BCD codes can be developed?

A	17	A	
B	256	B	
C	$>$ 256	C	

Q.1.9. How many bits does one need to encode all letters (26), 10 symbols and all numerals (10)?

A	5	A	
B	7	B	
C	6	C	

Q.1.10. What is the advantage of serial transport compared with parallel transfer for data transmission?

A	Needs only one wire	A	
B	Is faster	B	
C	Is BCD compatible	C	

Chapter 2

Boolean algebra

As we saw in the previous chapter, the switching elements in digital instruments can have one of two distinct states (''on'' or ''off''; ''pulse'' or ''no pulse''; ''high voltage'' or ''low voltage'', etc.). In theory, intermediate states don't exist, and an element can never be in both possible states at the same time (for example, a switch can never be simultaneously on and off).
It has been found that the laws dealing with the various combinations of these two states in different switching elements are formally identical with the laws governing the relation between logical propositions which may be either ''true'' of ''false''. The basis for this branch of logic was laid by the Greek philosopher Aristotle in classical times, but it was not until 1847 that George Boole published a mathematical description of the laws of ''propositional logic'' in a paper entitled ''An investigation of the Law of Thought''. This branch of mathematics, known as Boolean algebra, will be described briefly in this chapter. We shall see in chapter 3 how useful Boolean algebra is in providing a concise description of the properties of switching elements. It is because the switching elements used in digital instruments obey the laws of Boolean algebra, the mathematics of logic, that they are generally referred to as ''logic elements''.

The logic of classes

Classes

In mathematical logic a class is defined as a group of elements all of which possess at least one characteristic in common. Elements not possessing this common characteristic form the complementary class.

Examples of classes are:
a. the class of all ''natural'' numbers (1, 2, 3, 4, 5...)
b. the class of electronic measuring instruments
c. the class of all human beings,
and so on.

A class can contain a finite or an infinite number of elements. A class containing no elements at all is said to be ''empty''; this ''null class'' is indicated by the symbol ''0''. If a class contains at least one element it is said to be non-empty and may be denoted by the symbol ''1''.
A class together with its complementary class form the ''universe'' or universal class.

Subclasses

A class can be divided further into subclasses, the elements of which share a certain characteristic in addition to the characteristic defining the group as a whole.
By way of example, let us consider a couple of subclasses of the infinite class of natural numbers (1, 2, 3, 4, 5...), which we shall denote by the letter A. Numbers which are not natural numbers are elements of the complementary class $\bar{A}$. (The line above the symbol is used in Boolean algebra to indicate the process of negation; the symbol $\bar{A}$ is read ''A bar'', or ''not A''.) A given number can be an element of A or $\bar{A}$, but never of both.
Subclasses of A are for example the class of natural numbers which are divisible by 3, or the class of natural numbers less than 100. Such subclasses can be denoted by another letter, e.g. B.

Venn diagram

Properties of classes can be visualized by means of the Venn diagram, developed by the English logician John Venn.
In a Venn diagram the universe is symbolized by a square and a given class (A) by the area within a circle or ellipse drawn inside this square. The remaining area ($\bar{A}$) contains elements **not** belonging to class A. Together, (A) and ($\bar{A}$) form the universe. It can be clearly seen from this diagram that an element can never belong to both classes A and $\bar{A}$ (fig. 2.1.).

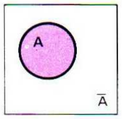


Fig. 2.1

We will now make use of Venn diagrams to illustrate various relationships between the subclasses of the infinite class of natural numbers which we mentioned above (fig. 2.2.).

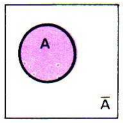


Fig. 2.2a

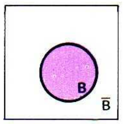
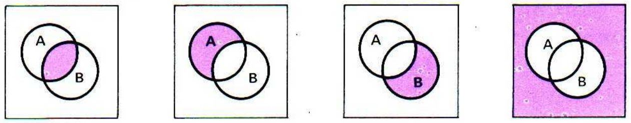


Fig. 2.2b

Let the class of natural numbers divisible by 3 be denoted by A, and the class of natural numbers smaller than 100 be denoted by B. We can now define four further classes obtained by combination of A and B, viz.:

1. All multiples of 3 (A) which are smaller than 100 (B)
2. All multiples of 3 (A) which are greater than 100 ($\overline{B}$)
3. All numbers which are NOT multiples of 3 ($\overline{A}$), and which are smaller than 100 (B)
4. All numbers which are NOT multiples of 3 ($\overline{A}$), and which are greater than 100 ($\overline{B}$).

These various subclasses can be represented very simply in the Venn diagram, as shown by the shaded areas in fig. 2.3.



A.B *A.$\overline{B}$* *$\overline{A}$.B* *$\overline{A}$.$\overline{B}$*

Fig. 2.3

Intersection (conjunction, logic multiplication); min-terms
Elements which belong to class A **and** class B form the **intersection** (or **conjunction**) or logic multiplication of the two classes. The operation of intersection is denoted in Boolean algebra in the same way as the operation of multiplication in normal algebra, by writing a full stop between the symbols concerned or merely by writing the two symbols together:

A and B = A.B. = AB.

It will be clear from the definition that the intersection of A and B is identical with the intersection of B and A, so we can write:

A.B. = B.A.

Inspection of a Venn diagram of the two variables A and B shows clearly that there are four possible conjunctions of these variables: A.B; A.$\overline{B}$.; $\overline{A}$.B and $\overline{A}$.$\overline{B}$. Finer sub-division of the space within the Venn diagram is not possible with the aid of the variables A and B; the subclasses A.B, A.$\overline{B}$, $\overline{A}$.B and $\overline{A}$.$\overline{B}$ are therefore also called "min-terms". These four min-terms are indicated by the shaded areas in fig. 2.3.

Union (disjunction, logic addition); max-terms
Negation of the intersections illustrated in fig. 2.3. gives a new kind of relationship, the union (see fig. 2.4.). Elements which belong to class A **or** class B or **both** form a class which is called the **union** or **disjunction** of A and B. Union in Boolean algebra is written like addition in normal algebra:

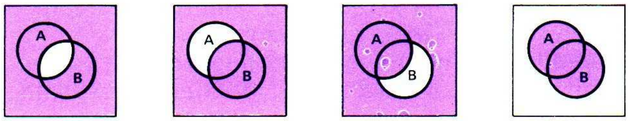
A or B = A + B

Here again, it makes no difference whether we talk of the union of A and B or the union of B and A, so we may write:

A + B = B + A

As may be seen from fig. 2.4, four possible unions can be formed starting from A and B, viz: $\overline{A}$ + $\overline{B}$, $\overline{A}$ + B, A + $\overline{B}$ and A + B. These four subclasses are the largest ones which can be formed from the two variables A and B, so they are also called "max-terms".

In all there are sixteen different functions which can be formed from the logic variables A and B. These are tabulated in Table 2.1, together with their names, notations and Venn-diagram representations, and indication of the various min- and max-terms.



$\overline{A}$ + $\overline{B}$ *$\overline{A}$ + B* *A + $\overline{B}$* *A + B*

Fig. 2.4

Laws of Boolean algebra

We already learned two laws, the commutive laws

A.B = B.A. (1)

and

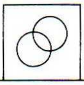
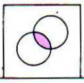
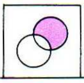
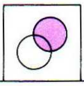
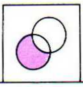
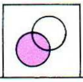
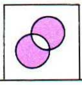
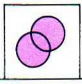
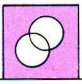
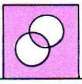
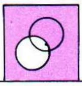
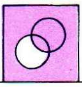
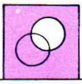
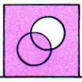
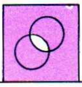
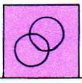
A + B = B + A (2)

We will now consider a number of more complicated laws which are very important for manipulation of Boolean functions.

De Morgan's theorem
Inspection of the smallest subclass, A.B (mini-term m_3 in Table 2.1.) and the largest, $\overline{A}$ + $\overline{B}$ (max-term M_0 in Table 2.1.) shows that these two are complementary, i.e. when added together they give the universal class. In Boolean notation: AB + ($\overline{A}$ + $\overline{B}$) = 1.

Another way to put this relation is to say that A.B is the **inverse** or negation of $\overline{A}$ + $\overline{B}$, or in Boolean notation:

A.B. = $\overline{\overline{A} + \overline{B}}$ or $\overline{A.B.}$ = $\overline{A}$ + $\overline{B}$ (3)

Nº	Name	Notation	X = f (A,B) for A = 0 0 1 1 B = 0 1 0 1	Venn diagram
0	Zero, false	0	X = 0 0 0 0	
1	And, conjunction	$A.B$ (m_3)	X = 0 0 0 1	
2	Exclusion	$A.\bar{B}$ (m_2)	X = 0 0 1 0	
3	Identity	A	X = 0 0 1 1	
4	Exclusion	$\bar{A}.B$ (m_1)	X = 0 1 0 0	
5	Identity	B	X = 0 1 0 1	
6	Exclusive OR	$A \oplus B$	X = 0 1 1 0	
7	OR, disjunction	$A+B$ (M_3)	X = 0 1 1 1	
8	Nor, nondisjunction	$\overline{A+B}$ (m_0)	X = 1 0 0 0	
9	Equivalence	$A \equiv B$	X = 1 0 0 1	
10	Negation Not, inversion	$\bar{B}$	X = 1 0 1 0	
11	Inclusion	$A+\bar{B}$ (M_2)	X = 1 0 1 1	
12	Negation Not, inversion	$\bar{A}$	X = 1 1 0 0	
13	Inclusion	$\bar{A}+B$ (M_1)	X = 1 1 0 1	
14	Nand, non-conjunction	$\overline{A.B}$ (M_0)	X = 1 1 1 0	
15	Unity, one, True	1	X = 1 1 1 1	

Similarly, for the **other** subclasses of fig. 2.3. and 2.4.:

$A.\bar{B}. = \overline{\bar{A}+B}$ or $\overline{A.\bar{B}}. = \bar{A}+B$ (3a)

$\bar{A}.B. = \overline{A+\bar{B}}$ or $\overline{\bar{A}.B}. = A+\bar{B}$ (3b)

$\bar{A}.\bar{B}. = \overline{A+B}$ or $\overline{\bar{A}.\bar{B}}. = A+B$ (3c)

The above four equations represent special cases of a very important law of Boolean algebra, known as De Morgan's theorem. This theorem may be stated in words as follows: any expression in Boolean algebra is equivalent to the inverse of the expression formed by replacing all logical products (AND's) by logical additions (OR's) and *vice versa*, and by simultaneously replacing each operand by its inverse.

Relations between a class A, class 1 and class 0

As we have mentioned above, the union of a class A and its inverse $\bar{A}$ is the universal class, or in Boolean notation:

$A+\bar{A}=1$ (4)

Furthermore, the intersection of A and $\bar{A}$ is the null class 0, since no element belongs to both A and $\bar{A}$:

$A.\bar{A}=0$ (5)

The negation of the negation of a class is the class itself:

$\bar{\bar{A}}=A$ (6)

Various other relations can easily be developed with reference to the Venn diagram.

The logical addition or union of a class A and the null class is again the class A:

$A+0=A$ (7)

The union of A with the universal class is the universal class:

$A+1=1$ (8)

The logic product or intersection of a class A with the null class is the null class:

$A.0=0$ (9)

and the intersection of A with the universal class is A:

$A.1=A$ (10)

The intersection of A with itself is A:

$A.A=A$ (11)

and the union of A with itself is also A:

$A+A=A$ (12)

m = min-term
M = max-term
Table 2.1. The sixteen functions of two variables

Associative laws

So far, we have discussed relations between at most two variables, A an B. Of course, it is possible to consider more variables. We shall now give a number of relations between three variables, with reference to the Venn diagram of fig. 2.5.

The intersection (logical product) of classes A, B and C in fig. 2.5. could be determined by first finding the intersection of A and B (areas 6+7 in fig. 2.5.), and then by finding the intersection of (A.B) with C (area 7).

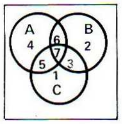


Fig. 2.5

Alternatively, we could start by taking the intersection of B an C (areas 7+3) and then finding the intersection of this with A (which again gives area 7, naturally). Whichever way we carry out this process, the final result is the same. In Boolean notation:

$$A.B.C. = A.\ (B.C.) = (A.B.)\ C = (A.C.).B. \qquad (13)$$

A similar relationship holds for the union (disjunction) of three classes:

$$A+B+C = A+(B+C) = (A+B)+C = (A+C)+B \qquad (14)$$

The two above relationships are called the **associative laws** of Boolean algebra.

Distributive laws

The subclass formed as the intersection of A with the union of B and C (areas 5, 6 and 7 in fig. 2.5.) can also be formed by taking the union of the intersections of A and B and A and C (areas 6+7 and 7+5 in fig. 2.5., giving areas 5+6+7):

$$A\ (B+C) = A.B. + A.C. \qquad (15)$$

This relation has precisely the same form as the corresponding distributive law in normal algebra, which is not the case with the following distributive law of Boolean algebra:

$$(A+B)\ (A+C) = A+BC \qquad (16)$$

This can be proved with the aid of the various relations we have derived above:

$$\begin{aligned}
(A+B)\ (A+C) &= A.A. + AC + AB + BC = && \\
&= A + AC + AB + BC = && (11)\\
&= A(1+C) + AB + BC = && (15)\\
&= A + AB + BC = && (8)\ (10)\\
&= A(1+B) + BC = && (15)\\
&= A + BC && (8)\ (10)
\end{aligned}$$

Absorptive laws

In the course of the proof of equation (16) above we saw that the terms AB and AC in the initial expression of the logical product do not appear in the final form. This effect is known as the ''absorptive'' property of Boolean algebra:

$$A + AB = A \qquad (17)$$

$$\begin{aligned}
A + AB &= A(1+B) = && (15)\\
&= A1 = && (8)\\
&= A && (10)
\end{aligned}$$

There are two other absorptive laws in Boolean algebra:

$$A + \overline{A}B = A + B \qquad (18)$$

$$\begin{aligned}
A + \overline{A}B &= (A+\overline{A})\ (A+B) = && (16)\\
&= 1\ (A+B) = && (4)\\
&= A+B && (10)
\end{aligned}$$

and

$$A\ (\overline{A}+B) = A.B \qquad (19)$$

$$\begin{aligned}
A\ (\overline{A}+B) &= A.\overline{A} + AB = && (15)\\
&= 0 + AB = && (5)\\
&= AB && (7)
\end{aligned}$$

Summary of the laws of Boolean algebra

The various relations derived above are summarized in the following table (Table 2.2.).

$\overline{0}=1$	$x+0=x$	$x.0=0$
$\overline{1}=0$	$x+1=1$	$x.1=x$
$\overline{\overline{x}}=x$	$x+x=x$	$x.x=x$
	$x+\overline{x}=1$	$x.\overline{x}=0$
commutative laws		
$xy=yx$	$x+y=y+x$	
associative laws		
$xyz=x(yz)=y(xz)=z(xy)$		
$x+y+z=x+(y+z)=y+(x+z)=z+(x+y)$		
distributive laws		
$x\ (y+z)=xy+xz$	$(x+y)\ (x+z)=x+yz$	
absorptive laws		
$x+xy=x$	$x+\overline{x}y=x+y$	$x\ (\overline{x}+y)=xy$
De Morgan's laws		
$\overline{x+y+z+\ldots+N}=\overline{x}.\overline{y}.\overline{z}\ldots.\overline{N}$		
$\overline{x.y.z.\ \ldots.N}=\overline{x}+\overline{y}+\overline{z}+\ \ldots\ +\overline{N}$		

Table 2.2.

Simplification of Boolean functions

The above rules provide us with very powerful means of simplifying Boolean functions.

An example will illustrate this:

Suppose we have the following function:

$$F_{ABC} = AB\overline{C} + A\overline{B}\overline{C} + \overline{A}B\overline{C} + ABC + A\overline{B}C + \overline{A}BC = \text{Rule 15}$$

(6) (4) (2) (7) (5) (3) (Corresponding areas in Fig. 2.5)

$$\begin{aligned}
&= A\overline{C}\,(B+\overline{B}) + \overline{A}B\overline{C} + AC\,(B+\overline{B}) + \overline{A}BC && = \text{Rule 4}\\
&= A\overline{C} + \overline{A}B\overline{C} + AC + \overline{A}BC && = \text{Rule 15}\\
&= (A+\overline{A}B)\overline{C} + (A+\overline{A}B)\,C && = \text{Rule 18}\\
&= (A+B)\overline{C} + (A+B)\,C && = \text{Rule 15}\\
&= (A+B)\,(C+\overline{C}) && = \text{Rule 4}\\
&= A+B
\end{aligned}$$

which is also clearly shown in the Venn diagram of fig. 2.5:

Areas: 6+4+2+7+5+3, together give A+B

Min-terms and max-terms

We have already seen above that the Boolean functions of two variables A and B include four min-terms, $\overline{A}.\overline{B}$, $\overline{A}B$, $A\overline{B}$ and A.B. In general, a min-term of n variables can be defined as the **Boolean product** of these n variables, each variable being present in its true or complemented form (1 or 0). Similarly the max-term of n variables is defined as the **Boolean sum** of these n variables, with each variable present in its true or its complemented form. As table 2.3 shows, there are eight min-terms and eight max-terms for a Boolean function of 3 variables (A, B and C).

A B C	Min-term	Notation	Max-term	Notation
0 0 0	$\overline{A}.\overline{B}.\overline{C}$	m_0	$\overline{A}+\overline{B}+\overline{C}$	M_0
0 0 1	$\overline{A}.\overline{B}.C$	m_1	$\overline{A}+\overline{B}+C$	M_1
0 1 0	$\overline{A}.B.\overline{C}$	m_2	$\overline{A}+B+\overline{C}$	M_2
0 1 1	$\overline{A}.B.C$	m_3	$\overline{A}+B+C$	M_3
1 0 0	$A.\overline{B}.\overline{C}$	m_4	$A+\overline{B}+\overline{C}$	M_4
1 0 1	$A.\overline{B}.C$	m_5	$A+\overline{B}+C$	M_5
1 1 0	$A.B.\overline{C}$	m_6	$A+B+\overline{C}$	M_6
1 1 1	$A.B.C$	m_7	$A+B+C$	M_7

Table 2.3 Min- and max-terms

In general there are 2^n different min-terms and 2^n different max-terms of n variables.

As shown in the table, each min- (or max-)term is given a symbol m (or M) with an index i, which is equal to the decimal value of the corresponding function (with A=B=C=1). There is of course a relationship between the min- and max-terms, as can be seen clearly in fig. 2.3 and 2.4. The complement of a min-term is a max-term and *vica versa*

In general:

$$\overline{m}_i = M_{2^n-1-i} \quad \text{and} \qquad (20)$$

$$\overline{M}_i = m_{2^n-1-i} \qquad (21)$$

e.g. for 3 variables:

$$\overline{m}_i = M_{7-i} \quad \text{and} \quad \overline{M}_i = m_{7-i}$$

$\overline{m}_0 = M_7$	$\overline{m}_4 = M_3$
$\overline{m}_1 = M_6$	$\overline{m}_5 = M_2$
$\overline{m}_2 = M_5$	$\overline{m}_6 = M_1$
$\overline{m}_3 = M_4$	$\overline{m}_7 = M_0$

These are illustrated in fig. 2.6

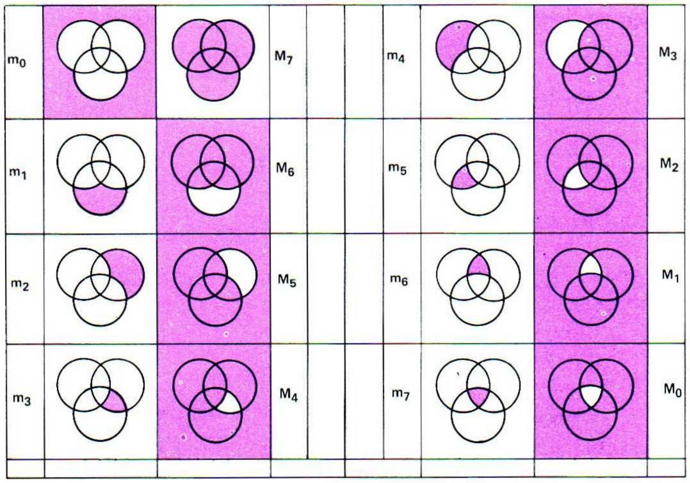


Fig. 2.6 The min-terms and max-terms of 3 variables

The min-terms of 3 variables are also indicated by their numbers in the areas of fig. 2.5

Fig. 2.6 shows very clearly that a min-term represents one of the smallest distinguishable areas in the Venn diagram, while a max-term represents one of the largest distinguishable areas: A very important law of Boolean algebra is: Any Boolean function can be expressed as a (Boolean) sum of min-terms.

In the example of the previous section (simplification of Boolean functions), F_{ABC} is $m_6+m_4+m_2+m_7+m_5+m_3$.

This is often simply written: $F_{ABC} = \Sigma\,(2,3,4,5,6,7)$

Where Σ is the summation sign.

As we know, there are eight min-terms of three variables we used six of these in the above example, so there must be two left: m_0 and m_1
Inspection of the Venn diagram of fig. 2.5. shows that Σ (0,1) is the inverse of the above function:
$\overline{F}_{ABC} = \Sigma\,(0,1) = m_0 + m_1$.

Application of rule 20 gives: $F_{ABC} = M_6.M_7$ (or in "short notation $F_{ABC} = \Pi\,(6,7)$.). This illustrates a second important law related to the above:
Any Boolean function can be expressed as the (Boolean) product of max-terms.
Proof: $M_7.M_6 = (A+B+\overline{C})\,(A+B+C) =$
$= A+AB+AC+AB+B+BC+A\overline{C}+B\overline{C}+C\overline{C} =$
$= A+B$

The above rule applies even when not all variables are present in the various terms of a Boolean function. For example, let us consider the function:

$F_{ABC} = \overline{A}B + AC$. This can be written
$= \overline{A}B\,(C+\overline{C}) + A\,(B+\overline{B})\,C =$
$= \overline{A}B\overline{C} + \overline{A}BC + ABC + A\overline{B}C = m_2 + m_3 + m_5 + m_7$

Or in the short notation $F_{ABC} = \Sigma$ (2,3,5,7)

Alternatively the function can be rewritten as a product of max-terms as follows:
$F_{ABC} = \overline{A}B + AC = (\overline{A}B + A)\;(\overline{A}B + C) =$
$= (\overline{A}+A)\;(A+B)\;(\overline{A}+C)\;(B+C)$
$\overline{A}+A = 1$
$A+B = (A+B+C)\;(A+B+\overline{C}) = M_7.M_6$
$\overline{A}+C = (\overline{A}+B+C)\;(\overline{A}+\overline{B}+C) = M_3.M_1$
$B+C = (A+B+C)\;(\overline{A}+B+C) = M_7.M_3$
$F_{ABC} = M_7.M_6.M_3.M_1.M_7.M_3 = \Pi(1,3,6,7)$

The logic of propositions
The theorems of Boolean algebra apply not only to the logic of classes but also to the logic of propositions.
Propositional logic is also based on the assumption that there are only two possibilities as far as a proposition or statement is concerned: the statement can be either **true** or **false**, but never partly true and partly false. If a given statement is true, we denote this by the symbol "1", while if it is false we denote this by a "0". The truth or falsehood of combined propositions can now be determined according to rules which are formally the same as those we derived above for the logic of classes.

Here again, we make use of two basic operations: the **conjunction** (intersection, AND function), which is the proposition that both A and B are true, and the **disjunction** (union, OR function), which is the proposition that A or B or both are true. Let us consider by way of example the propositions that someone has blue eyes (A = 1), and that someone is an adult (B = 1). The conjunction of these two propositions is the statement that someone has blue eyes **and** is an adult, and can be written in Boolean notation in the form A.B = 1. If we denote the derived proposition by X, we can also write X = A.B.

Now when is this statement true? This can be determined with the aid of a Venn diagram, but also by tabulating all the possible combinations of the variables A and B in a "truth table", table 2.4.

A	B	X		A	B	X
False	False	False		0	0	0
True	False	False	or	1	0	0
False	True	False		0	1	0
True	True	True		1	1	1

Table 2.4 Truth table for X = A.B.

The above truth table shows clearly that X is only true when A AND B are both true.
Of course the same procedure can be followed for the operation of disjunction (union, OR).

A	B	X		A	B	X
False	False	False		0	0	0
True	False	True	or	1	0	1
False	True	True		0	1	1
True	True	True		1	1	1

Table 2.5 Truth table for $X = A + B$.

The disjunction of the propositions A and B is written: $X = A + B$, giving the truth table of Table 2.5.
which clearly shows that the combined proposition is true when A OR B OR both are true.
Switching algebra, a derivative of propositional logic, will be used in the following chapter as an aid to the description of logic elements.
This is done by giving the Boolean function and the truth table of each of the logic elements discussed in chapter 3. Anticipating the subject matter of that chapter somewhat, we shall show briefly how this is done.

Let us take by way of example the AND gate discussed on page 24. By definition, the output of this gate will be "1" if and only if all of its inputs are 1. Now in the language of propositional logic, we can denote the statement e.g. "there is a high voltage at the output of the AND gate" at X, and if this statement is true we can write $X = 1$. Similarly we can denote the proposition "there is a high voltage at input A; if this proposition is true we write $A = 1$; the same method can be followed for the other inputs. Now the definition of the operation of the AND gate can be translated quite simply into Boolean notation as $A.B.C = X$.
The truth table for this gate can be built up by substituting the various possible combinations of 0's and 1's for A, B and C in this expression, and seeing what X is. We saw on page 17 that the logical product $A.0 = 0$; this means that as long as there is one 0 among the factors A, B and C, the product will be 0.

Questions

Q.2.1. In the universal class of electronic instruments we define three subclasses: X All electronic measuring instruments, Y All digital instruments, Z All instruments with battery supply.
The Boolean expression for the subclass (S) of all battery-supplied digital electronic measuring instruments can be written:

A $S = XYZ$

B $S = X + Y + Z$

C $S = \overline{X}\overline{Y}\overline{Z}$

A	
B	
C	

Q.2.2. The Boolean expression for the subclass (R) of all digital electronic instruments which are **not** measuring instrument and have **no** battery supply is:

A $R = X + \overline{Y}\overline{Z}$

B $R = XY + Z$

C $R = \overline{X}Y\overline{Z}$

A	
B	
C	

Q.2.3. The Boolean expression for the subclass (Q) of all electronic instruments, which are measuring instruments or are non-digital instruments with battery supply is:

A $Q = X(Y + \overline{Z})$

B $Q = X + \overline{Y}Z$

C $Q = X\overline{Y} + Z$

A	
B	
C	

Q.2.4. The function $F_{XYZ} = \Sigma\ (1,3,7)$ can also be written as:

A $\Sigma\ (0,2,4,5,6)$

B $\Pi\ (1,3,7)$

C $\Pi\ (0,2,4,5,6)$

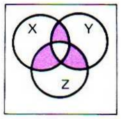
A	
B	
C	

Q.2.5. The Boolean expression for the shaded area in the accompanying Venn diagram is:

A $XY + YZ + XZ$

B $XY\overline{Z} + \overline{X}YZ + X\overline{Y}Z$

C $XYZ + \overline{X}\overline{Y}\overline{Z}$



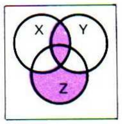
A	
B	
C	

Q.2.6. The Boolean expression for the shaded area in the accompanying Venn diagram is:

A $X + \overline{Y} + Z$

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

C $\overline{X}\overline{Y}Z + XY$



A	
B	
C	

Q.2.7. The simplified form of the Boolean function $F(X, Y, Z) = (X + Y + XY)(X + Z)$ is:

A $= X + Y + Z$

B $= X + YZ$

C $= XY + YZ$

A	
B	
C	

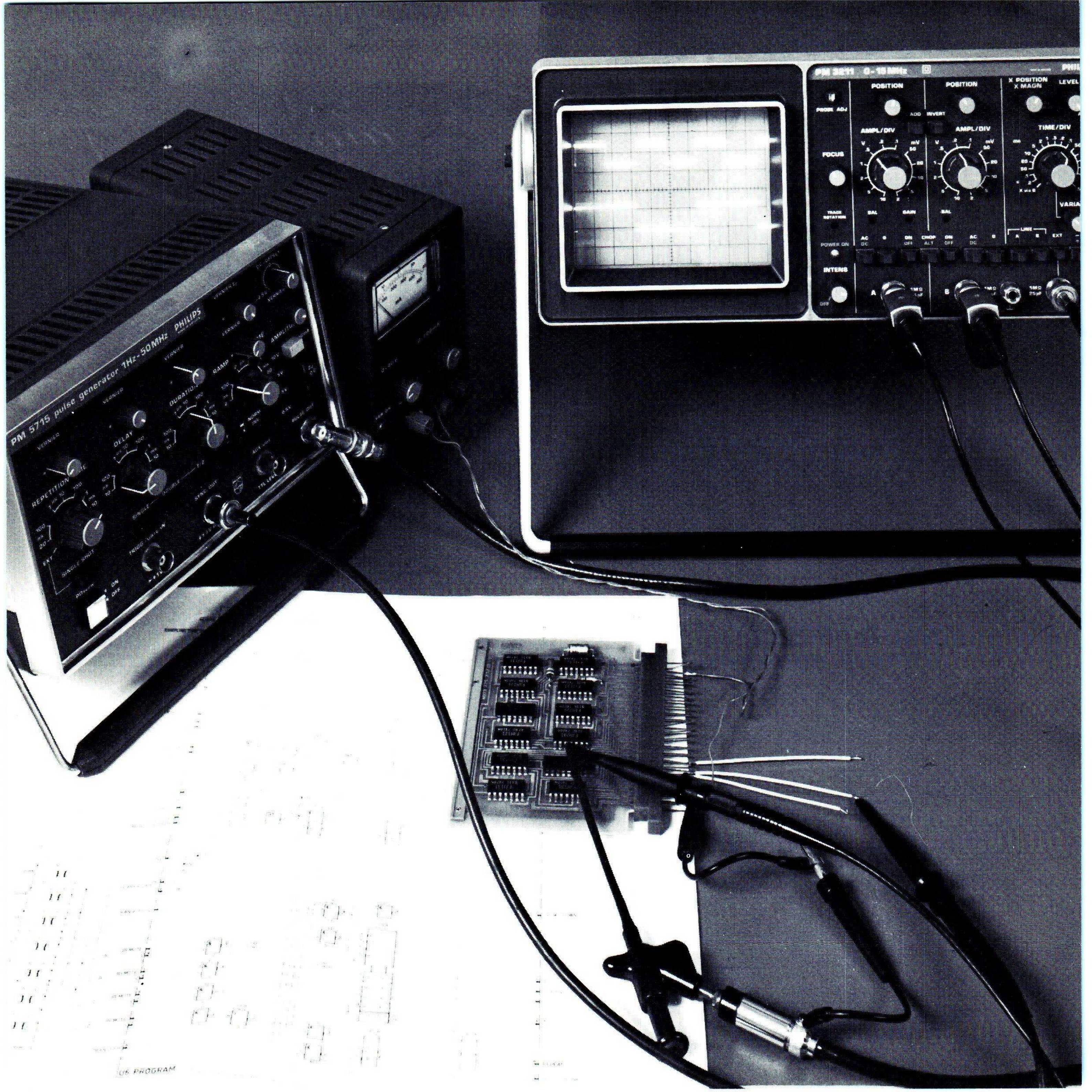
Q.2.8. The simplified form of the Boolean function $F(X, Y, Z) = (X + \overline{Y} + Z)(X + \overline{Y} + \overline{Z})(X + Y + Z)$ is:

A $= \overline{X}Y + \overline{Z}$

B $= X$

C $= X + \overline{Y}Z$

A	
B	
C	


PM 5715 pulse generator 1Hz-50MHz
PHILIPS
PM 3211 0-15 MHz
POSITION
AMPL/DIV
TIME/DIV
FOCUS
INTENS
POWER ON
REPETITION
DELAY
DURATION
RAMP
SINGLE SHOT
POWER
ON
OFF
US PROGRAM

Chapter 3

Logic elements

The logic elements in digital instruments are the basic building blocks of the circuits that control data flow and processing of standard signals. The standard symbols for the main logic elements are shown in fig. 3.1. Each logic element is a network of electronic components.

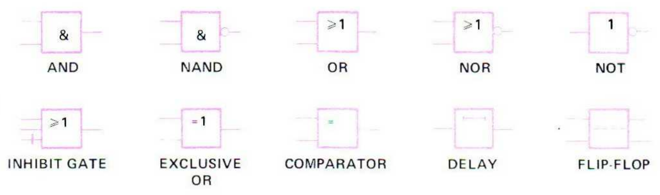


*Fig. 3.1. The standard symbols for the main logic elements.**

Logic elements generally have very graphic names, e.g. AND; OR; NOT; NAND; NOR; INHIBIT GATE; EXCLUSIVE-OR; COMPARATOR; DELAY and FLIP-FLOP, which provide a very compact description of the functions performed.

We shall now describe the various logic elements in turn. In each case we shall start with a definition of the mode of operation of the element, followed by its symbol, equivalent circuit diagram, truth table and Boolean switching function: finally we give for each element a brief discussion of its operating conditions and applications.

* Note. The graphical symbols used here for the logic elements all conform with the IEC publication 117-15A (1972). For a survey of other symbols used (American standard; German DIN 40700) see Table 3.1.

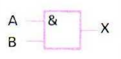
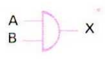
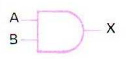
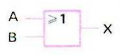
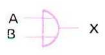
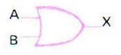
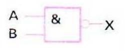
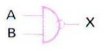
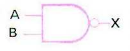
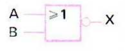
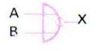
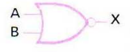
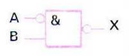
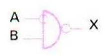
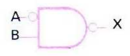
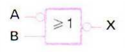
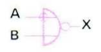
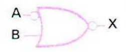
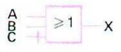
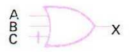
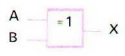
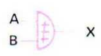
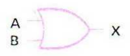
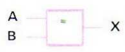
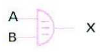
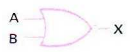
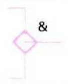
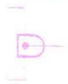
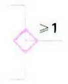
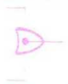
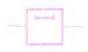
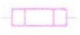
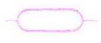
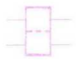
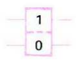
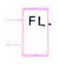
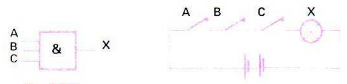
Circuit	IEC norm	DIN norm 40700	American standard	Boolean function
AND				$X = AB$
OR				$X = A+B$
NAND				$X = \overline{AB}$
NOR				$X = \overline{A+B}$
NAND with one inverting input				$X = \overline{\overline{A}B}$
NOR with one inverting input				$X = \overline{\overline{A}+B}$
Inhibit gate				$X = (A+B)\overline{C}$
Exclusive OR				$X = A\overline{B} + \overline{A}B$ $A \oplus B$
Comperator				$X = AB + \overline{A}\,\overline{B}$ $A \equiv B$
Distributed AND				
Distributed OR				
Delay				
Flip-flop				

*Table 3.1 Standard symbols for logic elements.**

The AND gate

Definition: The output of the AND gate will stand at its defined "**1**" state if, and only if, all of the inputs stand at their defined "**1**" states.
The input signals are **high** or **low** voltages, **pulses** or **no-pulses**, as are the outputs, representing the binary digits **one** and **zero**. If both inputs are **zero** the output is zero. If either input is **zero**, the output will again be **zero**. But if both inputs are **one**, there is a **one** output. In other words a **one** input AND a **one** input results in a **one** output. Hence the name AND gate.
The functioning of such a gate is like that of a set of switches in series, fig. 3.2. Only when they are closed simultaneously can there be an output. The AND gate is used primarily as a control element with one input regulating the traffic through the others. If a word as to be allowed to pass through the gate, a **one** at the control input will open the gate. The **zeros** in the word are maintained in the right position at the output because the gate is closed whenever there is at least one **zero** input.



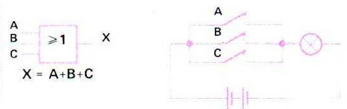
X = ABC

A	B	C	X
0	0	0	0
1	0	0	0
0	1	0	0
1	1	0	0
0	0	1	0
1	0	1	0
0	1	1	0
1	1	1	1

Fig. 3.2. Symbol, Boolean function, equivalent circuit and truth table of the AND gate.

The OR gate

Definition: The output of the OR gate will stand at its defined "**1**" state if, and only if, one or more of its inputs stand at their defined "**1**" state.



X = A+B+C

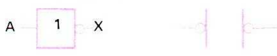
A	B	C	X
0	0	0	0
1	0	0	1
0	1	0	1
1	1	0	1
0	0	1	1
1	0	1	1
0	1	1	1
1	1	1	1

Fig. 3.3 Symbol, Boolean function, equivalent circuit and truth table of the OR gate.

The OR gate thus differs from the AND gate in that a **one** at one input OR the other input will give a **one** output. Hence the name OR gate. However, two **zero** inputs give a **zero** output and two **one** inputs give a **one** output. The functioning of the OR gate is similar to a set of switches connected in parallel (fig. 3.3.).
When any one arm of the circuit, or two or all has a **one** input a **one** output results.
The OR gate is designed to prevent interaction or feedback between inputs.

NOT gate (The inverter)

Definition: The output will stand at its "**0**" state if, and only if, the input stand at its defined "**1**" state.



$X = \overline{A}$

A	X
0	1
1	0

Fig. 3.4 Symbols, Boolean function and truth table of the INVERTER gate, symbol for the gate alone (not used frequently) and in combination with another gate.

Figure 3.4 gives the symbol for the INVERTER used in logic circuitry. This unit simply inverts voltage levels, **zero** becoming **one** and **one** becoming **zero**. The inverter is never used by itself, but always in conjunction with another logic element; it is then represented by a small circle directly connected to the other logic element.
The above are the basic elements. A number of variants are shown in table 3.2.

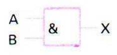
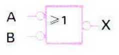
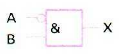
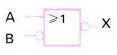
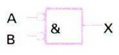
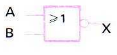
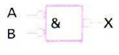
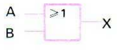
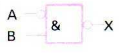
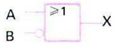
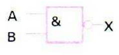
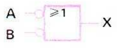
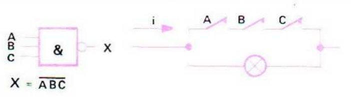
Boolean function	Gates: AND	A B X	OR	Boolean function
$X = AB$		0 0 0 1 0 0 0 1 0 1 1 1		$X = \overline{\overline{A} + \overline{B}}$
$X = \overline{A}B$		0 0 0 1 0 0 0 1 1 1 1 0		$X = \overline{A + \overline{B}}$
$X = \overline{A}\,\overline{B}$		0 0 1 1 0 0 0 1 0 1 1 0		$X = \overline{A + B}$
$X = \overline{\overline{A}\,\overline{B}}$		0 0 0 1 0 1 0 1 1 1 1 1		$X = A + B$
$X = \overline{\overline{A}B}$		0 0 1 1 0 1 0 1 0 1 1 1		$X = A + \overline{B}$
$X = \overline{AB}$		0 0 1 1 0 1 0 1 1 1 1 0		$X = \overline{A} + \overline{B}$

Table 3.2 Variants on the basic types of logic elements.

The NAND gate

Definition: The output will stand at its "**0**" state if, and only if, all inputs stand at their defined "**1**" states.



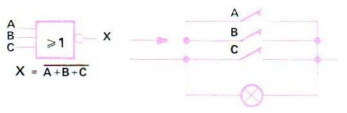
A	B	C	X
0	0	0	1
1	0	0	1
0	1	0	1
1	1	0	1
0	0	1	1
1	0	1	1
0	1	1	1
1	1	1	0

Fig. 3.5 Symbol, Boolean function, equivalent circuit and truth table of the NAND gate.

When an AND gate has an inverter at the output, the combined circuit is called a NAND gate, which is in effect the opposite of the AND gate. When **all** inputs are **one**, the output is **zero**.
The functioning of this gate is similar to a number of switches in series, in parallel with a load (lamp). At least one switch must be open in order to have the lamp ON (fig. 3.5.).

The NOR gate

Definition: The output will stand at its "**0**" state if, and only if, at least one input stand at its defined "**1**" state.



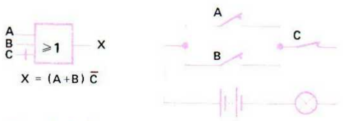
A	B	C	X
0	0	0	1
1	0	0	0
0	1	0	0
1	1	0	0
0	0	1	0
1	0	1	0
0	1	1	0
1	1	1	0

Fig. 3.6 Symbol, Boolean function, equivalent circuit and truth table of the NOR gate.

When an OR gate has an inverter at the output it becomes a NOR gate, which is in effect the opposite of the OR gate. When one or both inputs are **ones**, the output is **zero**, but when both inputs are **zeros** the output is **one**.
In other words when neither one input NOR the other is a **one**, the output is **one**. Hence the name NOR gate. The functioning of this gate is similar to that of a number of switches in parallel with a lamp or another load; all switches must be open if the lamp is to burn, fig. 3.6.

The INHIBIT gate

Definition: The inhibit gate is an OR gate with an inhibiting input: the output will stand at its "**1**" state if, and only if, the inhibit input stands at its defined "**0**" state AND one or more of the normal OR inputs stand at their defined "**1**" state.



A	B	C	X
0	0	0	0
1	0	0	1
0	1	0	1
1	1	0	1
0	0	1	0
1	0	1	0
0	1	1	0
1	1	1	0

Fig. 3.7 Symbol, Boolean function, equivalent circuit and truth table of the INHIBIT gate.

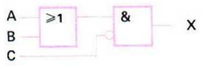
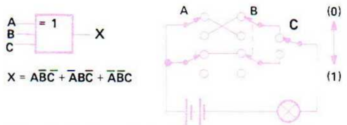


Fig. 3.8 The INHIBIT gate, regarded as the combination of an AND gate and an OR gate.

This gate is very useful for controlling A and B by means of the inhibiting signal C. When the inhibiting signal is present (C = 1) the output is always OFF (X = 0), but when the inhibiting signal absent (C = 0) the signals A and B can pass to the output X. The functioning of this gate is similar to that of the circuit of fig. 3.7.

The EXCLUSIVE-OR gate

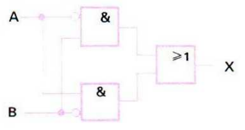
Definition: The output of the exclusive OR gate will stand at its defined "**1**" state only if **one**, and only **one**, of the inputs stands at its defined "**1**" state.
This gate may be regarded as a combination of AND and OR gates (see fig. 3.10.).


A
B
C
= 1
X
X = $A\overline{B}\overline{C} + \overline{A}B\overline{C} + \overline{A}\overline{B}C$
A
B
C
(0)
(1)

A	B	C	X
0	0	0	0
1	0	0	1
0	1	0	1
1	1	0	0
0	0	1	1
1	0	1	0
0	1	1	0
1	1	1	0

Fig. 3.9 Symbol, Boolean function, equivalent circuit and truth table of a (three-input) EXCLUSIVE-OR gate.

The functioning of a two input EXCLUSIVE-OR gate is similar to that of the circuit of fig. 3.10.

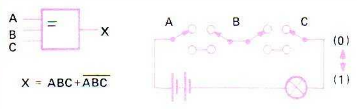


A	B	X
0	0	0
1	0	1
0	1	1
1	1	0

Fig. 3.10 A (two-input) EXCLUSIVE-OR gate regarded as the combination of two AND gates and an OR gate, and its truth table.

The **COMPARATOR (or logic identity gate)**

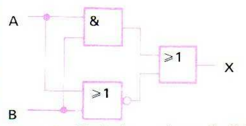
Definition: The output of the comparator will stand at its defined "**1**" state only if **all** of the inputs stand at their defined "**1**" states or if **none** of the inputs stands at their defined "**1**" states.



A	B	C	X
0	0	0	1
1	0	0	0
0	1	0	0
1	1	0	0
0	0	1	0
1	0	1	0
0	1	1	0
1	1	1	1

Fig. 3.11 Symbol, Boolean function, equivalent circuit and truth table of a (three-input) COMPARATION gate.

This gate can also be made from a combination of AND and OR gates (see fig. 3.12.).
The functioning of this gate is the complement of that of the circuit of fig. 3.10.



A	B	X
0	0	1
1	0	0
0	1	0
1	1	1

Fig. 3.12 A (two-input) COMPARATOR gate regarded as a combination of one AND and two OR gate, and its truth table.

Distributed connections

Definition: A distributed connection is a connection of the outputs of a number of elements that are joined together to achieve the effect of an AND/OR operation without the use of a special electronic component. Synonyms: Distributed AND, WIRED AND, phantom AND or DOT AND; Distributed OR, WIRED OR, phantom OR or DOT OR.

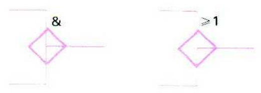


Fig. 3.13 Symbols for WIRED AND and WIRED OR connections.

The symbols for the WIRED AND and WIRED OR connection are shown in fig. 3.13.
See also chapter 7, page 51.

Delay

Definition: A delay element is a circuit in which each transition at the input causes a single delayed transition at the output.
The delay element is mostly used when a signal has to be held back for some time. It can be designed in such a way that the transition from the "**0**" state to the "**1**" state at the output occurs after a delay of e.g. t_1 sec. with respect to the same transition at the input and/or so that the transition from the "**1**" state to the "**0**" state at the output is delayed by t_2 sec. with respect to the same transition at the input.

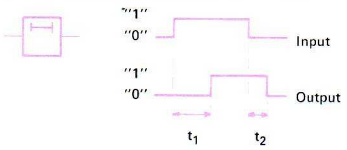
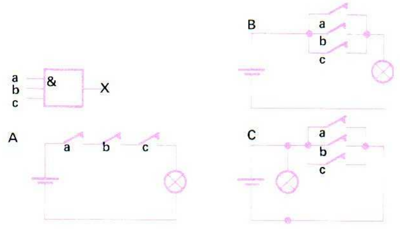


Fig. 3.14 Example of two delays.

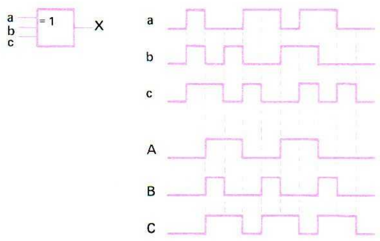
Questions

Q.3.1. Which of the circuits is represented by the logic diagram as shown here?



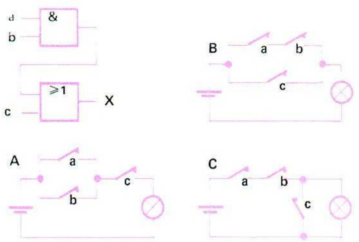
A	
B	
C	

Q.3.2. Which of the waveforms represent the output of the gate, if the input wave forms are a, b and c?



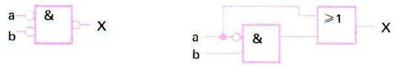
A	
B	
C	

Q.3.3. Which circuit is equivalent to the logic diagram as shown here?



A	
B	
C	

Q.3.4. The logic operations of the two circuits below are



A Complementary

B Identical

C Entirely different

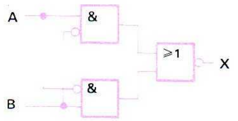
A	
B	
C	

Q.3.5. The circuit below represents a:

A Exclusive OR

B Comparator

C Inhibit gate



A	
B	
C	

Q.3.6. The Boolean expression for the above circuit is:

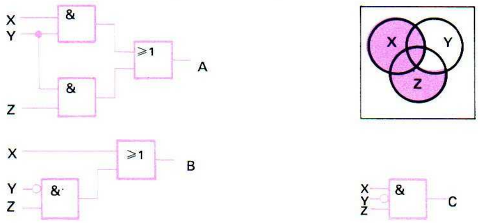
A $X = AB + \overline{A}\,\overline{B}$

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

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

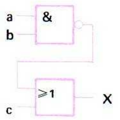
A	
B	
C	

Q.3.7. Which circuit represents the shaded area of the accompanying Venn diagram?



A	
B	
C	

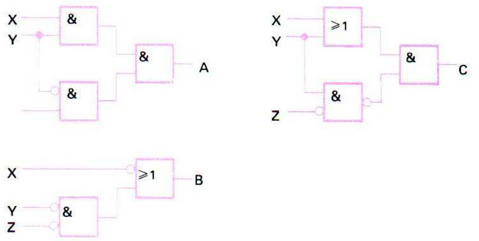
Q.3.8. Which of the truth tables below belongs to the logic diagram?



a	b	c	A	B	C
0	0	0	1	1	1
0	0	1	0	1	1
0	1	0	1	1	0
0	1	1	0	1	0
1	0	0	1	1	0
1	0	1	1	1	0
1	1	0	0	0	1
1	1	1	1	1	1

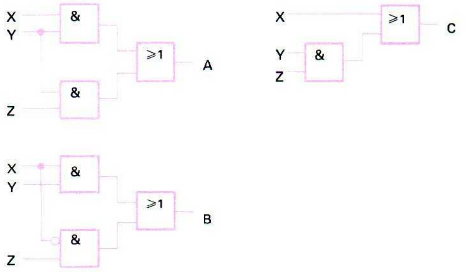
A	
B	
C	

Q.3.9. Which of the circuits A, B or C is the equivalent of the Boolean function $\Sigma\,(0,1,2,3,4)$?



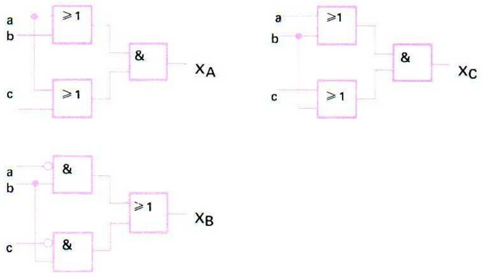
A	
B	
C	

Q.3.10 Which circuit is the logic equivalent of the following Boolean function $F\,(X,\,Y,\,Z) = (X+Y+XY)\,(X+Z)$?



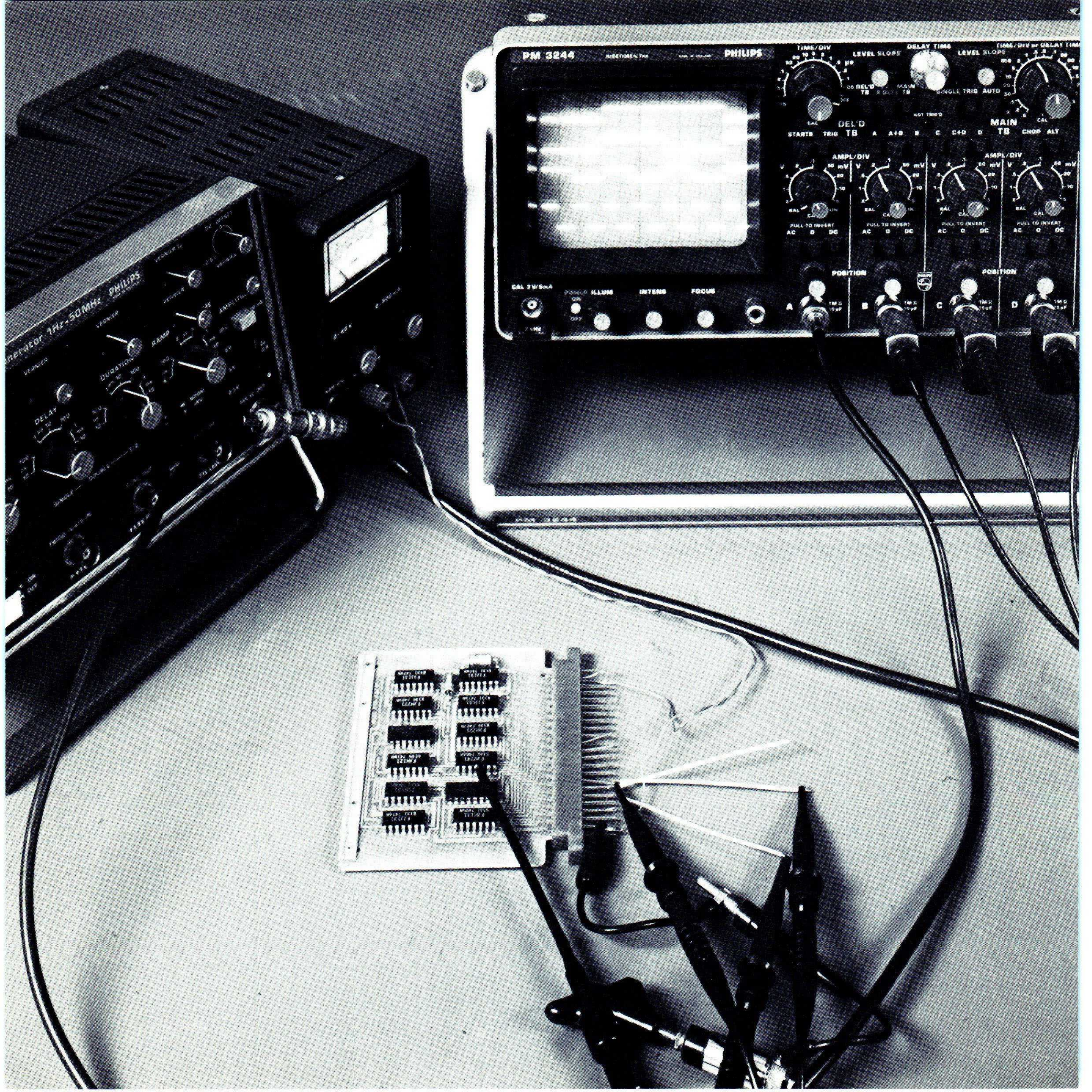
A	
B	
C	

Q.3.11. To which circuit does in truth table below belong?



a	b	c	x
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

A	
B	
C	



PM 3244
PHILIPS
TIME/DIV
LEVEL SLOPE
DELAY TIME
DEL'D
MAIN
TB
AMPL/DIV
PULL TO INVERT
AC O DC
POSITION
POWER
ILLUM
INTENS
FOCUS
A
B
C
D
Generator 1Hz-50MHz
PHILIPS
DELAY
DURATION
RAMP
VERNIER
AMPLITUDE

Chapter 4

Combinational logic

The logic elements in digital circuits can be used in a combinational or a sequential arrangement. In the former case, the output of the digital circuit depends only on the instantaneous value of the various input signals and not on the history of previous states, because the circuit does not contain a memory function.
Circuits whose operation is conditioned by their history, (i.e. where time is an operating parameter) are called sequential circuits. Their outputs are a function of the inputs and the state of the memory elements, which depends on previous inputs. Sequential circuits are discussed in Chapter 5 and following chapters.
Typical examples of combinational logic circuits are adders, subtractors, comparators, multiplexers, decoders, etc. These are discussed in the present chapter.

The half-adder
Arithmetic operations belong to the most important functions in digital computers.
The basic arithmetic operation is the addition of two binary digits (bits).
As we learned in chapter 1, there are 4 possible combinations, in binary addition, viz.
$0+0=0$
$0+1=1$
$1+0=1$
$1+1=1.0$,
where the last operation requires two digits (the sum bit, 0 and the carry bit "1"). These four addition operations can be realized with the half-adder, the truth table of which is given in fig. 4.1

X	Y	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Fig. 4.1. Truth table of half-adder

Inspection of this truth table shows S is the result of the EXCLUSIVE-OR operation on X and Y : $S=X\overline{Y}+\overline{X}Y$; and C is the AND function of X and Y: $C=X.Y$. These conclusions enable us to draw the logic diagram of the half-adder:

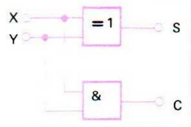


Fig. 4.2. Logic diagram of half-adder

Fig. 4.3. gives the logic diagram of half-adder made of NAND gates and fig. 4.4 the symbol used for the half-adder.

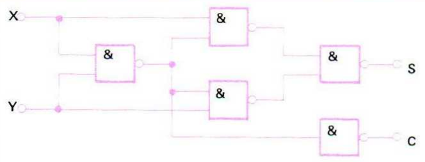


Fig. 4.3 Half-adder with NAND gates

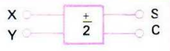


Fig. 4.4 Symbol of half-adder

However, the half-adder can add only two single bits; it cannot accept the carry from a lower position, which could be present when multiple-digit binary numbers are added. The "full-adder" has been designed to make such an addition possible.

The full-adder
This combinational circuit accepts three input bits: X and Y (the bits to be added) and C (the carry from the lower position). The Boolean function of a full-adder is:
$S = \overline{X}\overline{Y}Z + \overline{X}Y\overline{Z} + X\overline{Y}\overline{Z} + XYZ$ and
$C = XY + XZ + YZ$
The truth table of this logic function is given in fig. 4.5.

X	Y	C	S	C
0	0	0	0	0
1	0	0	1	0
0	1	0	1	0
1	1	0	0	1
0	0	1	1	0
1	0	1	0	1
0	1	1	0	1
1	1	1	1	1

Fig. 4.5 Truth table of full adder

This truth table shows clearly that a full adder can be built up from two half-adders: One for finding the sum of X and Y, and another for adding this sum to the input carry. The output carry is found by "ORing" the two carry outputs of the half-adders. The circuit diagram and symbol of a full-adder are given in fig. 4.6.

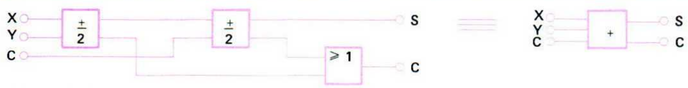


Fig. 4.6 Circuit diagram and symbol for full adder

An example of a complete 4 bit parallel-adder is given in fig. 4.7, which is self-explanatory.

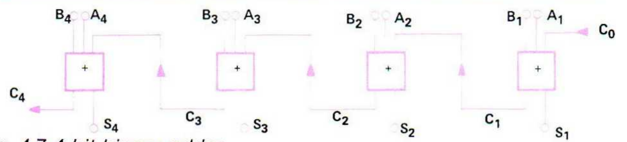


Fig. 4.7 4-bit binary-adder

The subtractor
As we have seen in chapter 1, (page 11), subtraction can be realized by adding the complement of the subtrahend to the minuend. Another way, however, is to subtract directly by means of logic elements in more or less the same way as with adders. There is, however, one important difference: when the minuend is smaller than the subtrahend, a "1" has to be borrowed from the next higher position. A "borrow bit" is used to inform the next higher bit pair about this loan.
Just as with the adders, we can design half-subtractors and full subtractors. The half subtractor is a two-bit subtractor, which offers the following possibilities:
$0-0=0$; $1-0=1$, $0-1=1+1$ borrow and $1-1=0$

Its truth table will thus have the following form:

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

Fig. 4.8 Truth table of half-subtractor

We can easily derive the Boolean functions for D (difference) and B (borrow) from this truth table:
$D = X\overline{Y} + \overline{X}Y$ and $B = \overline{X}Y$, and implement these functions with logic gates (fig. 4.9)

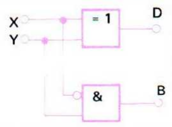
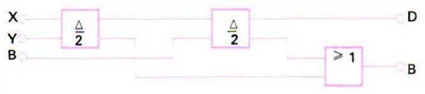


Fig. 4.9 Logic diagram of half-subtractor

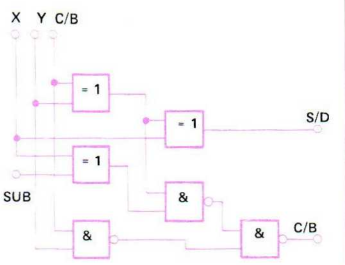
In analogy with the full adder we can also make a full subtractor from two half-subtractors and an OR-gate. The truth table and logic diagram of the full subtractor are given below:



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

Fig. 4.10 Truth table and logic diagram of full subtractor

The Boolean functions of the full subtractor are
$D = \overline{X}\overline{Y}Z + \overline{X}Y\overline{Z} + X\overline{Y}\overline{Z} + XYZ$ and
$B = \overline{X}Y + \overline{X}Z + YZ$
We see that the logic function for S in the full adder is exactly the same as for D in the full subtractor, and that the functions for carry and borrow are the same except for the X-input (the minuend) which appears inverted in the B function. In view of this similarity, it is an obvious idea to combine these two circuits. The only extra measure which has to be taken in case of subtraction is to invert the X input before it is fed to the carry/borrow gates. The logic diagram of such a combined circuit is given in fig. 4.11.
When the control input SUB is not true (logic "0") the circuit functions as a full adder, while when SUB = 1 it functions as a full subtractor. Here we see a very useful feature of the two-input exclusive OR. When one input (SUB in the present example) is held at logic "1", the output is always the complement of the signal at the other input. When the first input is held at "0" on the other hand, the output is not inverted.



			Addition SUB="0"		Sub-traction SUB="1"	
X	Y	C/B	S	C	D	B
0	0	0	0	0	0	0
1	0	0	1	0	1	0
0	1	0	1	0	1	1
1	1	0	0	1	0	0
0	0	1	1	0	1	1
1	0	1	0	1	0	0
0	1	1	0	1	0	1
1	1	1	1	1	1	1
Operation			X+Y+C		X−Y−B	

Fig. 4.11 Full adder/subtractor
SUB="0" for addition SUB="1" for subtraction
D=X−Y−B; S=X+Y+C

Comparators

Another very useful logic circuit is the comparator, a combinational circuit that compares two binary numbers X and Y. The simplest form of a binary comparator was discussed on page 26 (fig. 3.12). It merely indicates whether both numbers X and Y are equal (output "1") or not (output "0").
However, we may want to be able to distinguish between $X=Y$, $X>Y$ and $X<Y$. This can be done with the aid of a half-subtractor, by using the D and B outputs as indicators:

	D	B
X=Y	0	0
X>Y	1	0
X<Y	1	1

When both D and B are "0", $X=Y$; when only D is "1", $X>Y$; and when both D and B are "1", $X<Y$. In the above examples we compared 2 single-bit binary numbers. In practice, however, we generally want to compare multiple-bit binary numbers.

We could of course do this by using a half-subtractor for each pair of digits; in practice, however, special comparators for numbers with up to 4–5 digits are made by combining standard gates. We will discuss the example of a two-bit comparator here, as larger comparator are so complex they would take us beyond the scope of this booklet.

When we have two 2-bit numbers X_1X_0 and Y_1Y_0 we can derive the following three functions:
$X_1X_0 = Y_1Y_0$ output B is true
$X_1X_0 > Y_1Y_0$ output A is true
$X_1X_0 < Y_1Y_0$ output C is true
We thus have 4 input variables (X_1, X_0, Y_1 and Y_0) and three output variables (A, B and C).

The corresponding truth table is:

X_1	X_0	Y_1	Y_0	A	B	C	
0	0	0	0	0	1	0	X=Y
0	0	0	1	0	0	1	X<Y
0	0	1	0	0	0	1	X<Y
0	0	1	1	0	0	1	X<Y
0	1	0	0	1	0	0	X>Y
0	1	0	1	0	1	0	X=Y
0	1	1	0	0	0	1	X<Y
0	1	1	1	0	0	1	X<Y
1	0	0	0	1	0	0	X>Y
1	0	0	1	1	0	0	X>Y
1	0	1	0	0	1	0	X=Y
1	0	1	1	0	0	1	X<Y
1	1	0	0	1	0	0	X>Y
1	1	0	1	1	0	0	X>Y
1	1	1	0	1	0	0	X>Y
1	1	1	1	0	1	0	X=Y

The Boolean functions for A, B and C can be derived from the above truth table:

$$A = \overline{X}_1.X_0.\overline{Y}_1.\overline{Y}_0 + X_1.\overline{X}_0.\overline{Y}_1.\overline{Y}_0 + X_1\overline{X}_0\overline{Y}_1Y_0 + X_1X_0\overline{Y}_1\overline{Y}_0 +$$
$$+ X_1X_0\overline{Y}_1Y_0 + X_1X_0Y_1\overline{Y}_0 = X_1\overline{Y}_1 + X_1X_0\overline{Y}_0 + X_0\overline{Y}_1\overline{Y}_0$$
$$B = \overline{X}_1\overline{X}_0\overline{Y}_1\overline{Y}_0 + \overline{X}_1X_0\overline{Y}_1Y_0 + X_1\overline{X}_0Y_1\overline{Y}_0 + X_1X_0Y_1Y_0$$
$$C = \overline{X}_1\overline{X}_0\overline{Y}_1Y_0 + \overline{X}_1\overline{X}_0Y_1\overline{Y}_0 + \overline{X}_1\overline{X}_0Y_1Y_0 + \overline{X}_1X_0Y_1\overline{Y}_0 +$$
$$+ \overline{X}_1X_0Y_1Y_0 + X_1\overline{X}_0Y_1Y_0 = \overline{X}_1Y_1 + \overline{X}_1\overline{X}_0Y_0 + \overline{X}_0Y_1Y_0$$

These can be implemented with logic gates as shown in fig. 4.12:

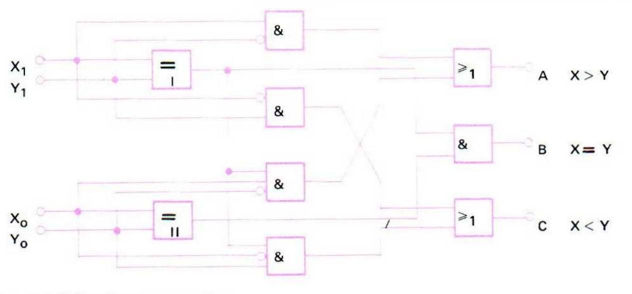


Fig. 4.12 2-bit comparator

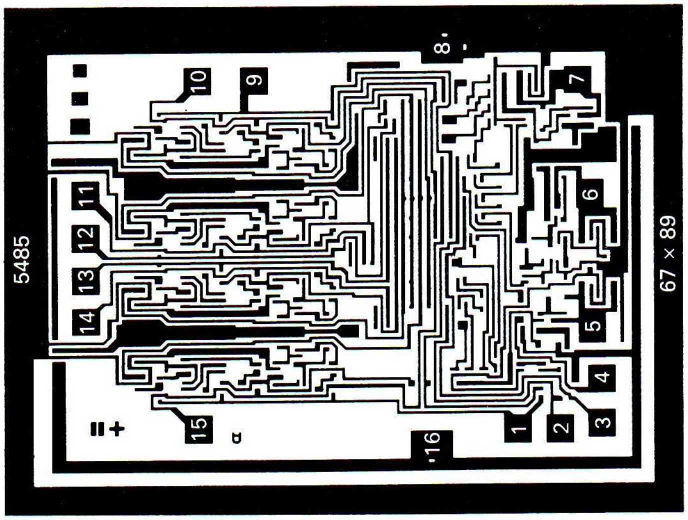
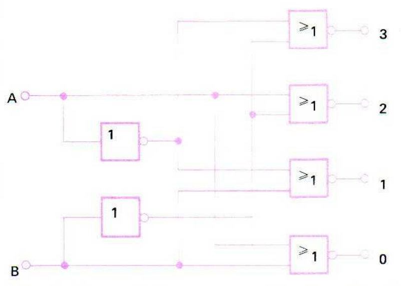


Fig. 4.13

Gates I and II in fig. 4.12 are standard comparator gates as described above (see. fig. 3.12) and are used for detecting equality of X_1X_0 and Y_1Y_0. A 4-bit cascadable comparator (standard IC) is depicted in fig. 4.13.

Decoders and encoders

A third family of useful combinational logic circuits comprises the decoders and encoders.
These two words are often used almost interchangeably in practice. Strictly speaking, however, an encoder is defined as a logic circuit that accepts any number of inputs and converts them into a binary code, while an decoder performs the reverse function. A simple example of a decoder is the 1-out-of-4 decoder (which converts a binary two-input signal into 4 output signals) is shown in fig. 4.14.



A	B	0	1	2	3
0	0	1	0	0	0
1	0	0	1	0	0
0	1	0	0	1	0
1	1	0	0	0	1

Fig. 4.14 Logic diagram and truth table of 1-out-of-4-decoder.

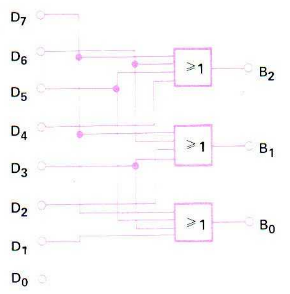
The Boolean functions for the 4 outputs variables are:

$$''0'' = \overline{A+B} \qquad ''2'' = \overline{\overline{A}+B}$$

$$''1'' = \overline{A+\overline{B}} \qquad ''3'' = \overline{\overline{A}+\overline{B}}$$

An example of an encoder is the octal-to-binary encoder shown in fig. 4.15, which realizes the following Boolean functions:

$$B_0 = D_1 + D_3 + D_5 + D_7$$
$$B_1 = D_2 + D_3 + D_6 + D_7$$
$$B_2 = D_4 + D_5 + D_6 + D_7$$

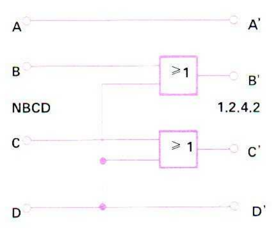


Octal digits								Binary digits		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	B_2	B_1	B_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

Fig. 4.15 Logic diagram and truth table of octal-to-binary encoder.

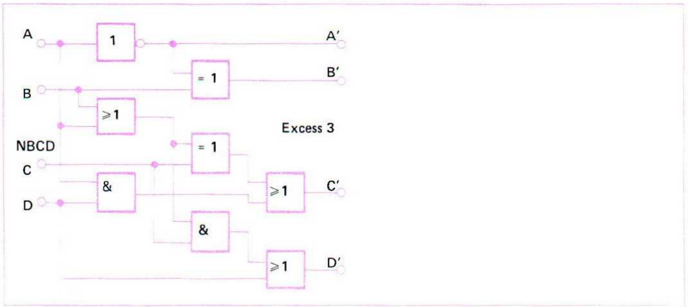
A very special class of code converters comprises the BCD-to-BCD converters, which convert from one BCD code into another. Two typical examples of such converters are given in fig. 4.16 and 4.17. Fig. 4.16 shows an NBCD (1,2,4,8)-to-1242 code converter, which realizes the Boolean functions:

$$A' = A$$
$$B' = B + D$$
$$C' = C + D$$
$$D' = D$$



	NBCD				1242			
	D	C	B	A	D'	C'	B'	A'
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	0
3	0	0	1	1	0	0	1	1
4	0	1	0	0	0	1	0	0
5	0	1	0	1	0	1	0	1
6	0	1	1	0	0	1	1	0
7	0	1	1	1	0	1	1	1
8	1	0	0	0	1	1	1	0
9	1	0	0	1	1	1	1	1

Fig. 4.16 Logic diagram and truth table of NBCD (1,2,4,8)-to-1,2,4,2 code converter.



Code	NBCD				Excess 3			
Dec.	D	C	B	A	D'	C'	B'	A'
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

Fig. 4.17 Logic diagram and truth table of NBCD-to-Excess-3 converter.

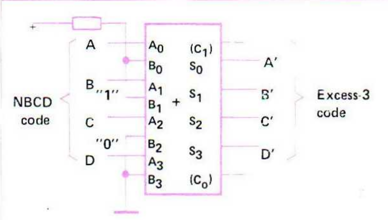
Here the following Boolean Functions apply:
NBCD word is DCBA
Excess-3 word is D′C′B′A′

$A' = \overline{A}$

$B' = A.B. + \overline{A}.\overline{B} = A \equiv B = \overline{A \oplus B}$

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

$D' = C.(A+B) + D$



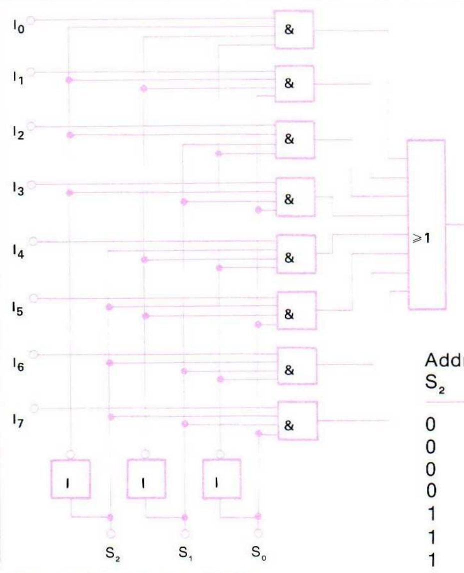
NBCD + 0011 = Excess-3

Fig. 4.18 NBCD-to-excess-3 converter using a 4-bit full-adder.

Fig. 4.17 depicts an NBCD-to-excess 3-code converter. As we have seen in chapter 1, the excess-3 code is derived from the NBCD code by adding binary 3 to each code, so we could have made this converter with the aid of a 4-bit full adder, as shown in fig. 4.18, which is self-explanatory.
Further examples of this important group of combinatorial circuits are given in chapter 8 (interfaces).

Multiplexers

The last group of combinational circuits we shall discuss here comprises the multiplexers and demultiplexers. The word multiplexing comes from telecommunication techniques, where a large number of voice channels are transmitted over a small number of lines. Similarly, the digital multiplexer can be used for transmission of a large number of digital signals over a few lines.
Another term frequently used for the digital multiplexer is "data selector". The reverse operation is called demultiplexing. In fact, a digital multiplexer is a combinational circuit that selects data from 2^n input lines (or group of lines) and transmits them through a single output line (or group of lines). Such a multiplexer closely resembles the encoders discussed above. The main difference is the presence of a number of additional input lines, the address or selector-lines.
An example of 8-input digital multiplexer is given in fig. 4.19.



Address			Selected
S_2	S_1	S_0	line
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

Fig. 4.19 8-input multiplexer

The 8-input lines are applied to 8 4-input and gates which are controlled by the address inputs S_0, S_1, S_2. Only one input at a time is connected to the output via the 8-input OR. In most cases the AND gates have a fifth input, the enable input. The enable inputs are then all connected in parallel to provide a master control which can be used to switch the output independently of the data or address inputs.

Fig. 4.20 shows an example of a multiplexer that selects one out of 8 data inputs, each input code consisting of 4-bits (e.g. a BCD code).

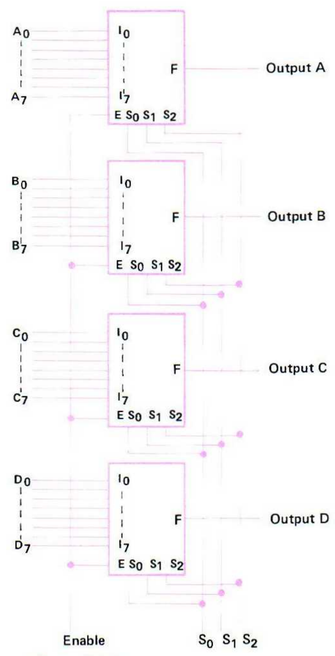


Fig. 4.20 8-input 4-bit multiplexer

The reverse of multiplexing is called demultiplexing. An example of a four-output digital demultiplexer is given in fig. 4.21, which is self-explanatory.

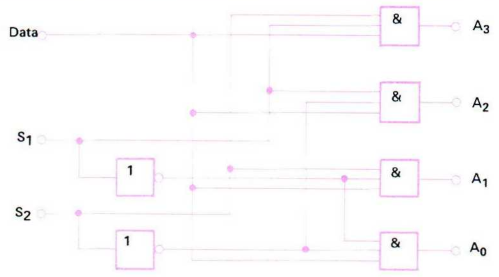


Fig. 4.21 Four-output digital demultiplexer.

Questions

Q.4.1. The 4-bit binary adder of fig. 4.7 is used to add two 4-bit numbers A and B. For this purpose the C_0-input should be kept at:

A	Logic "0"	A	
B	Logic "1"	B	
C	"1" in negative logic	C	

Q.4.2. The 4-bit adder of fig. 4.7 is used to perform a subtraction. For this purpose the number B is converted into its 2-'s complement. When now C_4 becomes logic "1", after the subtraction, this means that

A	The result is correct and positive	A	
B	1 has to be added to the result	B	
C	The result has to be complemented	C	

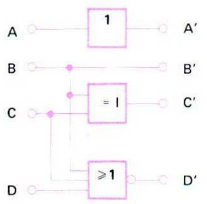
Q.4.3. A half-adder can be changed into a half-subtractor with the aid of:

A	An OR gate	A	
B	An inverter	B	
C	Cannot be done	C	

Q.4.4. The total number of connections in a three-input four-bit multiplexer is

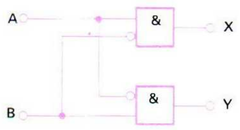
A	7	A	
B	12	B	
C	18	C	

Q.4.5. The circuit below is:



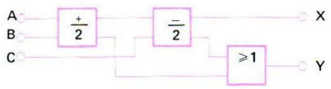
A	An NBCD (1248)-to-BCD (1242) converter	A	
B	An NBCD-to-9's-complement converter	B	
C	An octal-to-NBCD converter	C	

Q.4.6. The circuit below is:



A	A two-bit comparator	A	
B	A half-adder	B	
C	A two-bit multiplexer	C	

Q.4.7. A half-adder and a half-subtractor are connected as shown below:



The X-output is:

A	The same as the S output of the full adder	A	
B	The inverse of the D output of the full subtractor	B	
C	Always "0"	C	

Chapter 5

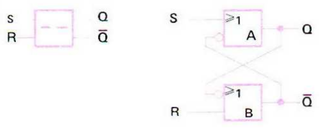
Bistable elements (flip-flops)

Elements whose operation is conditioned by their history, i.e. in which time is an operating parameter, are called sequential elements. The most important representative of this class is the flip-flop, which will be discussed in the present chapter.
Definition: A flip-flop is a bistable logic element with one or more inputs and two complementary outputs.
A flip-flop is essentially a bistable circuit that will remain in its last state until a specific input signal causes it to change state. Because of its ability to store bits of information in this way, the flip-flop became a basic building block in digital circuitry. The state of the flip-flop is available at one of the outputs, while the complement of the stored information is available at the second output. There are many forms of flip-flops, each of which has its specific features.

The RS flip-flop

However, all those various forms of flip-flops contain essentialy the same bistable element – the RS flip-flop. The RS flip-flop has two inputs called SET and RESET and two outputs called Q and $\overline{Q}$.
When the SET input receives a **pulse** (a **"1"**) and the RESET input **no pulse** (a **"0"**), output Q is a **sustained** voltage (a **"1"**), while output $\overline{Q}$ has **no voltage** (a **"0"**). This sustained voltage thus represents the binary **one** at the Q output.
Any additional pulse at the SET input will have no effect on the output. However, when a pulse is applied to the RESET input, the output reverses or "flips". Further pulses at the reset input have no effect on the outputs. Switching the inputs again causes the outputs to "flop" back to their original condition.

The flip-flop is like a toggle switch, either in one position or the other; and once the change-over has been made, repeating the action has no further effect. The condition is stable, either way. This type of flip-flop is called a bistable two-input flip-flop. It may consist of logic elements such as OR gates and invertors, with flow paths interconnected as shown in fig. 5.1.
To see how the flip-flop operates, let us begin with a **pulse** (a **"1"**) at the **set** input (S) and **no pulse** (a **"0"**) at the **reset** input (R).
The **"1"** input to OR gate A results in a **"1"** at the output Q. This **"1"** is applied to the invertor at one of the inputs of gate B, which changes it to **"0"**. The two **"0"**'s at the inputs of OR gate B result in a **"0"** at output $\overline{Q}$. In addition, this zero at output $\overline{Q}$ is applied to the invertor at one of the inputs of gate A where it is changed to a **"1"** that gives a second input to OR gate A. No change in the output of this gate results, because it already has a **"1"** output, due to the **"1"** set input. We thus see that a closed loop exists in the circuit, with a binary **"1"** in one half and a binary **"0"** in the other. The set-output voltage Q is therefore maintained even when the pulse at the set input has disappeared, as long as the reset input remains **"0"**.
When a **"1"** is applied to the reset input, the output of OR gate B is a **"1"** and the reset output $\overline{Q}$ is also **"1"**. This **"1"** is changed by the invertor at gate A to a **"0"**. With both inputs of gate A at **"0"** the set output Q is **"0"**. The invertor at gate B inverts this **"0"** and again the loop is closed: but now in the opposite direction maintaining the **"1"** at the reset output $\overline{Q}$. **"1"** at the set input will again cause the flip-flop action and reversal of outputs.
The output of the flip-flop thus "remembers" which input was the last one to receive a pulse. It may therefore be used for storage. As may be seen from the accompanying truth table, fig. 5.1., **"1"**'s at both inputs S and R are forbidden, since the flip-flop can never be in both states simultaneously.

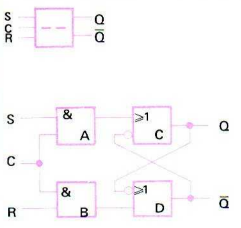


S	R	Q	$\overline{Q}$
1	0	1	0
↓			
0	0	1	0
	↓		
0	1	0	1
	↓		
0	0	0	1
↓			
1	0	1	0
	↓		
1	1	Undefined	

Fig. 5.1 Symbol, circuit diagram and truth table of the RS flip-flop.

Clocked RS flip-flop

It has been found in practice that it is often convenient to operate a flip-flop by applying the appropriate levels to the inputs while these are blocked, and then arranging for the flip-flop to change state on receipt of a pulse from another source (the "clock" pulse). The basic circuit for the clocked RS flip-flop is shown in fig. 5.2.



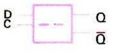
Initial condition		Input		After clock pulse	
Q	$\overline{Q}$	S	R	Q	$\overline{Q}$
0	1	0	0	0	1
0	1	0	1	0	1
0	1	1	0	1	0
0	1	1	1	Undefined	
1	0	0	0	1	0
1	0	0	1	0	1
1	0	1	0	1	0
1	0	1	1	Undefined	

Fig. 5.2 Symbol, circuit diagram and truth table of the clocked RS flip-flop.

If the SET input S is made logic "**1**", the output of gate A will become "**1**" after the clock line C has received a "**1**". The "**1**" at A is applied to the SET input (gate C) of the RS flip-flop proper, causing a "**1**" to be produced at the Q output.
If the SET input S is at "**0**" and the RESET line (R) is enabled with a "**1**", a "**1**" will be produced at the output of gate B on receipt of the clock pulse. This "**1**" is applied to the input of gate D, resetting the RS flip-flop.
As the truth-table in fig. 5.2. shows, the flip-flop will only change state when a clock pulse is applied.

D flip-flop

A way of avoiding the indeterminate state found in the operation of the simple RS flip-flop is to provide only one input (the D input). A "**1**" or a "**0**" applied to this input is passed directly to one of the inputs of the flip-flop proper, and inverted to the other input. The circuit diagram of this D flip-flop is shown in fig. 5.3.



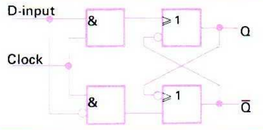
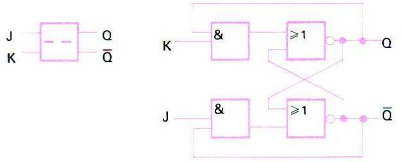


Fig. 5.3 Symbol and circuit diagram of the D flip-flop.

Whatever information is present at the D input prior to and during the clock pulse is propagated to the Q output when the clock pulse is applied, while the inverse of that information appears at the $\overline{Q}$ output. The flip-flop is thus set in the "**1**" state if the D input is made "**1**", and in the "**0**" state if the D input is made "**0**".

The JK flip-flop

A further refinement of the RS flip-flop is the JK flip-flop, which has become one of the most popular members of the flip-flop family. A unique feature of the JK flip-flop is that it has no ambiguous state.



J	K	Q	$\overline{Q}$
0	0	No change	
1	0	1	0
0	1	0	1
1	1	Complement	

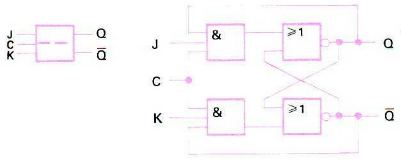
Fig. 5.4 Symbol diagram and truth table of the JK flip-flop

Like the RS flip-flop, the JK flip-flop has two inputs and two outputs. When a "**1**" is applied to the J-input, Q also becomes "**1**" (and $\overline{Q}$→ "**0**"), while a "**1**" is applied to the K-input causes the circuit to flop back to its initial position ($\overline{Q}$→"**1**", Q→"**0**"). When a "**1**" is applied to both J and K inputs simultaneously, both outputs switch over. A JK flip-flop often has several J and K inputs.

The clocked JK flip-flop

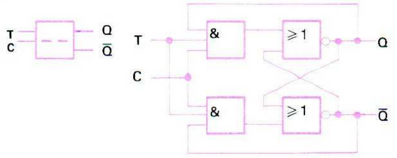
In this case, one J and K are tied together to form the clock input C. This kind of configuration is shown in the logic diagram of fig. 5.5.
The unique features of this kind of flip-flop are:

- A clock pulse will not cause any changes in the state of the flip-flop if neither J nor K input are activated.
- If both the J and K inputs are "**1**", the flip-flop will change state when the next clock pulse is received.
- The J and C input used together will set the flip-flop; K and C used together reset it.



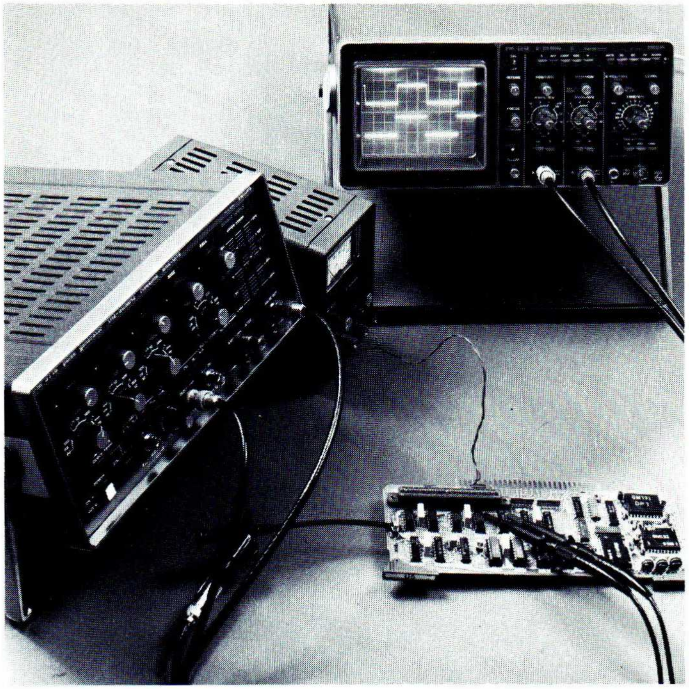
Q	J	K	Q(t+1)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Fig. 5.5 Symbol, logic diagram and truth table of clocked JK flip-flop



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

Fig. 5.6 Logic diagram, symbol and truth table of clocked T flip-flop.



The T-flip-flop (toggle flip-flop)
The T-flip-flop is a single input version of the JK flip-flop. As shown in fig. 5.6, the T flip-flop is derived from a JK type by typing one pair of J and K inputs together.
Just as with the other flip-flops, there are unclocked and clocked T flip-flops. The latter is illustrated in fig. 5.6. As can be seen from the truth table, the circuit toggles from **"0"** tot **"1"** after receipt of one pair of T and C pulses, and back again after receipt of the next T-C pair. The T flip-flop is usually represented in circuit diagrams by the symbol shown above. The set output can be used to count pulses. Since two input pulses are required for each **"1"**, from the set output, the deviced may be said to devide by two (i.e. it is a binary scaler) or to count in two's (i.e. it is a binary counter). A disadvantage of both the T flip-flop and the JK flip-flop is that when the inputs remain **"1"** after the outputs have been complemented the flip-flop will switch over again. This timing problem is eliminated in the master-slave flip-flop described below.

Master-slave flip-flops
A special form of the JK flip-flop is the ''JK master-slave flip-flop'' see fig. 5.7. The information present at the J and K inputs enters the master when the T input is **"1"**. When the T input is made **"0"** again, the information is transferred from the master to the slave and then appears at the outputs.
This can be seen as follows. Gates A and B form the master flip-flop, and gates C and D the slave. The Q output of the master flip-flop is the output of gate A. Suppose J and K inputs are both **"1"**, while the Q output of the slave flip-flop is also **"1"**. When the clock input is made **"1"** the lower input gate will be enabled and will reset the master flip-flop. The slave flip-flop remains in the **"1"** state as long as the clock pulse is **"1"**, because the invertors at the input gates of the slave flip-flop are blocking pulse transfer to this flip-flop for the moment. When the clock input goes to **"0"** the lower input gate of the slave flip-flop wil be enabled so that this flip-flop can now be reset and so on. D- and T-type flip-flops can also be realised in a master-slave version.

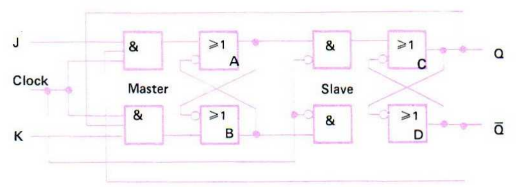
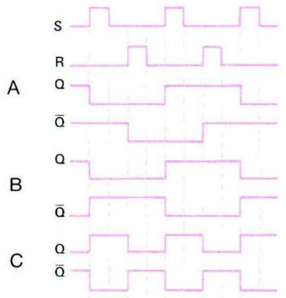


Fig. 5.7 The JK master-slave flip-flop.

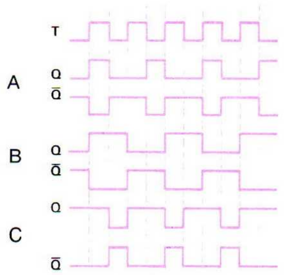
Questions:

Q.5.1. If a logic "1" is represented by a **pulse** and a logical "0" by **no pulse**, which of the sets of waveforms shown represents the behaviour of the RS flip-flop?



A	
B	
C	

Q.5.2. If a single-input T flip-flop has an input signal T, which of the sets of waveforms gives the output?



A	
B	
C	

Q.5.3. Construction of a SET/RESET flip-flop with AND gates requires:

A 2 AND gates

B 4 AND gates

C Cannot be done

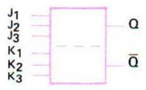
A	
B	
C	

Q.5.4. If one wishes to change a 3-input JK flip-flop into a single-input T flip-flop, should one:

A Connect all J and K inputs together, to give the T input?

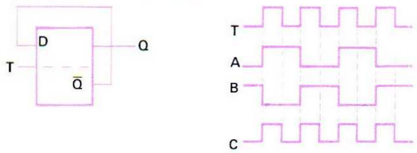
B Connect all J inputs together to give the T input and all K inputs to ground (logic "0")?

C Give up the idea, because it can't be done?



A	
B	
C	

Q.5.5. A D-type flip-flop is connected as shown below, initially with Q = "0" and Q̄ = "1".
A pulse is fed to the T input. Which waveform represents the signal at the Q output?



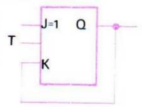
A	
B	
C	

Q.5.6. A JK flip-flop has its J input connected to logic "1" and its K input to the Q output. A clock pulse is fed to the T input. The flip-flop will now

A Change state at each clock pulse

B Go to "1" and stay there

C Go to "0" and stay there



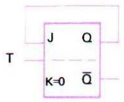
A	
B	
C	

Q.5.7. A JK flip-flop is connected as shown below. After a clock pulse is applied the Q output is "1". What was the Q output **before** the past clock pulse?

A "0"

B "1"

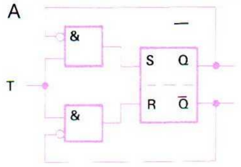
C Impossible



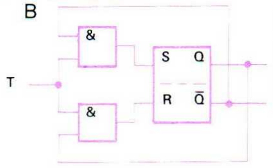
A	
B	
C	

G.5.8. The correct circuit of a T flip-flop is

A



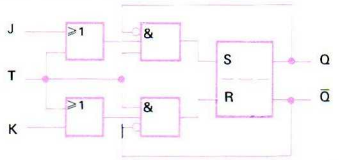
B



C Can be both

A	
B	
C	

Q.5.9. The circuit below:



A Will work as a JK flip-flop

B Will **not** work as a JK flip-flop

C Will work as a JK flip-flop when the Q and Q̄ connections are interchanged

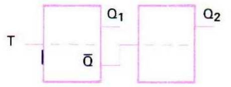
A	
B	
C	

Q.5.10. Two T flip-flops are connected in series as shown. After both flip-flops have been reset ($Q_1 = Q_2 =$ "0") clock pulses are fed to T. After two clock pulses the Q_2 output will be:

A "0"

B "1"

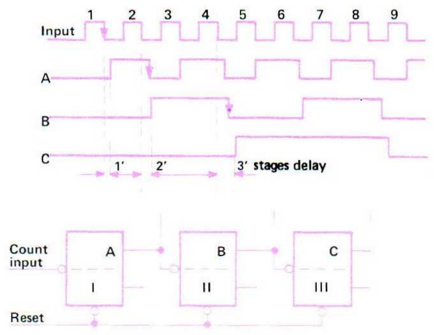
C Can be either



A	
B	
C	

Chapter 6

Counters, scalers and shift registers



Input	C	B	A
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1
8=0	0	0	0

Fig. 6.1 Logic/pulse diagram and truth table of an asynchronous binary up counter.

Binary counters

Up counters
When a number of flip-flops are connected in series we get a binary up (or forward) counter. In fig. 6.1 we see three flip-flops combined to give a three-bit counter, which can count to decimal 8. We can divide such counters into two subclasses:
a) asynchronous counters and
b) synchronous counters.

Asynchronous counters
With this type of counter the output of each flip-flop is connected to the input of the next. The operation can be explained as follows, with reference to fig. 6.1.
We assume that all outputs A, B and C are initially cleared to "0" (by a reset pulse). As can be seen from the pulse diagram in the figure, as soon as the first input pulse becomes "0" (the circle at the input of the T flip-flop indicates that the flip-flop toggles at the negative-going edge of the pulse) the A output of FF_I becomes "1". The circuit has now counted the first pulse: "001". The T input of FF_{II} is also "1" now. When the second pulse has passed, T_1 goes to "0", FF_I switches over and A becomes "0" so that a negative-going edge is presented to T_2 and FF_{II} goes to the "1" state while T_3 also becomes "1". The circuit has now counted the second pulse: "010". After the third pulse FF_I switches over again and A becomes "1", but nothing further happens the circuit has now counted three: 011. After the fourth pulse A and B become "0" and C becomes "1": 100 in binary notation.
The counting continues in this fashion until the circuit's output is "111" which as we know is 7 in binary notation. The eighth count resets the counter to 000 and gives a "carry" (a negative-going edge) to the next flip-flop, if there is one.

We thus see that the count pulses "ripple through" the binary chain, in such a way that e.g. the last binary stage cannot change state until all the preceeding stages have done so. Since each flip-flop takes some time to change state there will be a delay in the output signals with respect to the input pulses, so the output is not synchronous with the input. This can be seen very clearly from the pulse diagram of fig. 6.1.

Synchronous counters
The delay which, as we have seen, is an inherent property of ripple-through counters can be avoided by using synchronous counters. In this type of counters the input pulses are fed simultaneously to all T inputs of the various flip-flops in the chain, whereas the output pulses are used to condition subsequent flip-flops. In this way all stages required to change state on receipt of a particular pulse will do so simultaneously.
The circuit can be explained as follows (fig. 6.2.):
We again assume that all flip-flops are initially reset to 0, so outputs A, B and C are also "0". As can be seen from the pulse diagram, as soon as the first input pulse has become "0" again the output A (first flip-flop) becomes "1". Although the input signal is applied simultaneously to the other two flip-flops as well nothing happens to these flip-flops because their J and K inputs are both "0". The circuit has now counted the first pulse: 001. After this first pulse, however, the J and K inputs of the second flip-flop are "1" (as is A); after the next input pulse, flip-flop A **and** flip-flop B will thus toggle, A becoming "0" again, and B becoming "1". The second pulse has thus been counted "010", while nothing happens to the third flip-flop C because the J and K inputs are still 0.

($J_C = AB = 1.0 = 0$, similarly for K_C). Since A is "0" after the second pulse, J_B and $K_B = 0$, so the next (third) clock pulse will not cause flip-flop B to switch over. The first flip-flop (which has no conditioning inputs) will however switch over again on receipt of the third pulse making A and B both "1". Flip-flop C is now conditioned to change state with the next (4th) clock pulse, and as J_B and K_B are also "1" **all** flip-flops change state and the count is "100" (binary 4); the whole cycle described above is now repeated until $J_C = K_C = 1$ again, when all flip-flops will be reset to zero (8th count).

Down counters

In the counters discussed above each count increases the stored binary value by one bit. In binary down (or reverse) counters, on the other hand, each incoming pulse reduces the stored binary value by one bit. (Starting from 110 we get 101, 100, 011 and so on.)
This type of counter can be also made asynchronous or synchronous.

Asynchronous down counters

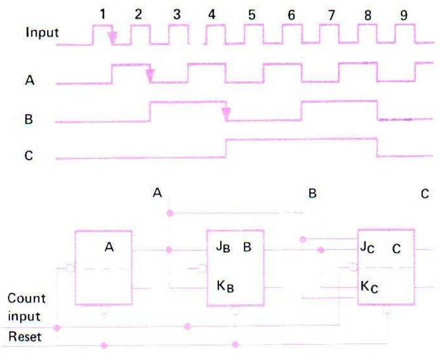
The asynchronous down counter differs from the corresponding up counter in that the $\overline{Q}$ outputs ($\overline{A}$, $\overline{B}$ and $\overline{C}$) are connected to the T inputs of the following stages, instead of the Q outputs (A, B and C).
This counter works in more or less the same way as the asynchronous up counter but for the sequence of the binary values.
We assume (for a 3-bit counter) that the three flip-flops all initially are reset to zero, which means in this case that the stored value is 8. After the first clock pulse, flip-flops I, II and III with all toggle because all their T inputs become LOW ("0"), resulting in a count of 7 ("111"). After the next pulse only flip-flop I will switch over; the count is now 6 (110). This process continues until the value is again "000".

Synchronous down counters

The basic difference between the synchronous down counter and the corresponding up counter is again that the $\overline{Q}$ outputs of the various flip-flops are used for passing on the information, instead of the Q outputs. The circuit is shown in fig. 6.4, which is self-explanatory.

Reversible counters

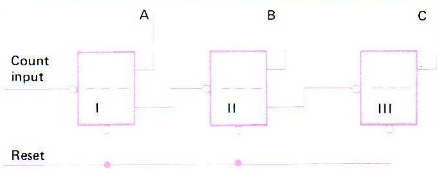
As we have seen above the only difference between up and down counters is the use of either the Q or $\overline{Q}$ output of the various flip-flops for passing on the information. It follows that inclusion of a selector switch in the circuit to choose between these two complementary sets of outputs will give a counter which can be made to count up of down at will.



				Conditioning inputs after clock pulse			
Input	C	B	A	J_B	K_B	J_C	K_C
0	0	0	0	0	0	0	0
1	0	0	1	1	1	0	0
2	0	1	0	0	0	0	0
3	0	1	1	1	1	1	1
4	1	0	0	0	0	0	0
5	1	0	1	1	1	0	0
6	1	1	0	0	0	0	0
7	1	1	1	1	1	1	1
8=0	0	0	0	0	0	0	0

Note: $J_B = A$ $J_C = A.B$
$K_B = A$ $K_C = A.B$

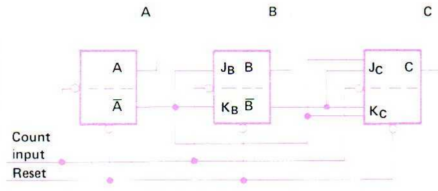
Fig. 6.2 Logic/pulse diagram and truth table of a synchronous binary up counter.



6.3.

Input	C	B	A
8=0	0	0	0
7	1	1	1
6	1	1	0
5	1	0	1
4	1	0	0
3	0	1	1
2	0	1	0
1	0	0	1
0	0	0	0

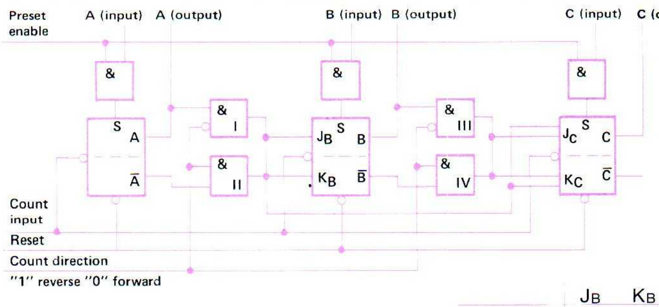
Fig. 6.3 Logic diagram and truth table of an asynchronous binary down counter.



				Conditioning inputs after clock pulse			
Input	C	B	A	J_B	K_B	J_C	K_C
8=0	0	0	0	1	1	1	1
7	1	1	1	0	0	0	0
6	1	1	0	1	1	0	0
5	1	0	1	0	0	0	0
4	1	0	0	1	1	1	1
3	0	1	1	0	0	0	0
2	0	1	0	1	1	0	0
1	0	0	1	0	0	0	0
0	0	0	0	1	1	1	1

Note: $J_B = \overline{A}$ $J_C = \overline{A}.\overline{B}$
$K_B = \overline{A}$ $K_C = \overline{A}.\overline{B}$

Fig. 6.4 Logic diagram and truth table of a synchronous binary down counter.



	J_B	K_B	J_C	K_C
Forward	A	A	A.B	A.B
Reverse	$\overline{A}$	$\overline{A}$	$\overline{A}.\overline{B}$	$\overline{A}.\overline{B}$

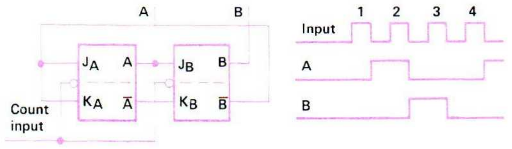
Fig. 6.5 Reversible synchronous binary counter with pre-set inputs.

In practice of course, logic gates are used instead of switches for this purpose. An example of such a reversible binary counter is given in fig. 6.5. When we apply a "0" to the count-direction terminal, the AND gates I and III are enabled and the Q outputs of the first two flip-flops (A and B) are coupled to the J and K inputs of the following stages. This gives a forward counter.
When a "1" is applied to the count-direction terminal, on the other hand, AND gates II and IV are enabled and we have a down counter.

Presettable counters
Very often we want to start counting from a specific value, e.g. a down counter is often set to a predetermined number and then allowed to count down towards zero. Arrival at zero marks the end of the count, and is often used to trigger a related process. The initial value could be set by putting the counter in the forward mode and counting up until the required value is reached, after which the counter is reversed for the count down; but this is rather a complicated procedure, especially as a decoding circuit is required to indicate when the required number has been reached.
A more practical solution is the presettable counter (fig. 6.5) which works as follows:
When a "1" is applied to the set input of a JK flip-flop, the flip-flop will switch to the "1" position ($Q = 1$) – independent of the previous state. The counter of fig. 6.5 can thus be preset to any desired value by clearing it, applying "1"s to the appropriate inputs, and enabling the upper AND gates.

Decade and other counters
We have seen above how a simple chain of flip-flops gives a binary counter, counting in powers of two (2, 4, 8, 16 and so on). For counters in any other number system we need a special circuit design, often making use of the specific properties of the J and K inputs of the JK flip-flop.
A modulo-n counter requires at least N flip-flops, where N is given by the relationship: $2^{N-1} < n < 2^N$
For example, a modulo-3 counter will require two flip-flops, since: $2^1 < 3 < 2^2$.

Modulo-3 counters
Simple connection of two flip-flops gives a modulo-4 counter, which is reset to zero at the count of $2^N = 4$. However, the circuit of fig. 6.6. is designed so that it will be reset to zero at the count of 3, as appears from the following description of its operation.
On receipt of the first pulse, flip-flop I switches because J_A and K_A are both "1" ($\overline{B}$ is "1"); flip-flop II however remains in the "0" state because $J_B = 0$ ($= A$) and $K_B =$ "1" ($= \overline{A}$). On receipt of the next pulse, I switches again (because $\overline{B}$ was still "1"), and II also switches (to the "1" state) because $J_B = A =$ "1"; the count is thus "10". The third pulse leaves flip-flop I unchanged because J_A and K_A are "0", while flip-flop II returns to the "0" state because $K_B = 1$ and $J_B =$ "0"; the counter is thus reset. The illegitimate state "11" is thus avoided and the cycle can start again after the count of 3.



Input	B	A
0	0	0
1	0	1
1	1	0
3=0	0	0

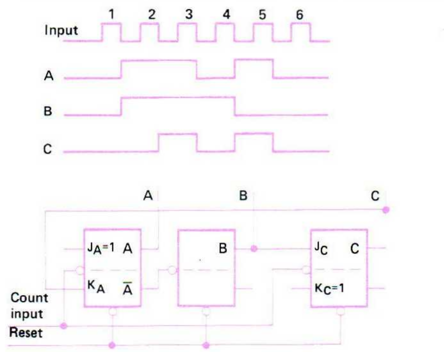
Fig. 6.6 Logic/pulse diagram and truth table of a synchronous modulo-3 counter.

Note: $J_A = \overline{B}$, $K_A = \overline{B}$, $J_B = A$, $K_B = \overline{A}$

Modulo-5 counters
Basically, the modulo-5 counter is just a modulo-3 counter with one more flip-flop. However, the more flip-flops there are in the counter, the more possible ways there are of arranging the various signal paths to give the desired result; in fact, there are more than 10 ways of making a modulo-5 counter with 3 JK flip-flops.
An example of such a counter is given in fig. 6.7. We have chosen this specific circuit not because it is a good one, but precisely because of two serious disadvantages it has. The discussion of these drawbacks given below will teach us a lot about the design of these counters.
First we will have a look at the functioning of the circuit. The first pulse (after reset) switches flip-flop I to "1" (because $J_A = 1$, fixed), and after a slight delay flip-flop II switches over too (because $\overline{A}$ goes from "1" to "0"); nothing happens to flip-flop III ($J_C = 0$, $K_C = 1$). After the second clock pulse flip-flop III changes state ($J_C = 1$), but not flip-flop I (because $K_A = 0$); consequently, flip-flop II also remains in its old state. Counting continues as indicated in the pulse diagram and truth table until the counter is reset to "000" on receipt of the 5th pulse.

Inspection of the truth table shows what can be regarded as one of the disadvantages of this modulo-5 counter, namely that the code used is not weighted like the normal binary code; indeed, there is no recognizable system in the coding.
However this illogical coding can be overcome by proper decoding circuits. A much greater disadvantage is the situation concerning the illegitimate states.
As we know, a modulo-5 counter has $8-5=3$ illegitimate states (code combinations we do not want to use). This does not matter in normal operations since as we can see from the truth table these illegitimate states do not occur at all. The problem arises when something out of the ordinary happens.
Let us suppose that because of some outside interference the counter comes into position "001". This is illegitimate combination I in the truth table. (Such a combination can also appear when an instrument is switched on.)
Now when a clock pulse arrives with the counter in this position, flip-flop I will not switch over (because it is already in the "1" position); consequently flip-flop II will not switch over either, nor will flip-flop III ($J=0$, $K=1$). In other words the counter will be locked in this position until a reset pulse is given.
Similarly: if a clock pulse arrives when the counter is in illegitimate state III the counter switches to position I and remains there for the reasons described above. (In the case of illegitimate state II, the counter will go to decimal state 1 after the clock pulse and is thus in the normal sequence again.)
Any design of a modulo-n counter must thus involve careful investigation in order to avoid any locked illegitimate code combinations.

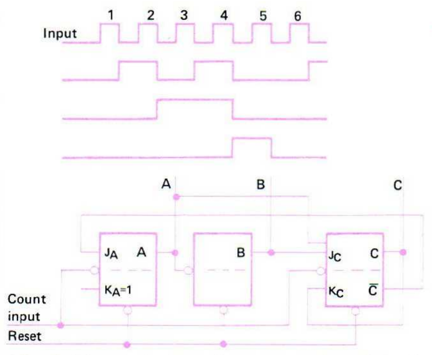
Fig. 6.8. shows a modulo-5 counter which does not possess the two above-mentioned disadvantages.
The logic diagram as given in fig. 6.8. together with the pulse diagram and the truth table are self-explanatory. As we see from the truth table the binary code is now true binary weighted (1, 2, 4 code). The special gating arrangement ensures that the counter is reset after the 5th clock pulse ($J_A=0$, so flip-flop I remains in the "0" state as does flip-flop II, whereas flip-flop III switches back to the "0" state because $J_C=0$ and $K_C=1$).
Naturally this modulo-5 counter also has 3 illegitimate binary code combinations, which could arise accidently. However, the next clock pulse changes the illegitimate states into normal legitimate combinations (I→ (2), II→ (2), and III→(0)) so no locking can occur.
Similar considerations apply to the design of any other type of modulo-n counters.



Input	C	B	A	K_A	J_C
0	0	0	0	0	0
1	0	1	1	0	1
2	1	1	1	1	1
3	0	1	0	0	1
4	1	0	1	1	0
5=0	0	0	0	0	0
Illegitimate states					
I	0	0	1	0	0 → [I]
II	1	0	0	1	0 → [1]
III	1	1	0	1	1 → [I]

Note: $J_A=1$, $K_A=C$,
$J_C=B$, $K_C=1$

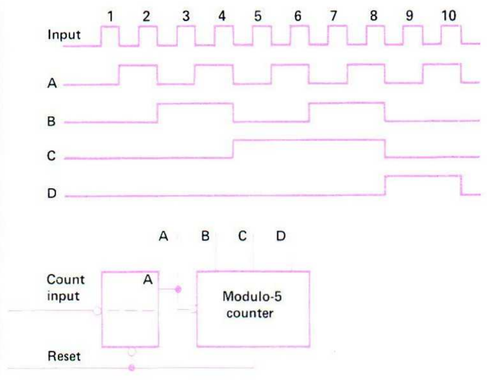
Fig. 6.7 Logic/pulse diagram and truth table of modulo-5 counter.



Input	C	B	A	J_A	K_C	J_C
0	0	0	0	1	0	0
1	0	0	1	1	0	0
2	0	1	0	1	0	0
3	0	1	1	1	0	1
4	1	0	0	0	1	0
5=0	0	0	0	1	0	0
Illegitimate states						
I	1	0	1	0	1	0 → [2]
II	1	1	0	0	1	0 → [2]
III	1	1	1	0	1	1 → [0]

Note:
$J_A=\overline{C}$, $K_A=1$
$J_C=AB$, $K_C=C$

Fig. 6.8 Logic/pulse diagram and truth tables of an improved modulo-5 counter.

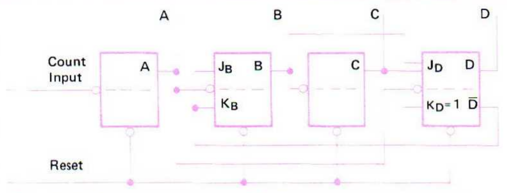


Input	D	C	B	A
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10=0	0	0	0	0

Input	D	C	B	A
0	0	0	0	
2	0	0	1	
4	0	1	0	0
6	0	1	1	
8	1	0	0	
1	0	0	0	
3	0	0	1	
5	0	1	0	1
7	0	1	1	
9	1	0	0	

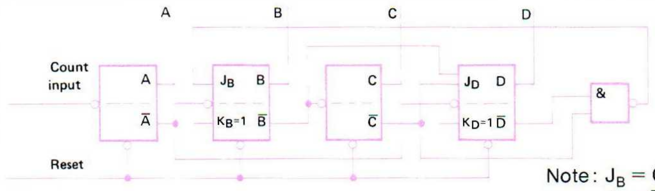
Note: $J_B=\overline{D}$, $K_B=\overline{D}$,
$J_D=BC$, $K_D=1$

Fig. 6.9 Logic/pulse diagram and truth trables of decade counter.



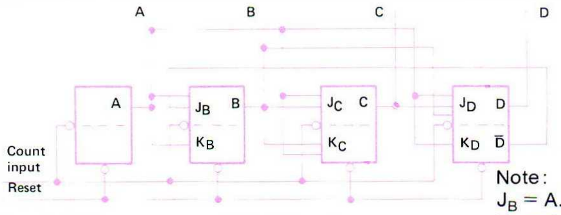
Note: $J_B = \overline{D}$, $K_B = \overline{D}$,
$J_D = BC$, $K_D = 1$

Fig. 6.10 Asynchronous NBCD decade up counter.



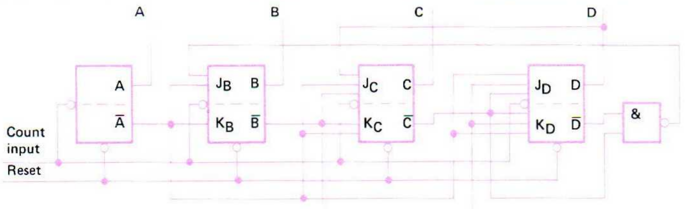
Note: $J_B = C+D$ $K_B = 1$
$J_D = \overline{B}.\overline{C}$ $K_D = 1$

Fig. 6.11 Asynchronous NBCD decade down counter.



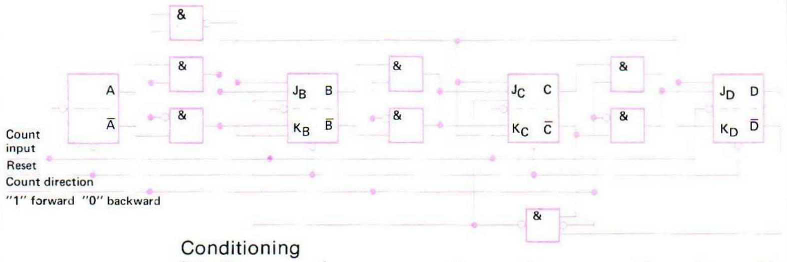
Note:
$J_B = A.\overline{D}$ $K_B = A$ $J_C = A.B$
$K_C = A.B$ $J_D = A.B.C$ $K_D = A$

Fig. 6.12 Synchronous NBCD decade up counter.



Note: $J_B = \overline{A}(C+D)$ $K_B = \overline{A}$ $J_C = \overline{A}.\overline{B}.D$
$K_C = \overline{A}.\overline{B}$ $J_D = \overline{A}.\overline{B}.\overline{C}$ $K_D = A.B.C$

Fig. 6.13 Synchronous NBCD down counter.



Conditioning inputs	J_B	K_B	J_C	K_C	J_D	K_D
Forward	$A\overline{D}$	$A.\overline{D}$	$A.B$	$A.B$	$A.C$	A
Reverse	$\overline{A}$ $(C+D)$	$\overline{A}$	$\overline{A}.\overline{B}$ $(C+D)$	$\overline{A}.\overline{B}$	$\overline{A}.\overline{C}$	$\overline{A}$

Fig. 6.14 Reversible synchronous NBCD counter.

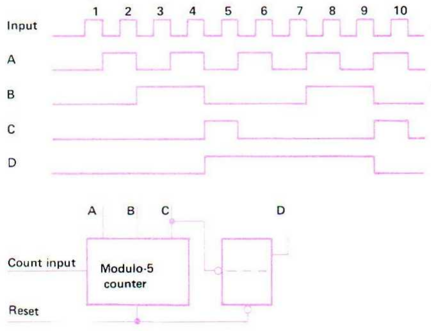
Decade counters

One of the most important counter types in measuring instruments is the decimal counter, since we still think and work in our decimal system. A decimal counter requires 4 flip-flops ($2^3 < 10 < 2^4$), which gives us 6 illegitimate code combinations.

A decade counter can simply be made by putting a binary scaler (flip-flop) in front of the modulo-5 counter as shown in fig. 6.9. Such a decade counter operates along the same lines as the modulo-5 counter of fig. 6.8, except that only every second input pulse is counted by the modulo-5 counter. This is well illustrated by the modified truth table of fig. 6.9, where the two identical truth tables of the modulo-5 counter can be clearly seen. Like the modulo-5 counter of fig. 6.8, the decade counter also has true binary weighting, and there is no danger of locking in illegitimate code combinations.

Like the pure binary counter, the decimal (and other modulo-n) counters can be made synchronous or asynchronous, reversible and presettable, and can count up or down. These various possibilities are illustrated in fig. 6.10 to fig. 6.14.

As we learned in chapter 1 there are quite a number of possible BCD codes. In the above example we have worked with the normal BCD (NBCD) code (1, 2, 4, 8). However, proper gating arrangements can of course give any other BCD code. For example, if we put the binary scaler in the decade counter of fig. 6.9 behind the modulo-5 counter instead of in front of it, we get a decade counter with the weighting 1, 2, 4, 5, (fig. 6.15.).



Input	D (= 5)	C (= 4)	B (= 2)	A (= 1)
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	1	0	0	0
6	1	0	0	1
7	1	0	1	0
8	1	0	1	1
9	1	1	0	0
10=0	0	0	0	0

Fig. 6.15 Logic/pulse diagram and truth table of decade counter with 1, 2, 4, 5 weighting.

Scalers

The counters decribed in this chapter have another useful property: they can also be used as scalers or dividers. For example, inspection of the pulse diagram of fig. 6.7 shows that the B output goes from "0" to "1" (or from "1" to "0") once in every 5 counts. In other words we have 1 output pulse at output B after each 5 input pulses. The counter has thus divided the number of pulses by 5: the modulo-5 counter is also a 5 divider or "quinary scaler". This holds for all modulo-n counters: they are also n-scalers. One can nearly always find an output which only changes state after n input pulses.
Sometimes this output pulse is rather asymmetrically situated as can be seen with the D output of the decade counter of fig. 6.9. We can always make a solution to this problem by rearranging the circuit (see D output of fig. 6.15). Scalers with scaling factors n_1, n_2, n_3...can be cascaded to give a new scaler with a scaling factor $n_1 \times n_2 \times n_3$... Of course, the order in which they are connected does not influence the final scaling factor.

Shift registers

When we connect a number of flip-flops in series as shown in fig. 6.16 (for D flip-flops), we get a shift register. A shift register is a circuit which can store information and transfer it upon command. It shifts the information from one bistable element to the next when a shift pulse is received. If the contents of a given flip-flop are shifted to the following one, the circuit is said to be a forward shifting register, but when the contents are shifted to the preceeding flip-flop the circuit is called a reverse-shifting register.
The operation of such a shift register is quite simple: Let us suppose that in the forward-shifting register of fig. 6.16 a logic "1" is applied to the D input of flip-flop A. As long as there is no clock pulse, nothing happens (as we learned in chapter 5) but as soon as the clock pulse is given the "1" at the input of flip-flop A will also appear at the Q output and hence at the D input of flip-flop B. One clock pulse thus shifts the "1" at the input of flip-flop A one flip-flop (one binary place) to the right, to the input of flip-flop B. The next clock pulse shifts this "1" one place further, and so on until after the fifth clock pulse the "1" is shifted out of the register.
Similar considerations apply, of course, to the reverse-shifting register of fig. 6.17 but only the other way around. The "1" has to be fed into the input of flip-flop D and will shift to the left.
If for example we start with the binary number "0110" in the register, we see that a shift to the right gives "0011", and a shift to the left "1100"; as we learned in chapter 1, this corresponds to dividing and multiplying by 2 respectivily.
This example indicates very clearly the importance of the shift register in computers in that it offers the possibility of arithmetic operations.
In the above examples we have used D-type flip-flops for shift registers, but of course our versatile JK flip-flop can also be used for this purpose.
As with counters, forward and backward operation can be combined in shift registers to give the reversible shift register, which is shown in fig. 6.18.
As we learned in chapter 1 the data can be in parallel or serial form. Different combinations of these give four possible types of shift registers: serial in, serial out; serial in, parallel out; parallel in, serial out or parallel in, parallel out. More details of these circuits are given in chapter 8.

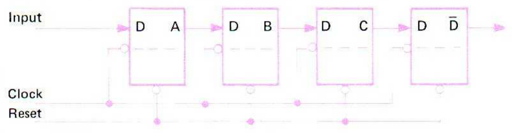


Fig. 6.16 Forward shift register.

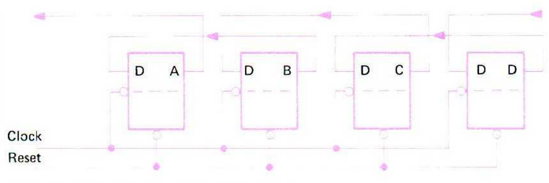


Fig. 6.17 Reverse shift register.

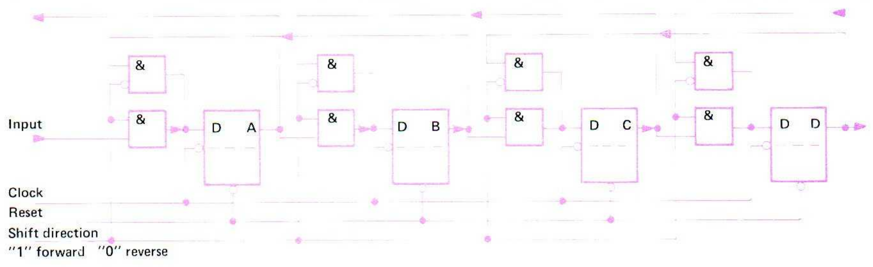
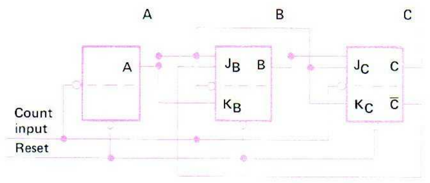


Fig. 6.18 Reversible shift register.

Questions:

Q.6.1. In the modulo-6 counter the state of the flip-flops is "101". (A = "1", B = "0", C = "1"). How many pulses has the counter counted after reset?



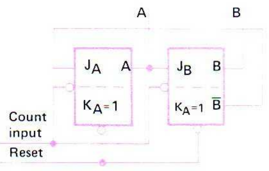
$J_B = A.\overline{C}$.
$K_B = A$
$J_C = A.B$
$K_C = A$

A = 4
B = 5
C = 6

A	
B	
C	

Q.6.2. The circuit diagram represents a:

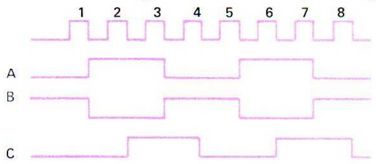
A Modulo-3 counter
B 3-bit shift register
C Modulo-5 counter

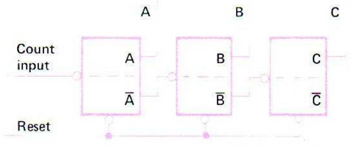


$J_A = \overline{B}$
$J_B = A$

A	
B	
C	

Q.6.3. A clock pulse is fed into the 3-bit binary down counter. Is the signal at the B output as given in A, B or C?





A	
B	
C	

Q.6.4. Which of the three truth tables is correct for a decimal counter coded in NBCD code?

	A	B	C
0	0000	0000	0000
1	0001	0001	0001
2	0010	0010	0010
3	0011	0011	0001
4	0100	0100	0110
5	0101	0101	0101
6	0110	0110	0110
7	0101	0111	0101
8	0100	1000	1010
9	1011	1001	1001

A	
B	
C	

Q.6.5. Two scalers, one with scaling factor n_1 and one with scaling factor n_2 are connected in series, n_1 before n_2. The maximum dividing speed of the combined scaler is determined by the speed of:

A The first scaler n_1
B The second scaler n_2
C Both $(n_1 \times n_2)$

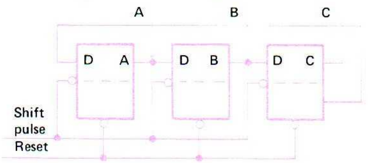
A	
B	
C	

Q.6.6. How many illegitimate states has a synchronous modulo-6 counter?

A = 3
B = 2
C = 1

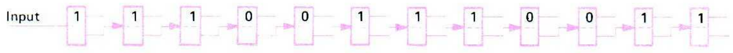
A	
B	
C	

Q.6.7. A three-bit shift register is connected as shown in the figure. After how many clock pulses (after reset to "0") are the contents of the shift register "000" again?



A	
B	
C	

Q.6.8. A 12-bit binary counter has the following state



Which is the octal number represented?

A 677
B 6347
C 7163

A	
B	
C	

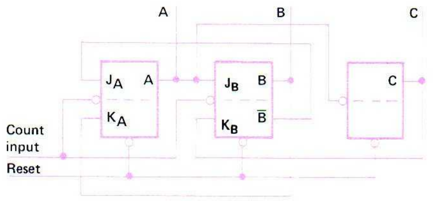
Q.6.9. What is the maximum number (in decimal notation) that can be counted with above counter?

A 999
B 4096
C 4095

A	
B	
C	

Q.6.10. In the modulo-7 counter below the illegitimate state is:

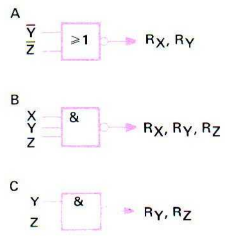
A Locked
B Not locked
C There is no illegitimate state

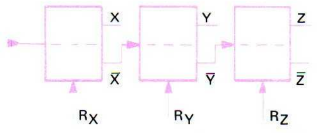


$J_A = \overline{B}$
$K_A = B$
$J_B = A$
$K_B = C$

A	
B	
C	

Q.6.11. In another number system (e.g. 6) a "decade" counter has to recycle to 0 at the 6th count. Which of the connections indicated will realize this resetting? (a logic "0" at the R inputs resets the counters)





A	
B	
C	

```
 77-11-22 14.36.43
CH000     20.  C  CH001   1710.  C  CH002   213.6  MV  CH003   0.001 V
CH004   16.07  MV CH005     5.0  %  CH006     0.0  %   CH007    940.
C
U
 77-11-22 14.36.46
CH000     19.  C  CH001   1709.  C  CH002   213.5  MV  CH003   0.001 V
CH004   16.06  MV CH005     5.0  %  CH006     0.0  %   CH007    980.
C
U
 77-11-22 14.41.29
CH000     20.  C  CH001   1670.  C  CH002   213.6  MV  CH003   0.002 V
CH004   16.08  MV CH005     5.0  %  CH006     0.0  %   CH007   1031.
C

                                 C  CH002   213.5  MV  CH003   0.001 V
                                 %  CH006     0.0  %   CH007    981.
```

Chapter 7

The circuitry of logic elements

In chapters 1 and 2 we showed how the binary number system (the one most commonly used in digital equipment) allows all calculations to be carried out by manipulation of the two digits 0 and 1, which can be represented by two distinct physical states such as "switch off" and "switch on".
In chapters 3–6 we discussed the basic structure and properties of the "logic elements" used for the manipulation of binary data, without considering how these elements are actually realized in practice. One thing which appeared clearly from this discussion was that most, if not all, logic elements can be considered as combinations of AND, OR and NOT gates. This modular approach, in which logic functions are built up by combination of a few basic elements, is widely used in digital techniques.
In this chapter we shall see how various basic logic elements can be realized as electronic circuits. As will become clear below, the nature of these basic elements depends on the properties of the electrical components used to realize them. In the early days of digital techniques, when diodes were largely used in the circuitry, it was natural to take the AND and OR gates as the basic elements.
Later, when transistors came to the fore, it became more natural to base logic circuits on the NAND and NOR gates (since the output signal of the transistor is opposed in sign to its input – in itself a kind of NOT function). When integrated circuits came to be widely used, things became simpler in some ways, but more complicated in others; however, the basic IC gates are generally NAND or NOR gates too.
Before going on to discuss the circuitry of the logic elements, we must make up our mind precisely what physical states are to be used to represent the binary digits 0 and 1.
We shall then describe the circuitry of diode gates, transistor gates (including flip-flops) and integrated-circuit gates, in that order.

Logic convention
In digital circuits, 0 and 1 are always represented by two different voltage levels, often called HIGH and LOW. However, the assignment of voltage levels to the two logic states has some consequences which cannot be seen from the logic diagram.
Inspection of table 3.1 shows that there is a duality of the AND and OR functions, because when we invert all inputs and outputs the AND function becomes an OR function and *vice versa*. This means that a circuit which acts as an AND gate when logic "0" is represented by a low level (L) and logic "1" by a high level (H) will act as an OR gate if L is taken to represent logic 1, and H for logic 0. We must thus clearly define which "logic convention" we are using. There are two possibilities.
In the **positive** logic convention logic "1" is assigned to the most positive (HIGH) level of the voltage in question and logic "0" to the least positive (LOW) level, while in the **negative** logic convention logic "1" is assigned to the most negative (LOW) level and logic "0" to the least negative (HIGH) level, fig. 7.1.
This convention is important not only for the determination of the function of a specific gate but also for the interpretation of the digital data available e.g. at the output of a measuring instrument. The following example shows very clearly the need to know which convention is valid: Suppose the following output levels are presented on a set of BCD output lines: HLLH (1-2-4-8 code). In positive logic this would mean 1001 (binary) = 9 (decimal), while in negative logic the same HLLH would mean 0110, which is the binary notation for decimal 6.
In order to avoid confusion, the positive logic convention will be used below unless otherwise specified.

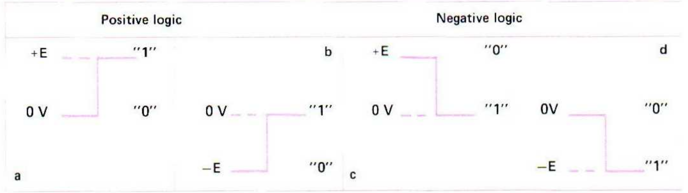


Fig. 7.1 Depedence of the logic properties on the direction of the voltage jump and the polarity of the nonzero potential E:
a) Positive logic with positive potential.
b) Positive logic with negative potential.
c) Negative logic with positive potential.
d) Negative logic with negative potential.

Basic circuitry

In the following text it is assumed that the reader has sufficient knowledge about basic semi-conductor theory; however, in order to facilitate the reading the main characteristics needed for the understanding of diodes and transistors are briefly reviewed here. A diode is conducting when forward biased, (i.e. when the anode is positive with respect to the cathode) and cut off when reverse biased. A transistor is conducting when the emitter-base junction is forward biased and the collector-base junction is reverse biased.

Diode gates

The basic circuit diagrams of a diode AND gate and OR gate are shown in fig. 7.2.

Since these gates are built up of resistors and diodes only, they are often refered to as resistor-diode logic gates in the literature.

Let us see how a resistor-diode logic AND gate works, with reference to fig. 7.2.a. In this circuit, logic "1" corresponds to +5 V (HIGH), and logic "0" to 0 V (LOW). If all the inputs A, B and C are LOW, all three diodes will be conducting so output X will be LOW too. In terms of our positive logic convention, this means that when A and B and C are "0", X is also "0". Similarly, when only one or two of the inputs are LOW, X will still be LOW. Only when A and B and C are HIGH will all diodes be cut off so that no current can flow through R, with the result that X is HIGH too. In other words, when all inputs are logic "1", the output is logic "1" too. This in accordance with the definition of the AND function.

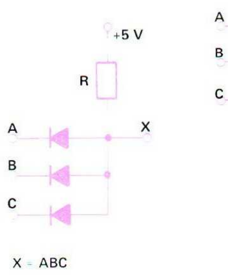
Let us now check that this circuit does become an OR gate when we use the negative logic convention, as mentioned above. If all inputs are HIGH (logic "0") all diodes are cut off, and X remains HIGH. If however one or more inputs are LOW the diode(s) in question will conduct, current will flow through R, and X will become LOW (= "1"). This thus corresponds exactly with the function of the OR gate.

The operation of the OR gate in positive logic (fig. 7.2.b) is as follows:

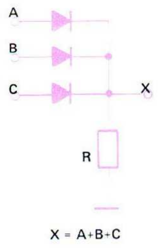
When all inputs are LOW (logic "0"), no current flows through R so X remains at earth potential (LOW = "0"), however, when one or more inputs go HIGH (+5 V = "1") the diode(s) in question conduct, current flows through R, and X becomes HIGH.

Transistor gates

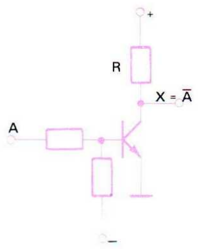
A single transistor in the common-emitter configuration may be regarded as a NOT gate or invertor; see fig. 7.3. A combination of this transistor with the diode gates described above gives a NAND and a NOR gate. These are shown in Table 7.1., which gives a survey of all the NAND and NOR gates in the various logic systems described in this article. No further comment is required on the gates in diode logic.



7.2a



7.2b



7.3

Fig. 7.2 An AND gate (a) and OR gate (b) in diode logic, also called dioderesistor logic. (Positive logic convention)

Fig. 7.3 A NOT gate formed by a transistor in the common-emitter configuration.

• Resistor-transistor logic

The basic function which can be realized with resistors and transistors alone is the NOR function. The resistor-transistor logic NOR gate is also shown in Table 7.1.

The operation of this gate may be explained as follows. When both inputs A and B are LOW the base-emitter junction is reverse biased and the transistor is cut off; there is thus no current through R, so X remains HIGH. If on the other hand one OR both of the inputs are HIGH the base-emitter junction is forward biased, the transistor is conducting and X becomes LOW. The circuit thus indeed functions as a NOR gate.

• Direct-coupled transistor Logic

A NAND gate and a NOR gate in direct-coupled transistor logic are shown in Table 7.1. The NAND gate works as follows. Only when both transistors are conducting (i.e. A and B both HIGH) does current flow through R so that X becomes LOW; in all other situations, X remains HIGH, which is what we require of a NAND gate by definition. If one or both inputs of the NOR gate are HIGH, one or both transistors will be conducting and current will flow through R so that X will become low. If on the other hand neither A nor B is HIGH, X remains HIGH; this is in line with the definition of a NOR gate.

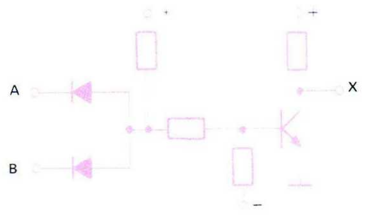
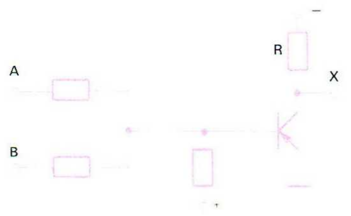
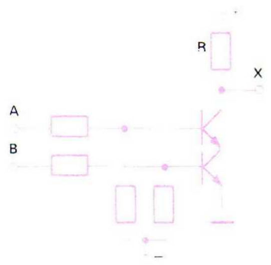
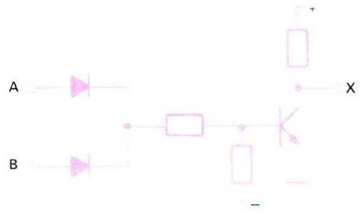
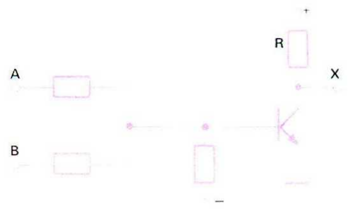
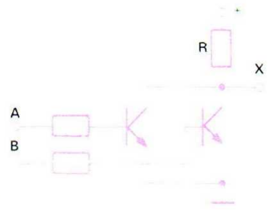
Logic type	Diode logic	Resistor-transistor logic	Direct-coupled transistor logic
NAND-gate circuit $X = \overline{A.B}$.			
NOR-gate circuit $X = \overline{A+B}$			

Table 7.1 Basic circuit diagrams of NAND and NOR gates in various types of logic, using discrete components.

Flip-flops

The basic circuit diagram of the flip-flop is shown in fig. 7.4. A flip-flop can be regarded as two invertors in cascade, the output of the second being connected to the input of the first (see also the block diagram of the RS flip-flop 5.1).

Let us assume that initially TS_1 is cut off and TS_2 is conducting; the output level at $\overline{Q}$ is thus LOW (≈ 0 V = "0"). Thanks to the resistor network, the base of TS_1 (S) is also low and output Q is HIGH ($\approx +5$ V = "1"); consequently, the base of TS_2 (R) will also be HIGH, thus maintaining the situation. A HIGH level (or a short HIGH pulse) at the SET input will make TS_1 conducting; the collector voltage of TS_1 and the base voltage of TS_2 drop, and so on. The effect is cumulative, resulting in a second stable state. A further HIGH pulse at the SET input will cause no change.

Inspection of fig. 7.4 will confirm the statement made in chapter 5 about the RS flip-flops: R and S cannot both be HIGH together, since when one of the transistors is conducting, the other must be cut off.

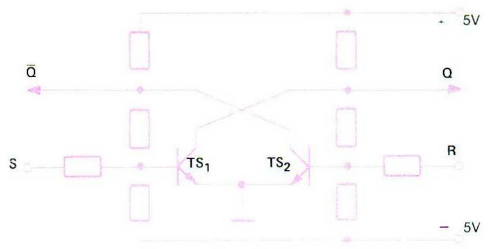


Fig. 7.4 Basic circuit diagram of the flip-flop.

Integrated-circuit (IC) families

As soon as the design of logic circuitry got under way, circuits started to get increasingly complex – and bulky. The introduction of integrated circuits (IC's) and in particular the development of planar techniques came just in time to solve the problem of bulk, as now a lot of functions could be combined in one little semi conductor chip. However, this increased complexity makes IC's conceptually somewhat more difficult to deal with. A considerable number of characteristics have to be borne in mind, of which we discuss the main ones below before entering on a brief discussion of the structure and operation of the various IC families where the values of these characteristics will frequently be quoted.

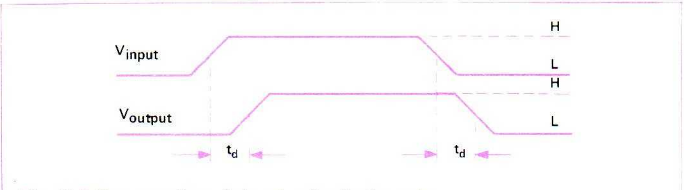


Fig. 7.5 Propagation delay t_d of a logic gate.

• CHARACTERISTICS OF IC GATES

• *Speed* A logic gate always requires a certain time to pass from one state to another. This is generally expressed in terms of the propagation delay t_d (see fig. 7.5), which is defined as the time required for a binary digit to be propagated from input to output.
The rate at which a flip-flop can switch from one state to the other is called its clock rate.

• *Threshold voltage* The threshold voltage of a gate circuit is defined as the input voltage at which the gate in question just switches from one state to the other.

• *Noise margin* In order to avoid errors in a logic system due to parasitic voltages (e.g. spikes generated by switching transistors, etc.), logic devices should have a wide noise margin, i.e. the voltage swing between the two binary states should be as large as possible.

• *Power dissipation* This is generally understood to mean the power required for operation of the logic device. As circuit complexity (in particular the complexity of IC chips) increases, the power dissipation per gate must decrease, in connection with limitations on the amount of heat which may be dissipated in the semiconductor junctions.

• *Fan-out and fan-in* A logic gate usually has to drive a number of other logic elements. The fan-out is defined as the maximum number of inputs which can be connected to one output. Similarly, the fan-in is the number of outputs which can be connected to one input.
We can now proceed to our discussion of the various IC logic "families" which have been developed so far. Most of the gates we shall deal with are NAND or NOR gates.

DCTL
Direct-coupled transistor logic

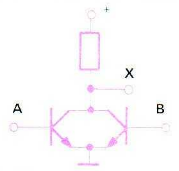


Fig. 7.6a

RTL
Resistor-transistor logic

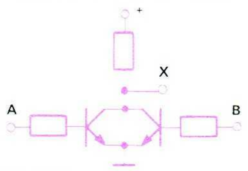


Fig. 7.6b

• DCTL family (direct-coupled transistor logic), fig. 7.6a. This was one of the first IC logic techniques. The basic logic gate has the NOR configuration. However, this circuit had the big disadvantage that slight differences in the properties of the various transistors used could cause one transistor to draw all the current, thus preventing proper operation of the circuit (current hogging). This led to the development of RTL family.

• RTL family (resistor-transistor logic), fig. 7.6b
The basic configuration is a NOR gate here too. With this circuit resistors are added in the base circuits which reduce the differences in transistor current. The power dissipation rate was rather low. On the other hand, the output voltage swing is also low, giving high sensitivity to noise spikes on the signal lines. Moreover, the series resistors reduced the speed of the circuit. The search for further improvements in design led to the development of the next family.

DTL
Diode-transistor logic

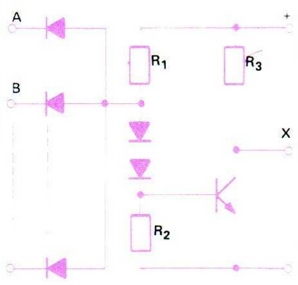


Fig. 7.7a

TTL
Transistor-transistor logic

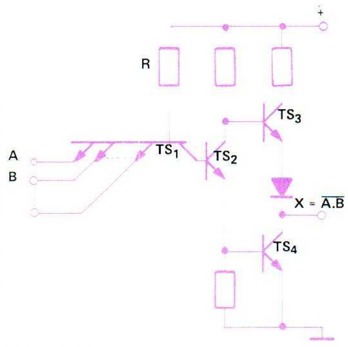


Fig. 7.7b

• DTL family (diode-transistor logic), fig. 7.7a
The basic configuration is the NAND gate, i.e. the output is LOW when all inputs are HIGH.
In this case current will flow through the input resistor (R) and the two stand-off diodes to the base of the output transistor which will thus be switched on. The output voltage will then be LOW. If any of the inputs drops to earth potential (logic "0"), the corresponding input diode will conduct and current flows through this diode and R, causing a voltage drop at the input of the stand-off diodes. The base voltage becomes LOW and the transistor remains cut off so the output is HIGH. The stand-off diodes are included to increase the threshold voltage of the gate.

The main characteristics are propagation delay 30 ns, flip-flop clock rate 10 MHz, fan-out 8 and noise margin 1.2 V. A further advantage is that DTL circuitry (like RTL) can be used in WIRED-AND configurations, so no additional gates are required to tie a number of outputs together.

• TTL family (transistor-transistor logic), fig. 7.7b
This logic family has become the most popular type of recent years. One of the basic differences between this type and DTL logic is that instead of an input circuit built up with diodes in the usual way a multi-emitter transistor is used. Each emitter-base diode serves as an input diode.
This multi-emitter input has the advantage that less space is required on the semi-conductor chip for a given number of inputs, which means that more functions can be realized on a single chip, TTL logic therefore lends itself much more to the realization of more complex circuits (MSI-medium-scale integration) than the above-mentioned families.
The basic function of the TTL gate, just as with the DTL gate, is the NAND function. This can be seen as follows: If one or more inputs are at earth level (logic "0"), current will flow through input resistor R. Consequently, the collector of the input transistor will be LOW. Only when all inputs are HIGH will the collector be HIGH too. The input circuit in fact gives normal AND operation. The next stage acts as a kind of phase splitter for driving the "totem-pole" output. When TS_2 is on (all inputs HIGH) TS_3 will be cut off and TS_4 will be off, resulting in a LOW output; which is the NAND function. The diode in the output chain ensures that TS_3 is cut off when TS_4 goes on. A disadvantage of the totem-pole output is that no outputs can be connected in parallel, which means that WIRED-AND connections (see "Distributed connections", page 26) are impossible: if two or more TTL outputs were connected together and one output was LOW and the other HIGH, this would give more or less short-circuit and the power dissipation would become much too high.

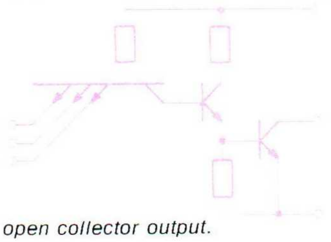


Fig. 7.8 TTL with open collector output.

A TTL gate with special output circuit (open collector output fig. 7.8) has therefore been designed to make WIRED-AND connections possible. In this type all collectors of the TTL gates are connected in parallel and have a common resistor to the positive power supply. The LOW level thus always predominates, and no harm is done to the other gates.
When the outputs of such open-collector logic gates are connected together and the common line is connected via a resistor to the positive supply, this common line will only be HIGH if all the outputs are HIGH. When only one of the outputs is LOW, the common line will also be LOW, which gives this arrangement the function of an AND gate (in positive logic), fig. 7.9.

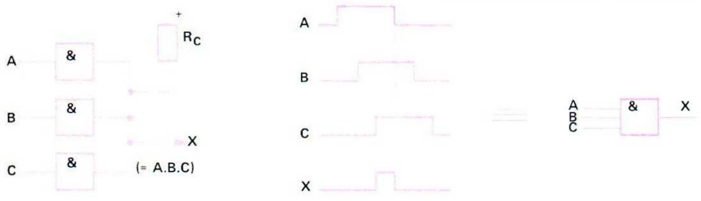


Fig. 7.9 WIRED-AND connection for positive logic (active HIGH).

As we have learnt, when we invert the logic convention an AND gate becomes an OR gate (and *vice versa*); so in a system with active LOW ("1" is low level, "0" is high level) we can realize a WIRED-OR gate as shown in fig. 7.10.

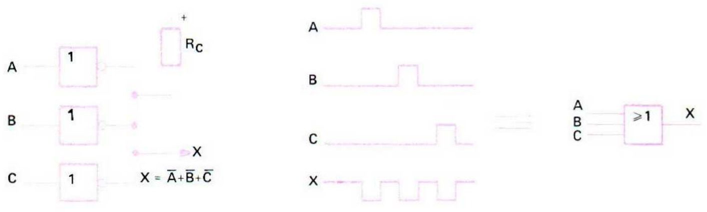


Fig. 7.10 WIRED-OR Connection for negative logic (active LOW).

When one of the signals A, B, C is HIGH (logic "0") the output of the corresponding gate is LOW with the result that the common line is LOW independent of the state of the other gates. One signal is thus sufficient to bring the common line to a LOW level (logic "1"), which is the function of the OR gate.
Summarising, we may state that a WIRED connection functions as an AND gate for positive logic and as an OR gate for negative logic.

The main characteristics of TTL logic are: propagation delay 10 ns, flip-flop rate 20 MHz, fan-out 10, noise margin 0.4 V, dissipation/gate 10 mW.
There are various types of TTL families, mostly differing only in a few aspects such as power dissipation or speed. A rather fast TTL logic is the so called "Schottky-TTL" logic.
In all the above-mentioned logic systems "saturated logic" is used, i.e. transistors which are conducting are driven into saturation. The speed of a circuit with saturated logic depends very much on storage time and RC constants (it takes time to get the transistor out of saturation). Thus, if a transistor can be kept out of the saturation region, the speed can be increased. This can be done by clamping the base-collector junction to a voltage below the saturation level. A germanium diode (with a low forward voltage drop) shunted across the base-collector junction would do just this. Of course, germanium diodes cannot be used in IC technology, but fortunately Schottky diodes can be integrated and can perform the same function. Use of these Schottky-clamped transistors in the IC process allows even faster TTL circuits to be made; this is the above-mentioned Schottky-TTL logic, fig. 7.11.

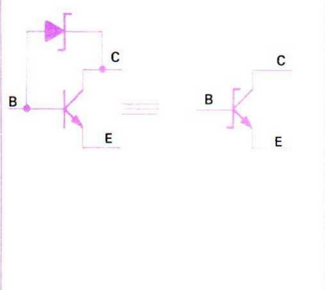


Fig. 7.11a Schottky transistor.

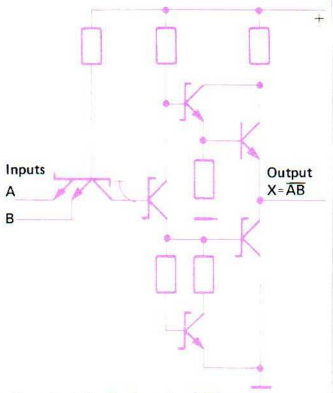


Fig. 7.11b Schottky TTL gate.

Using these Schottky techniques in normal TTL gates decreases the propagation delay by a factor of 3 to 4 at a given power consumption. Flip-flop count rates of 80–100 MHz can then be reached. A very popular TTL family nowadays is LPSTTL (Low power Schottky TTL), where the rather slow low-power TTL gates are equiped with Schottky transistors, resulting in propagation delays of the order of 5 ns. This is about the same as with standard TTL but with only 25% of the power consumption of the latter.

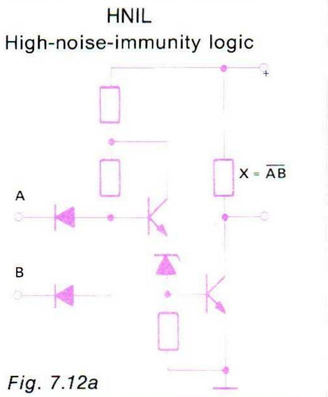


Fig. 7.12a

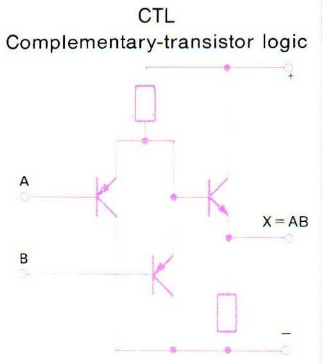


Fig. 7.12b

• HNIL family (high-noise-immunity logic), fig. 7.12a
This type, also called HTL (high-threshold logic), has been specially designed for application in noisy environments.
The basic gate form is the same as for DTL except for the Zener diode which has replaced the normal stand-off diode(s).
The higher threshold is thus obtained by adding the Zener breakdown voltage (about 7.0 V) to the base-emitter forward voltage, giving a threshold voltage of about 7.5 V.
The logic voltage ranges of HNIL, TTL and DTL are compared in fig. 7.13 which clearly shows the superiority of HNIL in this respect.

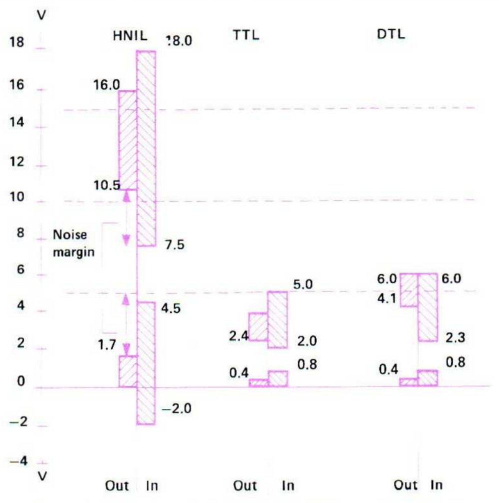


Fig. 7.13 Logic voltage levels for HNIL, TTL and DTL logic. In each column the upper shaded area indicates the voltage range corresponding to logic HIGH, and the lower shaded area to logic LOW. The exceptionally wide noise margin of HNIL may be clearly seen.

• ECL family (emitter-coupled logic), fig. 7.14a
This type – also called current-mode logic (CML) – differs completely from the previous types of logic families in that the transistors when conducting are not saturated, so logic swings are reduced. This is another way of increasing the speed of the gate.

The basic gate is shown in fig. 7.14a; it is basically a NOR gate, but also has a complementary OR output. Although the circuit looks rather complex, its operation is quite simple and more or less linear.
The input is a differential amplifier formed by the input transistors and TS_1; TS_2 provides the reference level for TS_1, TS_3 and TS_4 are emitter-followers giving the NOR and OR outputs. The emitter-followers give the gate a low output impedance, a large fan-out and fast switching times; quite high capacitive loading is also possible.
If both A and B are LOW, the common emitter line will be lower than the reference voltage (which is defined by the logic levels) and the input transistors will be cut off; their collector voltage will thus be HIGH so TS_3 is conducting and the output of TS_3 is HIGH (i.e. NOR function). TS_1, however, will be conducting (emitter-base junction is forward biased), so the collector of TS_1 and hence the base and emitter of TS_4 will be LOW (OR function).

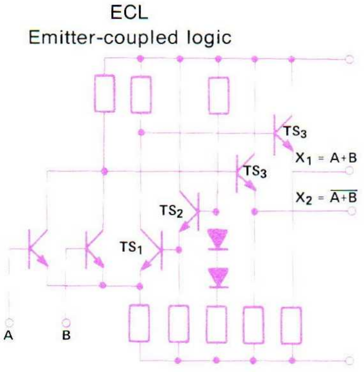


Fig. 7.14a

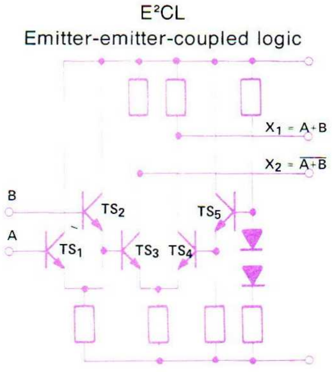


Fig. 7.14b

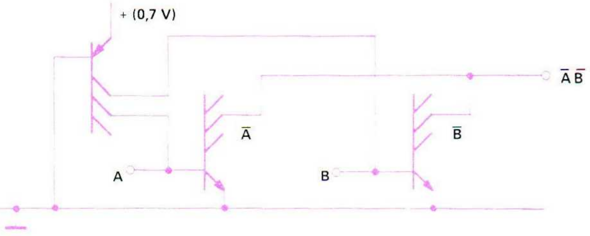


Fig. 7.15 I²L gate

• E²CL family
This type, see fig. 7.14b, is another variant with basically the same circuit except that the emitter-followers are at the input instead of at the output (when two gates are connected in series then there is no difference in principle between ECL and E²CL).

• CTL family (complementary-transistor logic)
Complementary-transistor logic is another form of current logic; here too, the transistors do not go into saturation.
The basic gate (fig. 7.12b), which performs the AND function, is made with a combination of PNP and NPN transistors.
The operation is as follows. With both inputs HIGH the input transistors are cut off, no current flows, the common emitter voltage is high and consequently the output is HIGH.
If one or more inputs are LOW, the transistor(s) are conducting, the emitter voltage drops so the output will be LOW too.

• I²L family (integrated injection logic)
The I²L family is especially designed for LSI (large-scale integration). The logic gates are small in area on the chip which means a good packing density (gates per mm²) and reasonable speed at very low power dissipation. The noise margin is rather low, so special input and output circuits are required.
The gate itself performs only the NOT function with multiple isolated outputs. A logic function is achieved by WIRED-AND connections. Compared with TTL where a multi-emitter transistor is used at the input, here the same transistor is upside down and thus works as a multi collector transistor. The supply current is delivered by a PNP transistor, the emitter of which is common to a large number of gates. The supply current is distributed (by current injection) in equal portions among the gates. When the NOT gate is "ON" the PNP transistor saturates and must therefore be properly designed. As no resistors are used, the circuit will work at any supply current; however, lower current levels cause longer delay times. The average delay time is 30 ns at 50 μW. Special HF processes can be used to give a better delay (5–10 ns). The circuit diagram of an I²L gate is given in fig. 7.15, which is self-explanatory.

• MOS families (metal-oxide semiconductor logic)
All the IC families described above are based on the "bipolar" transistor. A new development, the MOS (metal-oxide-semiconductor) transistor has however greatly influenced the design of modern integrated circuits.
Before giving further details of MOS logic families, we shall briefly describe the operation of the MOS transistor (MOST). The MOS transistor, like the bipolar transistor, consists basically of three electrodes: in this case the source, the drain and the gate (see fig. 7.16). As can be seen from the figure, there are two PN junctions back to back between source and drain. When a voltage is applied between source and drain in such a way that the drain junction is reverse biased (source positive, drain negative), only a small leakage current will flow from source to drain. When now the gate electrode (which is over the space between source and drain) is made sufficiently negative with respect to the source, holes in the N-type substrate are attracted towards the gate electrode (which is insulated from the substrate by means of a very thin layer of silicon dioxide, hence the name of insulated-gate FET (field-effect transistor) which is often given to the MOS transistor).

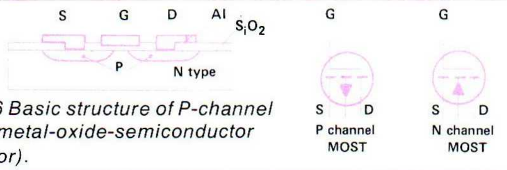


Fig. 7.16 Basic structure of P-channel MOST (metal-oxide-semiconductor transistor).

The extra holes will change the N-type region under the gate to P-type, so a P-type channel will join the two P-type regions at source and drain and a current will flow from source to drain. This current can be increased or decreased by varying the amplitude of the gate voltage. In fact a MOST can be regarded as a variable resistor between source and drain, controlled by the gate voltage. This type of MOS transistor is called a P-channel MOST. Similarly, if we interchange the P-type and N-type materials and the polarities, we get an N-channel MOST. One of the big advantages of the MOS technique in integrated circuits is that it requires a much smaller crystal area than normal bipolar IC's, so a much higher element density is possible on the chip, giving much lower manufacturing costs per element.
The MOST is often used with the gate connected to the drain resulting in two well defined states: OFF (no current, high impedance) and ON (max. current, saturation, low impedance). In this configuration, the MOST is very suitable for logic applications.

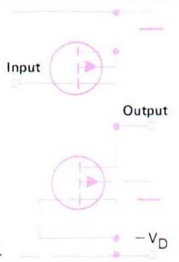


Fig. 7.17 P-channel MOS inverter.

The best way to describe the MOS logic gates is to start with the MOS invertor (fig. 7.17), which works as follows (negative logic: "0" is ground potential, "1" is negative). When the input is "0" (near ground potential), the upper MOST (driver) is OFF and the lower MOST pulls the output to the $-V_d$ (minus the threshold voltage) level, i.e. logic "1". The threshold voltage for MOST's is defined as the gate voltage at which conduction between source and drain just starts. When the input is "1" (at least a voltage above the threshold) the driver MOST is conducting and thus has a low impedance. The output voltage – defined by the impedance ratio of the two MOST's – then drops almost to ground potential (logic "0"). When two or more driver MOST's are added in parallel to the inverter, one gets a NOR gate – in negative logic, see Table 7.2.

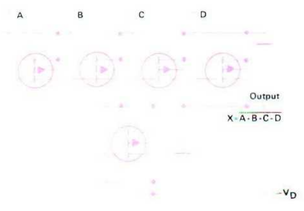
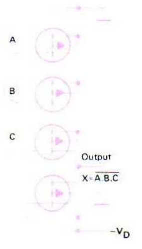
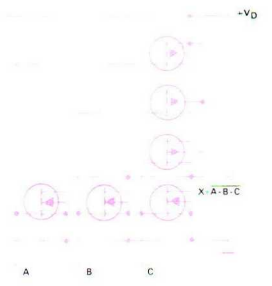
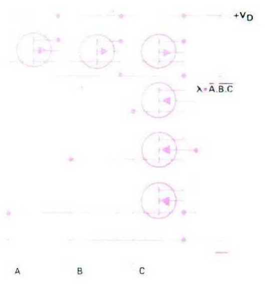
Logic type	NOR-gate circuit	NAND-gate circuit
P-MOS		
C-MOS		

Table 7.2 MOS families, simplified circuit diagrams.

	RTL DCTL	DTL	HNIL	TTL	Schottky TTL	LPS TTL	CTL	ECL E²CL	P-MOS	C-MOS	N-MOS	I²L
Basic gate	NOR	NAND	NAND	NAND	NAND	NAND	AND	OR-NOR	NAND/NOR	NAND/NOR	NAND/NOR	NOR
Propagation delay (ns)	10–100	20–70	90–150	6–12	3	5	2–3	1–4	300	70	10	30–100
Flip-flop rate (MHz)	5	10	4	10–50	100	20	100–200	60–400	2	10	20	3
Dissipation/gate (mW)	12	10	50	10–20	20	5	30	40–60	0.2–10	1	0.2	0.05
Fan-out	5	8	10	10	10	5	20	25	20	50	20	5

Table 7.3 Characteristic data of the main types of IC logic families.

Replacing the driver by a number of MOST's in series gives (in negative logic) a NAND gate (see Table 7.2 again).
In the latter case the total series impedance when all MOST's are ON must be as low as with the single-MOST inverter. It will be clear from this description that these MOS circuits bear a close resemblance to the "bipolar" logic circuits described above.
The high impedance of the MOST gives it the disadvantage of a lower speed than bipolar circuits, owing to the fact that stray capacitances cannot be charged quickly. Typical switching times are of the order of 1 μs (see Table 7.3).
Performance can be improved with a new type of load devices, the depletion n-channel MOS transistor. The main feature of this device is that it is conducting at zero gate source voltage. Only a slight increase in drain source voltage is required to saturate the transistor, i.e. to limit the current through it. When this transistor is used as a load device its behaviour is more or less ideal. With a "0" at the input of the inverter (fig. 7.18) and the gate connected to the output, the driver will be off. The load transistor remains fairly well conducting up to a voltage very close to the drain voltage of the load transistor which gives a nearly full swing of the output. With a "1" at the input, the output voltage drops to zero. The current through the load is limited, because this type of transistor is so easily saturated.
Switching times can be improved by making use of C-MOS (complementary MOS) techniques, in which both

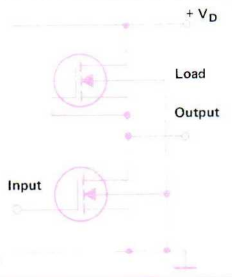
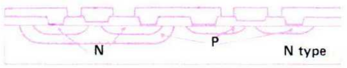


Fig. 7.18 N-channel MOS inverter.



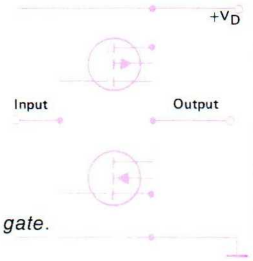
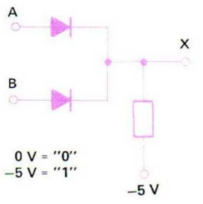


Fig. 7.19 Basic structure of a C-MOS logic gate.
Fig. 7.20 C-MOS inverter.

P-channel and N-channel MOST's are used. The basic structure is given in fig. 7.19 (C-MOS inverter). The bottom transistor of the inverter (fig. 7.17) is replaced by an N-channel MOST, thus giving a P-channel and an N-channel MOST connected in series (fig. 7.20). Only one device at a time is turned ON (low impedance), the other device being OFF (high impedance).
With a "0" at the input, the N-channel MOST will be OFF and the P-channel will be ON, which gives an output close to the supply voltage (logic "1"). With a "1" at the input the situation will be reversed, with an output voltage close to ground potential (logic "0"). The advantage of this circuit is that when the impedance of one device is decreasing, the impedance of the other is increasing, and *vice versa*, giving a push-pull effect which narrows the transition region, giving sharper transfer characteristics and thus increasing speed. Another advantage of C-MOS is the low power consumption, because only one transistor is turned on at a time.
In the same way as described above for P-channel gates, a number of basic elements in parallel gives a NOR gate, while a series combination gives a NAND gate. For these two circuits see also Table 7.2.
As mentioned above, the great advantage of MOS techniques is the high-density packing. In other words, large complex circuits (LSI – large-scale integration) can be made by MOS techniques. Simple gates or flip-flops would cost too much if made this way. Since MOS techniques are mainly used for LSI circuits, there will be a tendency for each MOS circuit to be unique (custom-built for a given application) rather than general-purpose.

Questions

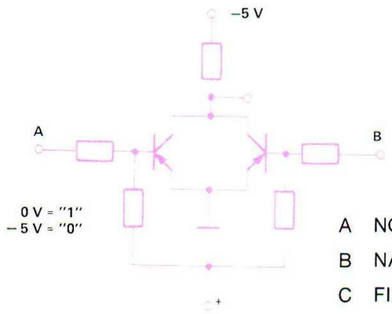
Q.7.1. If negative logic is used, the diode gate in the circuit below represents an:



A AND gate
B OR gate
C NOR gate

A	
B	
C	

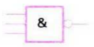
Q.7.2 The circuit below (positive logic) is a:



A NOR gate
B NAND gate
C Flip-flop

A	
B	
C	

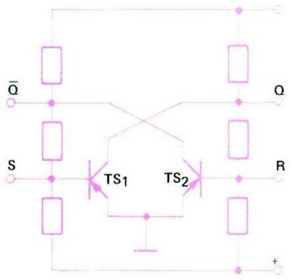
Q.7.3 A three-input NAND gate is to be used as inverter. Which of the following measures will achieve this end?



A All inputs are connected together, making one input.
B The two inputs not used are connected to the logic "1" level.
C The two inputs not used are connected to ground ("0").

A	
B	
C	

Q.7.4 In the flip-flop circuit below, TS_1 is conducting and TS_2 is cut off. To which input must a positive pulse be applied to change the state of the flip-flop?



A R input
B S input
C Either R or S

A	
B	
C	

Q.7.5 What is the purpose of the stand-off diodes in the basic DTL gate?

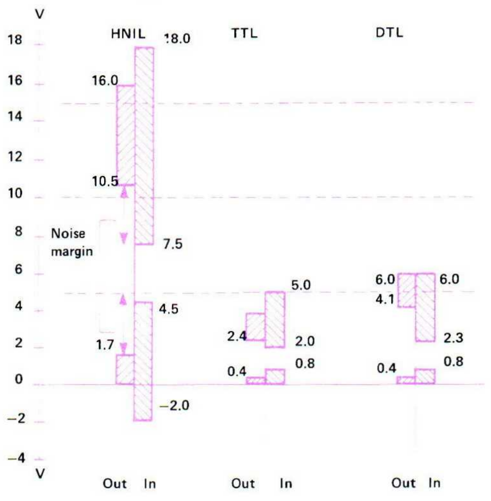
A To protect the input against over-voltages.

B To raise the threshold level.

C To provide facilities for WIRED-AND connections.

A	
B	
C	

Q.7.6 The noise margin in TTL for the "1" state is:



A : 3.9 V
B : 0.4 V
C : 2.4 V

A	
B	
C	

Q.7.7 The basic HNIL gate is

A AND
B NOR
C NAND

A	
B	
C	

Q.7.8 The switching speed of emitter-coupled logic is very high because:

A In this type of logic transistors are not saturated when conducting.
B The circuit uses emitter-coupled transistor circuits.
C The gate operates in negative logic.

A	
B	
C	

Chapter 8

Interfaces and measuring systems

Interfaces

In the language of modern technology, the combination of a number (often a large number) of instruments, machines, etc., for a single or over-all purpose is called a ''system''. Thanks to the simplicity of their ''logic'', digital measuring instruments lend themselves very well to use in test systems.
An example of a simple system is a digital voltmeter (DVM) and a printer. The value measured by the DVM is encoded (e.g. in the BCD code) and transported to the printer. In this example the instrument (DVM) has a digital output, figure 8.1.

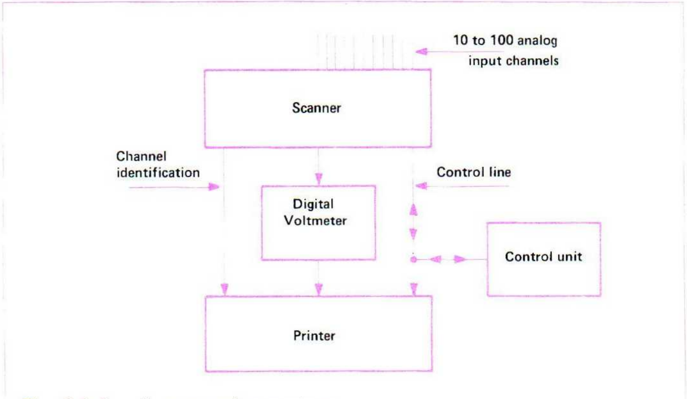


Fig. 8.1 Small measuring system.

One can also imagine an instrument being remote-controlled by means of digital signals (in case of the DVM, e.g. the voltage range could be controlled in this way); the instrument then also has a digital input. It follows that one could build up a more or less automatic measuring system by connecting the digital inputs and outputs of the instruments involved in some appropriate way. Addition of a computer to the system would make it fully automatic. In order to get all these instruments working together, it is necessary to establish a standard digital ''language'' for the various signals passing between them.
These signals (mostly in digital form) are processed in the ''interface'' (a special circuit in which the connections between the instruments are realized). This interface can be regarded as a kind of translater for the various communication signals passing to and from the instruments in the system. As there are no international standards for digital systems yet, the interface can also play the role of interpreter for the various logics.

Logic polarity and level
First of all, the logic polarity must be defined; either positive or negative logic can be used, as described in the previous chapter. The two offer equal performance and flexibility (fig. 7.1). Secondly, the logic level must be fixed. Any voltage could be used in theory, but in practice we naturally tend to choose the levels used in standard integrated circuits (e.g. 5 V TTL; 6 V DTL; 12 V HNIL). The choice of the type of IC family used in the instrument thus more or less defines the logic levels.
Each level has its specific advantages and disadvantages. The levels typical of DTL (6 V) and TTL (5 V) have the advantages of rather low price and direct compatibility with other instruments using these types of IC's (and this includes most of the instruments made today). They have, however, the big disadvantage of poor noise immunity and poor protection against overload (fig. 7.13). The higher logic levels (such as HNIL), however, have the disadvantages of higher price, slower speed and restricted availability which weigh against the big advantages of good noise immunity and good overload protection.
Depending on the technical requirements on the signal lines between the various instruments, a lower or a higher voltage level will be chosen.

Data signals and control signals
The various digital signals passing between the instruments can roughly be divided into two main groups:

- a. Data signals
- b. Control signals

• Data signals are the signals which contain information concerning the results of measurements (e.g. the BCD output of counters or DVM's) or concerning the remote control of an instrument.
In case of a 9-digit BCD counter, the output data would contain $9 \times 4 = 36$ bits. As regards the use of data signals for programming (control) purposes, a two-position switch can be controlled by one bit of information (e.g. "0" is OFF and "1" is ON), while e.g. a 6-position rotary switch (for selection of the voltage range, for example) can be controlled by 3 bits of binary encoded information.
• Control signals are signals which do not contain DATA, but morely initiate or prevent a certain action, e.g.:

START	starts the measurement of e.g. a DVM or counter
READY	gives the information that the instrument has completed its measurement
CALL	sets the required signal at the output of an instrument – e.g. output voltage of power supply or generator
INHIBIT	prevents the start of a new measurement by a counter or DVM until another instrument (e.g. printer) has fully processed a previous batch of data from the instrument involved
STOP	stops the instrument

Control signals are generally processed apart from the data signals; there are often direct control lines from one instrument to the other, since this ensures that the data and control signals are kept quite separate at the cost of a few extra lines. DATA signals often require more processing than control signals. This is easy to understand when we remember that e.g. the BCD output of a 9-digit counter already has 36 output bits (and would thus require an equal number of output lines unless special measures were taken), while a fully programmable modern pulse generator requires more than 100 input bits (lines). Especially when these DATA have to be transported, the number of lines involved can give trouble.
A number of data signals (bits) are generally grouped together in a WORD (or BYTE). Such a word can contain any number of bits (8, 12, 16, etc).
Now the transport of DATA generally occurs in one of three different ways:

• Full parallel (fig. 8.2a).
All signals (words and bits) are sent (or received) simultaneously. The advantage of this method is the speed at which the transfer is realized; the disadvantage is the higher number of lines required.

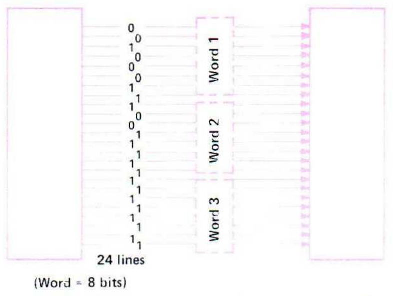


Fig. 8.2a Data transport full parallel (Bit parallel word parallel).

• World serial – bit parallel (fig. 8.2b).
Here the total information is split up into N words, each containing P bits. The words are transferred one after the other, while the bits in each word are transferred in parallel. Since the information is not all sent at the same time, the receiving instruments need a STORAGE (or memory, e.g. D flip-flops) for the information contained in the various words.
The disadvantage of this method is quite clear: it takes at least N times longer to transfer all the information. The advantage, on the other hand, is that only P lines are required (N times less than in full parallel).

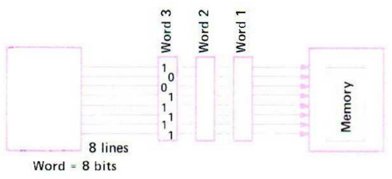


Fig. 8.2b Data transport (Word serial bit parallel).

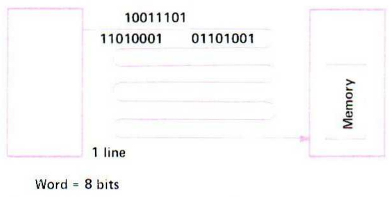


Fig. 8.2c Data transport full serial (Word serial bit serial).

• Full serial (fig. 8.2c).
In this mode of transfer not only the words but also the bits in each word are transported one after the other: first of all the bits of the first word are transferred serially, then those of the second word and so on. Here too, the receiving instrument requires a memory.
It will be clear that the transfer speed is drastically reduced in this method (N × P times compared with full parallel). The great advantage, however, is that only one signal line is needed, which can be very useful for transmission via telephone lines, radio lines etc.

Code converters
As we have explained above, digital signals have quite a number of different parameters which may have to be changed at the interface between two different parts of a system, or between two different systems. We shall now discuss the various converters used for this purpose.

• Positive-to-negative logic converter (and *vice versa.*)
This is in fact nothing but the NOT gate or inverter mentioned earlier in this course (figure 8.3).

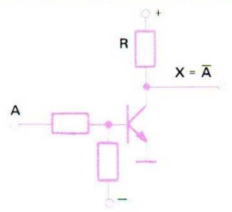


Fig. 8.3 Inverter.

• Logic-level converter
Conversion of a logic level generally means amplification of the logic signal (low to high) or attenuation (high to low). The average DC value is sometimes shifted at the same time. For low-to-high conversion one or more transistors are used, while either active or passive devices (resistor networks) can be used for the opposite conversion. Some examples are shown in fig. 8.4.

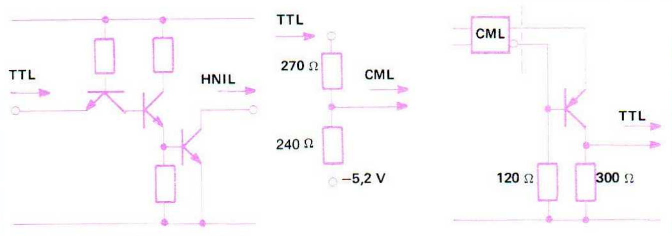


Fig. 8.4 Level converters.

• Series-parallel (S/P) converter
Serial data can be converted into parallel by means of a shift register. In a shift register (see chapter 6) a number of flip-flops are connected in series, an example of which is given in fig. 8.5.

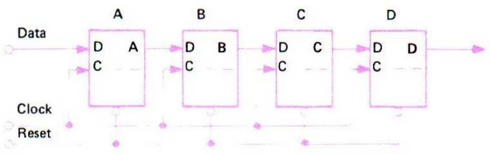
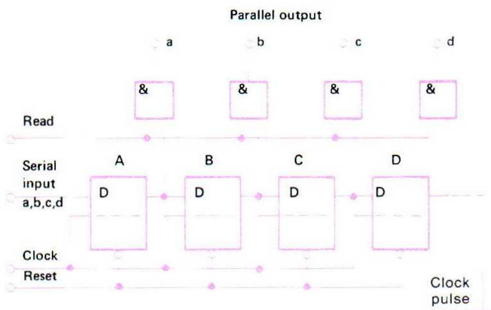


Fig. 8.5 Shift register.

When now e.g. a 4-bit serial data word a, b, c, d is applied to the input of the shift register as shown in fig. 8.6, in such a manner that bit d (which can be either "0" or "1", as can a, b and c) is the first to enter, then after the first clock pulse flip-flop A contains bit d, after the next clock pulse bit d is shifted to flip-flop B and bit c enters flip-flop A on so on, until after the fourth clock pulse the four flip-flop A, B, C and D contain the bits a, b, c and d, which remain there until the shift register is reset. It is obvious that clock rate and data rate must be synchronized.
The data are now available in parallel form at the outputs of the four flip-flops, and can be read out via the four AND gates shown in fig. 8.6, with the aid of a read pulse. It will be clear that the capacity of this converter can easily be expanded by adding more flip-flops and an equal number of AND gates to the shift register.



Clock pulse	Q_A	Q_B	Q_C	Q_D
0	0	0	0	0
1	d	0	0	0
2	c	d	0	0
3	b	c	d	0
4	a	b	c	d

Fig. 8.6 Series/parallel converter with truth table.

• Parallel-series (P/S) converter
The circuit diagram of the parallel-series converter is given in fig. 8.7. This circuit uses a shift register the other way round. First the parallel data are fed into the appropriate flip-flops by means of the ENTER pulse. (For this purpose, the flip-flops used have a preset input, see chapter 6). After the first clock pulse, bit "d" appears at the output and bits a, b and c shift one place to the right. The second clock pulse will then produce bit "c" at the output while bits a and b shift one place further and so on. After the fourth clock pulse (in this example) the parallel word will have been completely transformed into a serialized version.

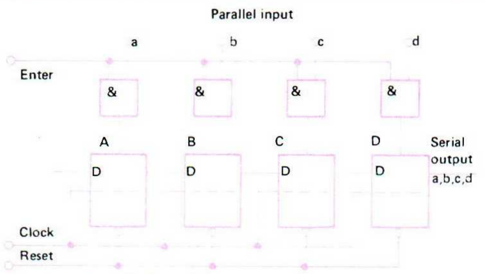


Fig. 8.7 Parallel/series converter.

Because of the great similarity between S/P and P/S converters, it is an obvious idea to combine the two into a single circuit. This combined converter is depicted in fig. 8.8, which is self-explanatory.

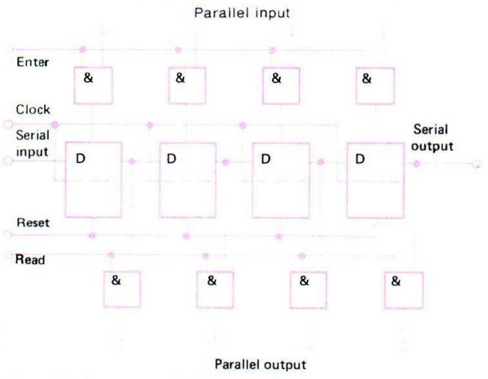


Fig. 8.8 Combined S/P and P/S converter.

• Binary-to-BCD converter (and *vice versa*)
BCD-to-binary conversion is sometimes used to cut down the number of lines needed for the transport of large amounts of data (only 3.3 binary bits are needed for one decade, as compared with 4 BCD bits). The disadvantage of the pure binary code is that it is not easily readable, so a binary-to-BCD converter will generally be needed too. There are many possible ways of carrying out this conversion; the one described below makes use of forward and reverse counters. The down counter only differs with the normal BCD or binary up counter (described in chapter 6 of this course) in that at each clock pulse the total BCD or binary value stored is decreased by 1 bit. Now a BCD-to-binary converter works as follows: First the appropriate BCD value is set in the BCD down counter. Then the clock pulse is applied simultaneously to the BCD chain (fig. 8.9 and to the upcounting binary chain. At each clock pulse the value stored in the BCD chain decrease by one bit, while that in the binary chain increases by one bit, in a pure binary fashion. As soon as the contents of the BCD counter have become zero which is detected by the NULL detector, both AND gates in fig. 8.9 are closed, and counting is stopped; the binary chain now contains the pure binary equivalent of the value initially set in the BCD chain.

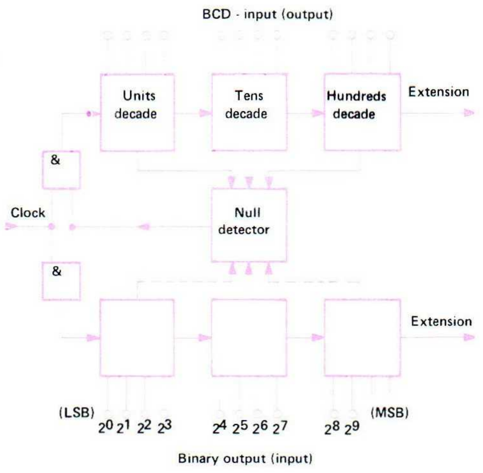
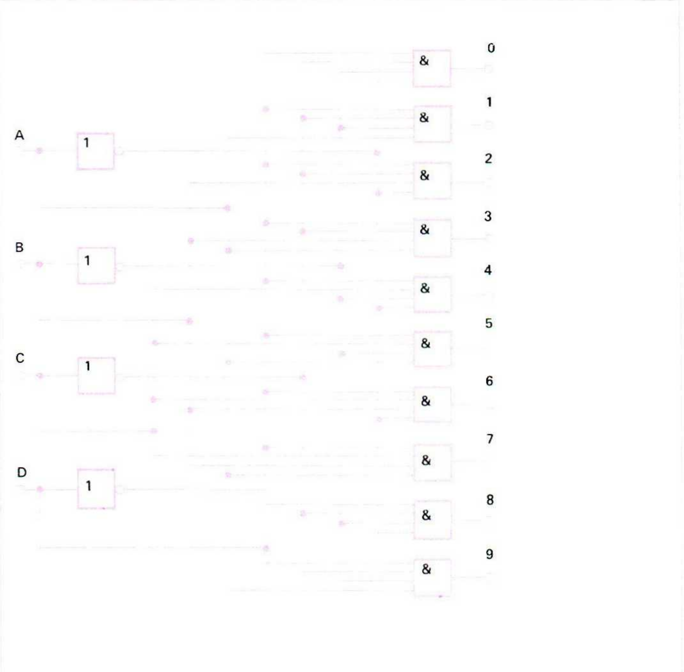


Fig. 8.9 BCD-to-binary converter (or vice versa).

The mechanism of the conversion from binary to BCD is indicated in fig. 8.9 by the dotted lines; in this case the BCD chain will be a 1, 2, 4, 8 "up" counter and the binary will be a "down" counter.
As with the S/P converters, the similarity of the two BCD-to-binary and binary-to-BCD converters often leads to their being combined in one circuit.

• BCD-to-decimal converter
In many cases BCD information has to be converted into decimal form, e.g. for driving numerical indicator tubes (NIT's). Special decoders have been developed for this purpose. The logic diagram and truth table of such a decoder are given in fig. 8.10.



Inputs				Outputs										Decimal
A	B	C	D	1	2	2	3	4	5	6	7	8	9	
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	0	0	0	0	0	0	0	0	1
0	1	0	0	0	0	1	0	0	0	0	0	0	0	2
1	1	0	0	0	0	0	1	0	0	0	0	0	0	3
0	0	1	0	0	0	0	0	1	0	0	0	0	0	4
1	0	1	0	0	0	0	0	0	1	0	0	0	0	5
0	1	1	0	0	0	0	0	0	0	1	0	0	0	6
1	1	1	0	0	0	0	0	0	0	0	1	0	0	7
0	0	0	1	0	0	0	0	0	0	0	0	1	0	8
1	0	0	1	0	0	0	0	0	0	0	0	0	1	9
0	1	0	1	0	0	0	0	0	0	0	0	0	0	
1	1	0	1	0	0	0	0	0	0	0	0	0	0	
0	0	1	1	0	0	0	0	0	0	0	0	0	0	
1	0	1	1	0	0	0	0	0	0	0	0	0	0	
0	1	1	1	0	0	0	0	0	0	0	0	0	0	
1	1	1	1	0	0	0	0	0	0	0	0	0	0	

Fig. 8.10 NBCD-to-decimal decoder with truth table.

The BCD value applied to inputs A, B, C and D is decoded and will cause one of the decimal outputs to become high. Let us assume by way of example that the information presented at the input is the BCD value for 6, so A and D will be low and B and C will be high. When D is low, the inverter following D will make the signal high again and the upper input of gate 6 will be high. Input C, which is high, is directly connected to the second input of gate 6 as is the case with input B. Finally input A will also give a high at gate 6 after inversion, resulting in 4 highs at the AND gate, so the output will be high and all other outputs will remain low, because at least one of their inputs will be low.
As can be seen from fig. 8.11, a high level at the decoder's output will open the appropriate driving transistor for the NIT, causing ignition of the proper cathode; in the present case, the digit 6 will light up. In modern IC's the drive transistors (10 of them) are incorporated in de decoder, giving direct compatibility between the BCD code and the numerical indicator tube. A specific feature of the decoder circuit shown in fig. 8.10 is the fact that the forbidden states of the BCD code (positions 10 to 15) are recognised and all outputs are then held low (see also truth table).

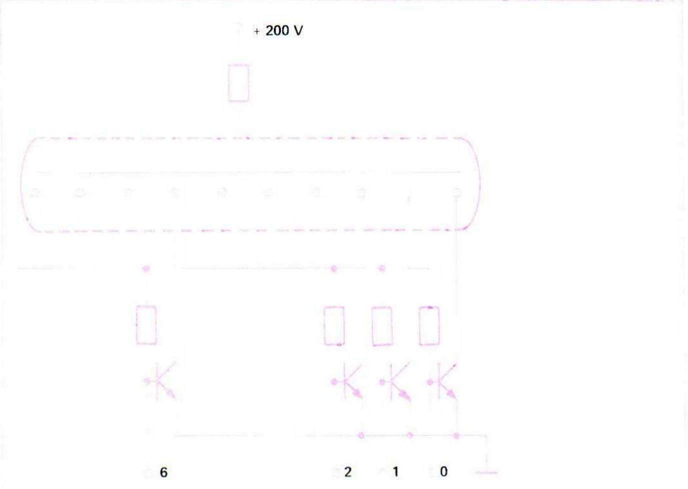
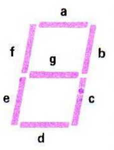


Fig. 8.11 Driving circuit for numerical indicator tubes.

The "seven-segment display" (fig. 8.12) is another type of display which is very popular nowadays. The numerical indicator contains 7 bars positioned in such a way that the required figure can be displayed by selection of the appropriate bars. For example, when the decimal digit 4 has to be shown bars b, c, f and g are illuminated. Of course, a special decoder is required for driving such an indicator from a BCD code. As a BCD-to-decimal decoder already exists, a decimal-to-7-segment decoder would be sufficient – but it is simpler to combine the two converters into one. The logic diagram of such a decoder is shown in fig. 8.13, and its truth table is given in fig. 8.12. These are self explanatory. In the above example an NBCD code is used, but of course any BCD code can be decoded in a similar way.



Decimal	D	C	B	A	a	b	c	d	e	f	g	Display
0	0	0	0	0	1	1	1	1	1	1	0	0
1	0	0	0	1	0	1	1	0	0	0	0	1
2	0	0	1	0	1	1	0	1	1	0	1	2
3	0	0	1	1	1	1	1	1	0	0	1	3
4	0	1	0	0	0	1	1	0	0	1	1	4
5	0	1	0	1	1	0	1	1	0	1	1	5
6	0	1	1	0	0	0	1	1	1	1	1	6
7	0	1	1	1	1	1	1	0	0	0	0	7
8	1	0	0	0	1	1	1	1	1	1	1	8
9	1	0	0	1	1	1	1	0	0	1	1	9

Fig. 8.12 Configuration of the bars in the 7-segment display with truth table.

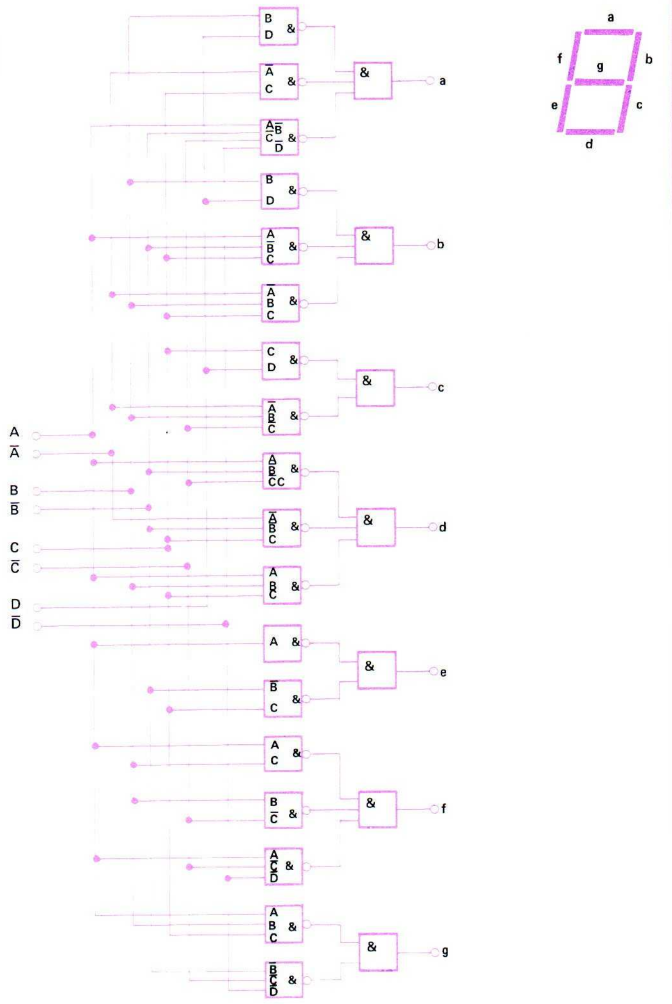


Fig. 8.13 BCD (1, 2, 4, 8 code) to 7-segment converter.

- Decimal-to-BCD converters

After the detailed discussion of the previous sub-section, little remains to be said here, see also chapter 4.
A decimal-to-NBCD converter is shown in fig. 8.14, which is self-explanatory. Circuits converting from decimal to any other type of BCD code can be made in a similar way.

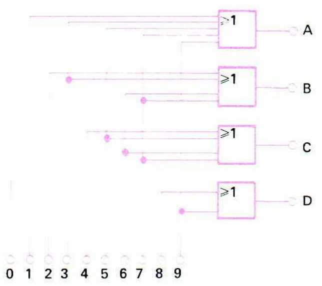


Fig. 8.14 Decimal to NBCD converter.

- Digital-to-analog converter

It is often necessary to change binary or BCD data into an analog signal. Where output data is concerned, this may be needed e.g. to make the signal suitable for driving a recorder, while with input data it may be needed e.g. for controlling the frequency range of a signal generator. The simplest form of a digital-to-analog converter (DAC) is illustrated in fig. 8.15. The summing resistors of an operational amplifier are weighted in a binary fashion (i.e. $R_1:R_2:R_3... = 1:2:4...$ and all are connected via an electronic switch (logic gate) to a reference voltage or to ground.

A logic "1" applied to a given input connects the corresponding resistor to the reference voltage and increases the output voltage by the binary-weighted increment in question. When all inputs are "1", the output is maximum.

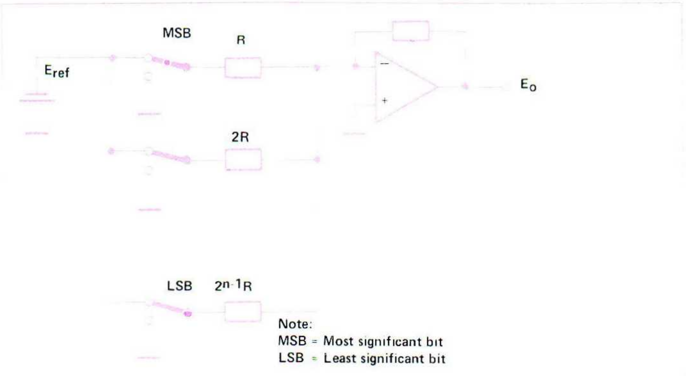


Fig. 8.15 Simplified binary weighted resistor DAC.

A big disadvantage of this simple weighted-resistor approach is that the value of each input resistance is twice the preceeding one, so absolute values become quite large. Furthermore, because the resistances are spread over such a wide scale, is becomes rather difficult to match the resistors in physical properties such as tolerance, temperature stability and so on, so that it is no simple matter to design a reasonably stable, accurate converter on this basis.
One way of overcoming these problems is to make use of the "R-2R ladder" network illustrated in fig. 8.16. This works as follows:
If one "leg" of the ladder is connected to the reference voltage by means of the electronic switch and the remaining "legs" are grounded, a current is produced in the first-mentioned leg and travels through the ladder, being divided by a factor of two at each junction.
The current contribution from the "leg" (e.g. bit) in question at the summing input of the operational amplifier in thus binary-weighted in accordance with the number of junctions through which it passes; hence the LSB (least significant bit) is on the far left of the circuit shown.
A very specific feature of the ladder converter is that the input resistance of the operational amplifier is independent of the binary word.
The above two converter types are pure binary. It is of course easily possible to adapt the circuits to BCD logic. In the case of the weighted-resistor arrangement, we only have to change the resistances and not the basic configuration (fig. 8.17). The R-2R ladder may be adapted to BCD by using R-2R subsections for each decade and summing these in parallel, with additional decade-weighted resistors (fig. 8.18).
It should be noted that a 3-decade BCD–DAC (with 12 lines) has a basic resolution of 0.1% (1 part in 1000). In a pure binary set-up, the same number of lines would give a resolution of 0.025% ($2^{12} = 4096$).
This illustrates once again that pure binary coding makes more efficient use of the available capacity than BCD (which as we know is caused by the 6 unused code combinations of BCD).

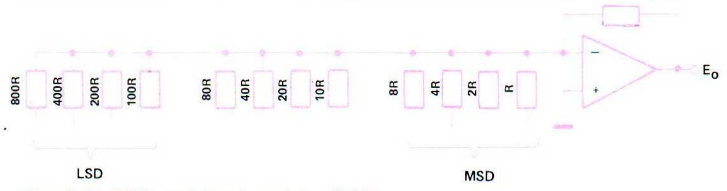


Fig. 8.17 BCD weighted-resistor DAC.

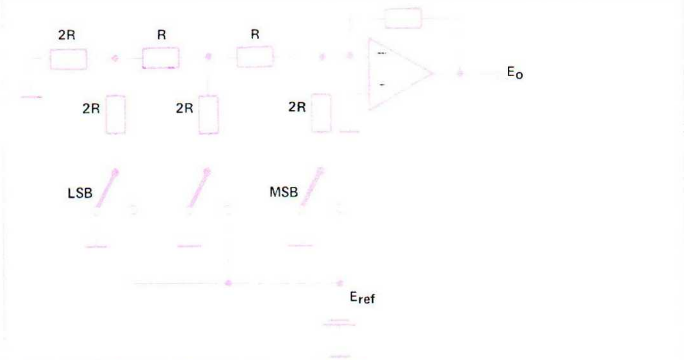


Fig. 8.16 Binary R-2R "ladder" DAC.

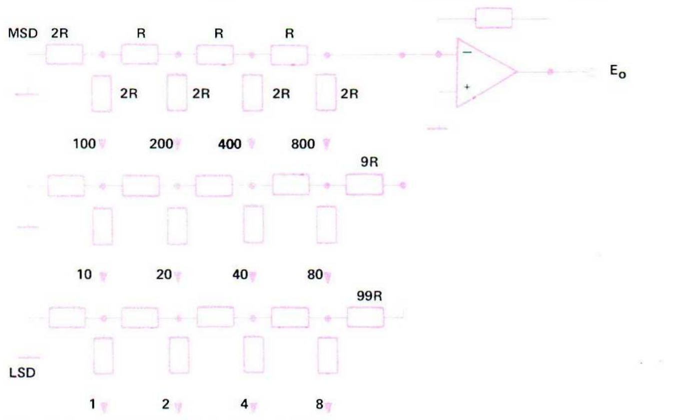


Fig. 8.18 BCD R-2R "ladder" DAC.

• Analog-to-digital converter (ADC)
As most physical or electrical properties which need to be measured or analysed are in analog form, we will need a converter for making analog data suitable for processing in digital systems.
The digital voltmeter (DVM) is one of the most important examples of analog-to-digital converters. A detailed description of this instrument will be found in Part III of this course. However, for the sake of completness we shall give here a brief description of one of the most popular circuits used in DVM's, that is based on the dual-slope integration method. (fig. 8.19).

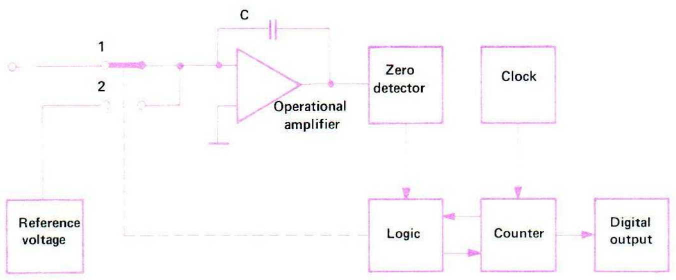


Fig. 8.19 ADC with dual-slope integration.

When the switch is in position 1 for a fixed time T, the input voltage V_i will be integrated as a charge in capacitor C; V_c is thus a linear function of V_i.
After this fixed integration time, switch 1 is opened and switch 2 is closed so as to connect V_{ref} to the input of the operational amplifier. However, care must be taken to ensure that V_{ref} is of opposite polarity to V_i. The capacitor C will now be discharged with a constant current (since V_{ref} is constant). The time t needed to discharge C completely is thus a linear function of V_c and hence also of V_i.
In practice, the circuit operates as follows. On receipt of a start pulse switch 1 is closed and the counter starts counting 1000 clock pulses (fig. 8.20).
Upward integration in capacitor C is started with a slope depending on V_i. After the 1000 pulses, the switches are reset so that V_{ref} is connected to the amplifier input while the counter (which has been reset to 0) is started again. The capacitor is now discharged with a constant slope. As soon as the zero detector at the output finds that V_c is zero, the counter is stopped. The pulses counted by the counter in this way thus have a direct relation with the input voltage. If a BCD counter is used the original analog value is converted into BCD code, while a pure binary counter gives conversion into binary code.

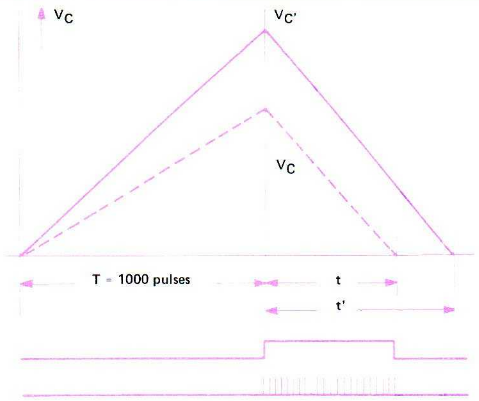


Fig. 8.20 Time diagram of dual-slope ADC.

Measuring systems

As we mentioned above, the great advantage of digital signals is the ease of building test systems which use such signals. In view of the present trend from manual testing and measurement via semi-automatic systems to the use of fully automatic and computer-controlled systems, it would seem to be worth while to conclude this book with a short description of the various systems involved.
In general, automated electronic measurement systems can be defined as a combination of a number of electronic instruments, controlled by some central unit. This central unit can not only control the operation of the individual instruments but also determine the sequence of the measurements more or less automatically, and make possible the operations of
a. sorting
b. recording
c. calculating

Kinds of automated measuring systems
We can sub-divide automated measuring systems into three groups:
• Programmer-operated systems
These are preset test and measuring set-ups with partly manual control or operation. The various test programmes are preset manually (e.g. by means of a diode matrix) in the central unit (programmer) and are initiated by a push-button or by automatic sequencing.
• Controller-operated systems
These test and measuring systems have simple processing facilities (e.g. sorting and recording), but no calculation facilities. The various test programmes are stored on punched tape or another medium in the central unit ("controller").

• Computer-operated systems
These fully automated measuring systems have all processing facilities, e.g.
a. sorting
b. recording
c. computing
The various test programmes are stored in a computer memory, or on punched tape.

Organisations of automated measuring systems
• Star system (fig. 8.21)*

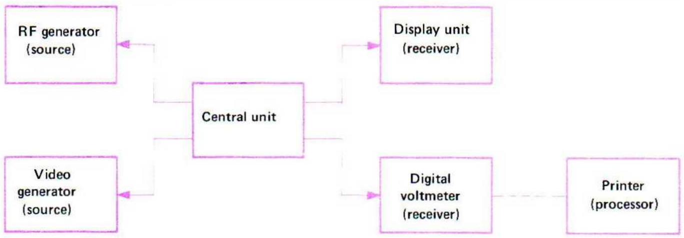


Fig. 8.21 Principle of a STAR system

Each instrument is individually connected to the central unit by its own bundle of control lines.
In STAR systems, the programming information is given as levels; or in other words, for each code a continous control level is applied as long as a certain function is required. A "programmer" (programme board) is often used for presetting the programme of operation in such systems. Inserting diode pins at the appropriate spot in the programme board closes a contact to ground, thus giving the right voltage level at the right place in the system. Further, a programmer normally carries a number of push-buttons, each of which activates a complete test programme by controlling a number of lines (e.g. 64). It will often also have a number of analog control lines by means of which continuous functions can easily be set. The input and output information of a STAR-programmed instrument is given simultaneously via a separate input and output connector.

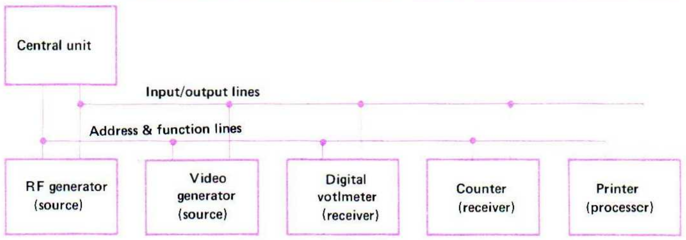


Fig. 8.22 Principle of a BUS system

• BUS-line system (fig. 8.22)**
All instruments are connected to the central unit via a common set of lines.
For each test, the instrument is "activated" by means of the appropriate INPUT/OUTPUT LINES (I/O lines) and the ADDRESS and FUNCTION lines.
In BUS-line systems, the programming information is given as a number of pulses. Because of the short duration of these pulses, the programming information must be stored in the instrument to be programmed by means of a "memory". This memory retains the information as long as the functional setting of the instrument is required. When a new setting is required the memory is first reset and then new code is read in.
Because all instruments to be programmed are connected in parallel to the DATA BUS line, measures have to be taken to ensure that only one specific instrument is set during each run of coded information, while the next run of pulse codes can set another instrument. For this purpose each instrument has its own ADDRESS, given via a separate ADDRESS BUS LINE. Only when the instrument recognises its own address will it read the information on the DATA bus. The DATA bus line is generally designed as a DUAL DATA BUS, in other words it can be used for both input and output information depending on the instrument concerned and on the code given on the ADDRESS bus.
This sequential information provided makes the BUS-line system ideal for computer or punched-tape control.

* An example of a STAR-operated system is given in Philips test and measuring notes 1971/3.
** Also called "Party-line" system, "Serial system", "Highway system" of "Dataway system" in the literature.

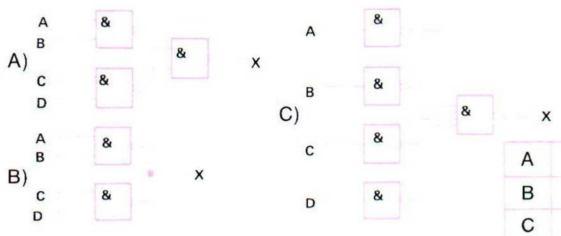
Questions

Q.8.1 As a converter from negative to positive logic can be used.

A An AND gate
B A NOT gate
C An OR gate

A	
B	
C	

Q.8.2 In a reversible shift register (fig. 6, 18) the Boolean function X = AB + CD is needed. This can be realised with only NAND gates in the following way:

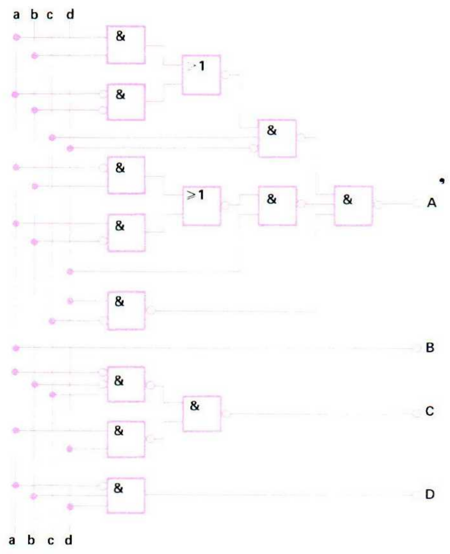


A	
B	
C	

Q.8.3 The converter circuit below is:

A An NBCD-to-1242 code converter
B An Excess 3 – NBCD code converter
C A XS-3 GRAY – NBCD code converter

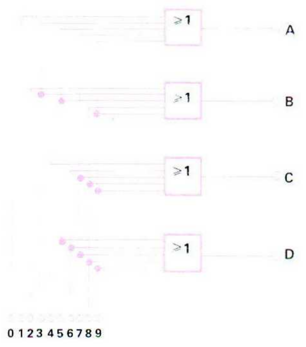
A	
B	
C	



Q.8.4 The converter circuit below is:

A A decimal-to-NBCD code converter
B A decimal-to-1242 code converter
C An NBCD-to-decimal code converter

A	
B	
C	



Q.8.5 A 6-bit binary number has to be converted into XS-3 Gray BCD code. The number of output lines required is:

A 2
B 6
C 8

A	
B	
C	

Q.8.6 In negative logic the code 0110 stands for:

A 6
B 9
C Not defined

A	
B	
C	

Q.8.7 The BCD code 0001/0000/0000 (decimal 100) is put in the BCD to binary converter of fig. 8.9. After how many clock pulses is the number converted into pure-binary!

A 1
B 12
C 100

A	
B	
C	

Q.8.8 A 3-digit 1242-coded number has to be converted into an analog signal. The proper way to do this is

A Design a proper DAC
B Is not posible
C First convert into NBCD and then use a DAC

A	
B	
C	

Q.8.9 N × P bits of data have to be transported in the word (N) serial, bit (P) parallel mode. The transfer speed is then:

A N times faster than in full serial transport
B P times faster than in full serial transport
C N times slower than in full serial transport

A	
B	
C	

Glossary of terms

AND gate – A binary circuit having two or more inputs and a single output in which the output is ON (1) only if all inputs are ON (1) together, and is OFF (0) if any one of the inputs is OFF (0).

BINARY SYSTEM – A system for mathematical computation based on the scale of 2, or a system in which all stages can only have one of two possible states.

BINARY CODED DECIMAL – Four bits of binary information can be used to encode one decimal digit. When a decimal digit is encoded in this way it is called a Binary Coded Decimal (BCD).

BIT – The words "binary digit" are often abbreviated to BIT.

CLOCKED RS FLIP-FLOP – The clocked RS flip-flop has two conditioning inputs which control the state to which the flip-flop will go at the arrival of the clock pulse. If the S (Set) input is enabled, the flip-flop goes to the "1" state when clocked. If the R (Reset) input is enabled, the flip-flop goes to the "0" state when clocked. The clock pulse is required to change the state of the flip-flop.

COMPARATOR – A comparator is a device used to determine whether two numbers of bits of information are equal.

C-MOS – Complementary MOS. A MOST or IC involving both P-channel and N-channel MOS-FETS.

CML – Current-mode logic. Basically equivalent to ECL.

COUNTER – A counter is a device which will maintain a continuous record of the number of pulses which it has received at its input. The output of the counter indicates the sum of the number of input pulses.

CTL – Complementary Transistor Logic. A logic system using emitter-coupled circuits with a combination of PNP and NPN transistors.

CUT-OFF – The condition when the emitter junction of a transistor is at zero voltage or is reverse biased so that no collector current flows.

DCTL – Direct-Coupled Transistor Logic. A system of transistor logic in which the collector output of one gate is connected directly to the base input of the next gate.

DTL – Diode-Transistor Logic. A logic system in which the logic decisions are carried out by a group of diodes and the resulting output coupled through a transistor output stage.

D-TYPE FLIP-FLOP – A D-type flip-flop will propagate whatever information is at its D (data) conditioning input prior to the clock pulse, to the Q output, on receipt of a clock pulse.

DECODER – A decoder is a device used to convert information from a coded form into a more usable form (e.g., binary-to-decimal decoder).

DIGIT – A digit is one character in a number. These are 10 digits in the decimal number system. There are two digits in the binary number system.

ECL – Emitter-Coupled Logic – A logic system using emitter-coupled transistor circuits.

ENCODER – An encoder is a device which takes information in one code and encodes it into another (e.g. BCD-to-binary encoder).

EXCLUSIVE OR – The Exclusive OR function is valid or its value is 1, if one and only one of the input variables is present. The Exclusive OR applied to two variables is present, or 1, if the two binary input variables are different.

FAN-IN – The number of inputs connected to a logic gate.

FAN-OUT -- The fan-out of an output is the number of unit loads it can drive.

FLIP-FLOP – A flip-flop is a storage device which can be used to retain one bit of information. A flip-flop can be in the "1" state or the "0" state. In the "1" state, its Q output presents a HIGH level and its $\overline{Q}$ output a LOW level. In the "0" state, its Q output presents a LOW level and its $\overline{Q}$ output a HIGH level.

FET – Field-effect transistor, a voltage-controlled semiconductor analogous to a triode.

GATE – A logic circuit having two or more inputs and a single output designed to give an output signal only when a certain combination of input signals exists.

HEXADECIMAL – The number system which has 16 distinct digits viz: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A. B, C, D, E, F.

HNIL – High-noise-immunity logic. Closely resembling DTL, the basic difference being that HNIL uses a Zener diode for stand-off (raising the threshold voltage).

INVERTER – An inverter (NOT gate) is a device which performs the operation of inversion. It will present at its output the inverse or complement of the information at its input.

JK FLIP-FLOP – A JK flip-flop has two conditioning groups of inputs (J and K) and a clock input. If both J and K inputs are LOW prior to the clock pulse it will remain in its initial condition when the clock pulse appears. If the J input is HIGH and the K input LOW, the flip-flop will go to the "1" state on receipt of the clock pulse; when J is LOW and K is HIGH, the flip-flop will switch to the "0" position at the clock pulse. When both J and K are HIGH the flip-flop will complement its initial state.

LOGIC – In computer language logic is a form of mathematics based upon two-state truth tables. Electronic logic uses two-state gates and flip-flops to perform decision-making functions.

MASTER-SLAVE – A master-slave flip-flop is one which contains two flip-flop, a master flip-flop and a slave flip-flop. The master flip-flop receives its information during the leading edge of a clock pulse and the slave or output flip-flop receives its information during the trailing edge of the pulse.

MOS-FET – Metal-oxide-semiconductor field-effect transistor which consist of source and drain regions on either side of a P-type of N-type channel plus a gate electrode insulated from the channel by silicon dioxide.

NAND GATE – A NAND gate is enabled when both its inputs are present or HIGH. When a NAND gate is enabled, its output is LOW. The term NAND is a contraction of the words NOT AND.

NOISE MARGIN or noise immunity is a critical IC parameter. It is the difference between the normal operating logic levels and the threshold voltage.

NOR GATE – A combination of a NOT and an OR circuit. A binary circuit having two or more inputs and a single output, in which the output is OFF (0) if any one of the inputs is ON (1) and is ON (1) only if all inputs are OFF (0) together.

NOT CIRCUIT – A binary circuit having a single input and a single output, in which the output is always the opposite of the input. When the input is ON (1) the output is OFF (0) and vice versa. This circuit is also called an inverter circuit.

OCTAL – The octal number system is one which has 8 distinct digits, namely, 0, 1, 2, 3, 4, 5, 6, 7.

ONE'S COMPLEMENT – One's complement arithmetic provides a method of negating a binary number so that binary subtraction can be performed using addition techniques. To obtain the 1's complement of a binary number, all bits in that number must be complemented. (i.e. 1's changed into 0's and vice versa).

OR GATE – A binary circuit having two or more inputs and a single output, in which the output is ON (1) if any one of the inputs is ON (1), and is OFF (0) only if all inputs are OFF (0) together.

POWER DISSIPATION OF A LOGIC CIRCUIT – The supply power when a logic circuit is operating with a 50% duty cycle, i.e. when it is in the 0 state half of the time and in the 1 state the other half of the time.

PROPAGATION DELAY – The time delay between the application of a signal to the input of a logic circuit and the change of state at the output.

RCTL – Resistor-Capacitor-Transistor Logic. A variant of resistor transistor logic in which a capacitor is connected across the series resistor to permit faster switching.

RESET – If a Reset input to a flip-flop is enabled, the flip-flop will go to the "0" state.

RS FLIP-FLOP – The RS flip-flop has two inputs, a Set input and a Reset input. If the Set input is enabled (HIGH), the flip-flop goes to the "1" state. If the Reset input is enabled (HIGH), the flip-flop goes to the "0" state.

RTL – Resistor-Transistor Logic – A system of transistor logic in which a resistor is included in series with the base of each transistor in order to reduce differences in transistor currents.

SATURATION – A transistor is saturated when a further increase of base current causes no further increase in collector current.

SET INPUT – When the Set input to a flip-flop is enabled, the flip-flop goes to the "1" state.

SHIFT REGISTER – A shift register can contain several bits of information. When a shift introduction is received, all the information in the register is shifted one place.

THRESHOLD VOLTAGE – The input voltage level at which a binary logic circuit changes from one state to the other.

TTL – Transistor-Transistor Logic. A logic system similar to diode-transistor logic in which the logic diodes are replaced by a multi-emitter transistor.

Answers to questions

Questions	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9	1.10
Answers	B	B	C	C	B	B	B	C	C	A

A.1.1 As we know, with 4 bits we can count to 15, so for 17 we need one bit more: 10001. Answer B is correct.

A.1.2 The "weight" of the first binary digit is 1 (odd), all other binary digits are even, so when the least significant digit is present the decimal number is **always** odd, when the LSD is not present (0) the number is thus **always even.** Answer B.

A.1.3 Answer C is correct. (The other two possibilities even contain illegitimate codes!)

A.1.4 The BCD notation is read in the same way as we read our decimal number. 619 (C) is thus the correct answer.

A.1.5 The (r − 1)'s complement of N is: $r^n - 1 - N = M$
The (r − 1)'s complement of M is: $r^n - 1 - M = r^n - 1 - (r^n - 1 - N) = N$
(Answer B)

A.1.6 715_8 in decimal is $7 \times 8^2 + 1 \times 8 + 5 = 461_{10}$
The 10's complement of 461_{10} is 539 (Answer B)

A.1.7 The sum of the weight in a self-complementing BCD code must be nine, because the 9's complement of 0000 is 1111 (answer B).

A.1.8 Four bits in combination provide a maximum of sixteen configurations or states. Only ten of these states are required for decimal coding so six of them must be discarded. Thus the number of possible four-bit BCD codes is 16!/6! or approx. 2.9×10^{10}. Many of these codes are reflections of others, but if these are eliminated the number of codes is still greater than 10^9. Answer C is thus correct.

A.1.9 The total number of characters which have to be encoded is 26 + 10 + 10 = 46. So one needs at least 6 bits ($2^5 = 32$; $2^6 = 64$); answer C.

A.1.10 The correct answer is A, only one wire is needed. It certainly is not faster because parallel coding transmits all bits in one single time interval and serial coding one after the other. The BCD coding of course has nothing to do with the transportation method.

Questions	2.1	2.2	2.3	2.4	2.5	2.6	2.7	2.8
Answers	A	C	B	C	B	C	B	C

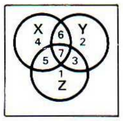
A.2.1 The required expression involves the proposition that X and Y and Z are all true, i.e. it is:
$S = XYZ$ (answer A) (areas 7 in fig. 2.5.)

A.2.2 The class of all digital instruments is Y (Y is true), that of non-measuring instruments is $\overline{X}$ (X is false) and that of instruments without battery supply is $\overline{Z}$ (Z is false) so the required function is:
$R = \overline{X}Y\overline{Z}$ (answer C) (area 2 in fig. 2.5).

A.2.3 All electronic measurements are in class X (X is true). The expression for all non-digital instruments ($\overline{Y}$) with battery supply (Z) is $\overline{Y}Z$.
The total function is thus
$Q = X + \overline{Y}Z$ (answer B) (areas 1, 4, 5, 6 and 7 in fig. 2.5).

A.2.4 If $F_{XYZ} = \Sigma\ (1,3,7)$, then $\overline{F}_{(XYZ)} = \Sigma\ (0,2,4,5,6)$.
Inversion gives $\overline{\overline{F}}_{(XYZ)} = \Pi\,(0,2,4,5,6) = F_{XYZ}$.

A.2.5 The best way to solve this question is to give all the areas in the Venn diagram a number (as in fig. 2.5).
The function for area 5 is $X\overline{Y}Z$, that for area 3 is $\overline{X}YZ$, and that for area 6: $XY\overline{Z}$.



The total function required is obviously the union of the above 3 logic products, viz.:
$F\ (XYZ) = (XY\overline{Z} + \overline{X}YZ + X\overline{Y}Z)$ (answer B)

A.2.6 In this case, area 6 is $XY\overline{Z}$, area 7 is XYZ
Area 6 + 7 is $XY\overline{Z} + XYZ = XY\ (\overline{Z} + Z) = XY$
(which is correct because area 6 + 7 is the intersection of X AND Y)
Finally area 1 is $\overline{Y}\overline{Z}Z$
Thus areas $6 + 7 + 1 = XY + \overline{Y}\overline{Z}Z$ (answer C)

A.2.7 Here we just apply the appropriate rules of algebra:

$$
\begin{aligned}
\overline{F}\ (X, Y, Z) &= (X + Y + XY)\ (X + Z) \\
&= [X\ (1 + Y) + Y]\ (X + Z) = \\
&= (X + Y)\ (X + Z) = \\
&= XX + XZ + XY + YZ = \\
&= X + XZ + XY + YZ = \\
&= X\ (1 + Z) + XY + YZ = \\
&= X + XY + YZ = \\
&= X\ (1 + Y) + YZ = X + YZ \text{ (answer B)}
\end{aligned}
$$

A.2.8 $F\ (X, Y, Z) = (X + \overline{Y} + Z)\ (X + \overline{Y} + \overline{Z})\ (X + Y + Z)$
Applying De Morgan's theorem, we find

$$
\begin{aligned}
\overline{F}\ (X, Y, Z) &= (\overline{X}Y\overline{Z}) + (\overline{X}YZ) + (\overline{X}\overline{Y}\overline{Z}) = \\
&= \overline{X}Y\ (\overline{Z} + Z) + \overline{X}\overline{Y}\overline{Z} = \\
&= \overline{X}Y\ 1 \quad + \overline{X}\overline{Y}\overline{Z} = \\
&= \overline{X}Y \quad + \overline{X}\overline{Y}\overline{Z}
\end{aligned}
$$

Complementing again gives:

$$
\begin{aligned}
F\ (X, Y, Z) &= (X + \overline{Y})\ (X + Y + Z) = \\
&= X + \overline{Y}\ (Y + Z) = \\
&= X + \overline{Y}Y + \overline{Y}Z = \\
&= X + \overline{Y}Z, \text{ (answer C)}
\end{aligned}
$$

Questions	3.1	3.2	3.3	3.4	3.5	3.6	3.7	3.8	3.9	3.10	3.11
Answer	A	C	B	B	B	A	B	B	B	C	C

A.3.1 A is the correct answer: only when all switches are closed (all inputs to the AND gate are "1") is the lamp on.

A.3.2 The output of the EXCLUSIVE-OR gate is "1" (high) if one and only one of its inputs is "1" (high) which happens in time intervals 2, 3 5, 6, 8 and 9; so C is the correct answer.

A.3.3 Diagram B is the correct logic diagram; the OR gate can be considered as a switch in parallel with the AND gate (series switches).

A.3.4 The best way to solve this problem is to make a truth table for both circuits. Comparison of these truth tables will show that both circuits are identical (answer B is thus the right one). With Boolean algebra:
Circuit 1: $a+\bar{a}b=a+b$ and Circuit 2: $\overline{\bar{a}\bar{b}}=a+b$

a	b	X_1	X_2
0	0	0	0
0	1	1	1
1	0	1	1
1	1	1	1

A.3.5 The best way to solve this problem is to make a truth table and compare it with those of the exclusive-OR, comparator and inhibit gates. Another approach is to find the logic expression of all the circuits involved (see A 3.6.). In both ways one finds that the circuit is a comparator, so answer B, is correct.

A.3.6 The Boolean function of the upper AND gate is $A\bar{B}$ and of the lower one $\bar{A}B$. Both outputs are logically added and inverted so $\overline{A\bar{B}+\bar{A}B}$.
Applying de Morgan's rules gives:
$A\bar{B}+\bar{A}B=(\bar{A}+B)\ (A+\bar{B})=$
$A\bar{A}+\bar{A}\bar{B}+AB+B\bar{B}=AB+\bar{A}\bar{B}$
which is the Boolean function of a two-input comparator gate, so answer A is correct.

A.3.7 In order to solve this, first find the Boolean function which can be done as described in chapter 2, by logic addition of the functions of the various shaded areas, then simplify the expression found:

$$\begin{aligned} F=\Sigma 1, 4, 5, 6, 7 &= X\bar{Y}\bar{Z}+XY\bar{Z}+X\bar{Y}Z+XYZ+\bar{X}\bar{Y}Z= \\ &= X\bar{Z}\ (\bar{Y}+Y)+XZ\ (\bar{Y}+Y)+\bar{X}\bar{Y}Z= \\ &= X\bar{Z}+XZ+\bar{X}\bar{Y}Z=X\ (\bar{Z}+Z)+\bar{X}\bar{Y}Z= \\ &= X+\bar{X}\bar{Y}Z=X+\bar{Y}Z \end{aligned}$$

of which circuit B is the equivalent circuit.

A.3.8 When C is "1" X is also "1", this is also so when the output of the NAND gate is "1". Only when both inputs (a and b) to the NAND gate are "1" can output (X) be "0", which happens when C is also "0". B is thus the correct answer.

A.3.9 With reference to fig. 2.5 we see that the Boolean function $\Sigma\,(0, 1, 2, 3, 4)$ can be written

$$\begin{aligned} F\ (XYZ) &= \bar{X}\bar{Y}\bar{Z}+\bar{X}\bar{Y}Z+\bar{X}Y\bar{Z}+\bar{X}YZ+X\bar{Y}\bar{Z}= \\ &= \bar{X}\bar{Y}(\bar{Z}+Z)+\bar{X}Y(\bar{Z}+Z)+X\bar{Y}\bar{Z}= \\ &= \bar{X}\bar{Y}+\bar{X}Y+X\bar{Y}\bar{Z}=\bar{X}(\bar{Y}+Y)+X\bar{Y}\bar{Z}=\bar{X}+X\bar{Y}\bar{Z}=\bar{X}+\bar{Y}\bar{Z} \end{aligned}$$

Which is the configuration of answer B.

A.3.10 We can answer this question by working out the Boolean expression and then simplifying it as much as possible.

$$\begin{aligned} F\ (X.Y.Z.) &= (X+Y+XY)\ (X+Z)= \\ &= XX+XZ+XY+YZ+XXY+XYZ= \\ &= X+XZ+XY+YZ+XY+XYZ= \\ &= X+XZ+XY+YZ+XYZ= \\ &= X\ (1+Z)+XY+YZ+XYZ= \\ &= X+XY+YZ+XYZ= \\ &= X\ (1+Y)+YZ(1+X)=X+YZ, \end{aligned}$$

which is the circuit of answer C.

A.3.11 From the truth table one can see that the output is always "1" when b = "1", independent of a and c; which is only the case in logic diagram C; this thus represents the correct answer.

Questions:	4.1	4.2	4.3	4.4	4.5	4.6	4.7
Answers	A	A	B	C	B	A	A

A.4.1 As there is no less significant digit, there cannot be a carry so C_0 should be logic "0" (Answer A) (Of course, the type of logic has nothing to do with the problem).

A.4.2 The appearance of a carry in a subtraction with r's complement means that the result is positive and correct (Answer A). In (r − 1)'s complement subtraction, the carry "1" has to be added to the result. The result has to be complemented when there is no carry.

A.4.3 As we learned in this chapter, the Boolean-functions for a sum and a difference are identical. The carry function is $C = A.B$, and the borrow function: $B = \bar{A}.B$. (Where A is the subtrahend), so A should be inverted to turn a half-adder into a half-subtractor. (Answer B)

A.4.4 Three 4-bit inputs means 12 connections, one four-bit output brings the number to 16 and 2 select lines makes the total 18. (Answer C)

A.4.5 It cannot be an octal-to-NBCD converter because it has only 4 inputs instead of 8, neither can it be an NBCD-to-1242 converter because the A bit is inverted. So, it must be a 9's complement (Answer B).
This can be verified by reference to the Boolean equations and truth table of the 9's complement given below

	NBCD				9's compl.			
0	0	0	0	0	1	0	0	1
1	0	0	0	1	1	0	0	0
2	0	0	1	0	0	1	1	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	0	1
5	0	1	0	1	0	1	0	0
6	0	1	1	0	0	0	1	1
7	0	1	1	1	0	0	1	0
8	1	0	0	0	0	0	0	1
9	1	0	0	1	0	0	0	0

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

A.4.6 It is of course not a multiplexer. (It has two inputs and two outputs) nor is it a half-adder (because of the symmetry of the circuit), so it must be a two-bit compator (Answer A). This is confirmed by the Boolean function and truth table below:

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

A	B	X	Y	
0	0	0	0	A = B
1	0	1	0	A > B
0	1	0	1	A < B
1	1	0	0	A = B

A.4.7 As we have seen, there is no difference between the Boolean Functions of the S and D outputs of half (and full) adders and subtractors
$S = D = A\overline{B} + \overline{A}B$ (half-adders/subtractors)
$X = S = D = ABC + A\overline{BC} + \overline{A}B\overline{C} + \overline{AB}C$ (full adders/subtractors)
so, answer A is correct.

Questions	5.1	5.2	5.3	5.4	5.5	5.6	5.7	5.8	5.9	5.10
Answers	C	B	C	A	A	A	B	C	B	B

A.5.1 When S goes HIGH Q goes HIGH too, and remains so until R goes HIGH. This is only the case with answer C.

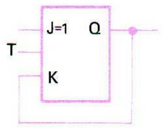
A.5.2 When T receives a "1", Q also becomes "1"; while when T goes LOW again nothing changes at the output. However, when T goes HIGH again the flip-flop switches over and Q goes LOW as is shown in answer B.

A.5.3 Inverters are always needed for making flip-flops, so one needs either NAND or NOR gates; AND's alone are not enough (answer C.).

A.5.4 As can be seen from the truth table, when all J and K's are connected together the flip-flop changes state with each T pulse. When the K inputs are connected to 0 the flip-flop remains in the Q = "1" position (Answer A).

A.5.5 A D flip-flop will transfer the data present at the D input to the Q output on receipt of the clock pulse. So assuming initially Q = 0 and $\overline{Q}$ = "1", then at the first clock pulse (D = 1) Q becomes "1" and $\overline{Q}$ = D = "0". At the next clock pulse Q goes to "0" again, and so on, so answer A is correct.

A.5.6 The circuit described is shown below. A truth table shows the answer (A):



J	K	T → Q	
1	0	1	1
1	1	1	0
1	0	1	1
1	1	1	0

A.5.7 Suppose Q was initially "0", then a clock pulse will never make Q = "1" because J is also "0". Q can only become "1" when J is "1" which is only possible when Q is "1". So Q has always been "1". (Answer B.)

A.5.8 Both circuits are the same (answer C). In the A version the $\overline{Q}$ is coupled to the upper AND gate, and in version B the Q output is inverted and coupled to the upper AND gate.

A.5.9 The circuit will not work as a JK flip-flop but as a T flip-flop because the T pulse is passed directly through the input OR gates (independent of the status of J and K) to the AND gate, thus reducing the circuit to a normal T flip-flop (Answer B).

A.5.10 A counting table shows the situation clearly.

T-pulse	Q_1	$\overline{Q}_1$	Q_2
0	0	1	0
1	1	0	0
2	0	1	1
3	1	0	1

and so on.

The Q_2 output is a "1" so answer B is correct.

Questions	6.1	6.2	6.3	6.4	6.5	6.6	6.7	6.8	6.9	6.10	6.11
Answers	B	A	A	B	A	B	C	B	C	A	C

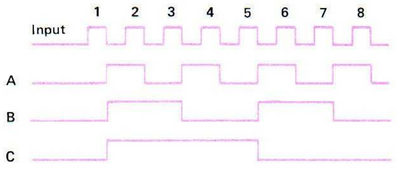
A.6.1 In the modulo-6 counter the count can never be 6 because the counter is then reset. Futhermore, as can be seen from the circuit diagram, flip-flop A toggles at every count, (at even counts A = 0 and at odd counts A = 1). Thus the count must be 5 (cannot be 4 or 6) which is confirmed by the truth table below (Answer B).

	C	B	A
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	0	0	0

A.6.2 This is of course a modulo-3 counter (Answer A), because there are not enough flip-flops for either a 3-bit shift register or a modulo-5 counter. The truth table below confirms this.

	B	A	J_B	J_A
0	0	0	0	1
1	0	1	1	1
2	1	0	0	0
3	0	0	0	1

A.6.3 We see clearly from the complete pulse diagram given below that answer A is the correct one.



A.6.4 The best way to see the correct answer immediately is to look at the sequence of the bits in the various columns. The right-hand column in each group A, B and C has the sequence 0 1 0 1 0 1 etc, the second from the right 0 0 1 1 0 0 1 1, number three 0 0 0 0 1 1 1 1 and so on. Only the bits in answer B follow this pattern.

A.6.5 The speed of this combined scaler is of course determined by the first scaler (n_1) because the output of this scaler is already n_1 times slower and will in general be slow enough for the next scaler. So answer A is correct.

A.6.6 Any modulo-6 counter has 8 − 6 = 2 illegitimate states, so answer B is correct.

A.6.7 The best way to solve this problem is to make a truth table:

Clock pulse	C	B	A
0	0	0	0
1	0	0	1
2	0	1	1
3	1	1	1
4	1	1	0
5	1	0	0
6	0	0	0

N.B. This illustrates another way of making a modulo-6 counter.

We see from this truth table that after the 6th pulse the shift register is "000" again, (Answer C).

A.6.8 The least significant bit is the left-hand one in the drawing. In binary notation this bit becomes the righ-hand one, thus the binary number is 110 011 100 111
which is in octal notation: 6 3 4 7
(Answer B)

A.6.9 The maximum count in binary notation is 111 111 111 111
which is in octal notation 7 7 7 7
and in decimal notation 4095
At pulse number 4096 the counter is reset to zero again; the maximum count is thus 4095, (Answer C).

A.6.10 First we have to make the truth table in order to find the illegitimate state.

Clock pulse	C	B	A
0	0	0	0
1	0	0	1
2	0	1	1
3	1	1	0
4	1	0	0
5	1	0	1
6	1	1	1
7	0	0	0

From this truth table we see that the code of the illegitimate state is 010.
When we put this value in the counter we see that after a clock pulse A will not change because $J_A = 0$ and $K_A = 1$, so C will not change either. Flip-flop B will not change because both J_B and K_B are "0", in other words the illegitimate state is locked and answer A is correct.

A.6.11 The best way to solve this problem is to make a truth table.

Count	X	$\overline{X}$	Y	$\overline{Y}$	Z	$\overline{Z}$
0	0	1	0	1	0	1
1	1	0	0	1	0	1
2	0	1	1	0	0	1
3	1	0	1	0	0	1
4	0	1	0	1	1	0
5	1	0	0	1	1	0
6	0	1	1	0	1	0
0	0	1	0	1	0	1

At count 6, flip-flop X already has the correct state, but flip-flops Y and Z have to be reset. Since both Y and Z are "1", the NAND gate of answer C will give a "0" output signal which is passed to R_Y and R_Z, thus resetting flip-flops Y and Z. C is thus the correct answer.
The gate of answer A would give the same output signal, but would pass it to the wrong flip-flops (X and Y would be reset, and Z would not be reset), while the configuration of answer B would reset the counter at the 7th count.

Questions	7.1	7.2	7.3	7.4	7.5	7.6	7.7	7.8
Answers	A	B	B	B	B	B	C	A

A.7.1 The diode gate in question is an AND gate (answer A) which can be very well seen from the truth table below.

A	B	X	Diodes
0	0	0	Both conduct
1	0	0	D_B conducts
0	1	0	D_A conducts
1	1	1	Both cut off

A.7.2 The circuit is a NAND gate (Answer B) because the output is LOW when any one transistor or both transistors conduct (Input LOW).

A.7.3 The correct answer is B. As a matter of fact, answer A is also correct. However, this solution has the disadvantage that all three inputs (tied together) could be connected to the output of another gate, thus loading the latter unnecessarily.

A.7.4 A positive pulse must be applied to the base of TS_1 (Answer B – set input) TS_1 is then cut off, causing the base voltage of TS_2 to drop; TS_2 then starts to conduct, making output Q (and hence the base of TS_2) more positive. This gives a new stable state.

A.7.5 The stand-off diodes are included to increase the threshold level (the voltage drop over the diodes have to be added to the switching signal).

A.7.6 Answer B is correct, as can be seen clearly from the table: 2.4 V is the minimum output voltage of a TTL gate in its HIGH state. A noise signal of more than 0.4 V added to the output brings the signal into the undefined region of the logic gate.

A.7.7 The basic HNIL gate is nothing but a DTL gate with a higher threshold voltage, which means that the basic HNIL gate is also a NAND (Answer C).

A.7.8 As we explained, keeping semiconductors out of saturation increases the switching speed which is one of the features of ECL; so answer A is the right one. (The emitter coupling features a larger fan-out: positive or negative logic is of course only a matter of definition and has nothing to do with the actual parameters).

Questions	8.1	8.2	8.3	8.4	8.5	8.6	8.7	8.8	8.9
Answers	B	A	C	B	C	A	C	A	B

A.8.1 Neither AND nor OR gate have an inversion function. This is only the case with the NOT gate, so answer B is correct.

A.8.2 The best way to solve this problem is to write out the Boolean functions of the three possibilities and apply De Morgan's rule:
A: $\overline{(\overline{AB})\,(\overline{CD})} = AB + CD$
B: $\overline{AB} + \overline{CD} = \overline{AB} + \overline{CD}$
C: $\overline{\bar{A}\bar{B}\bar{C}\bar{D}} \quad = A + B + C + D$ (is a 4-input OR gate)
So answer A is correct.

A.8.3 First write out the Boolean functions and simplify for A, B, C and D:

$$A=\overline{\overline{\{(\overline{ab+\bar a\bar b})\ c\bar d\}}\ \overline{\{(\overline{\bar ab+a\bar b})\ d\}}\ \overline{\{\bar cd\}}}=$$

$$A=\{(\overline{ab+\bar a\bar b})\ c\bar d+(\overline{\bar ab+a\bar b})\ d+\bar cd)=$$

$$A=(\bar a+\bar b)\ (a+b)\ c\bar d+(a+\bar b)\ (\bar a+b)\ d+\bar cd=$$
$$A=\bar abc\bar d+a\bar bc\bar d+abd+\bar a\bar bd+\bar cd$$
$$B=a$$
$$C=\overline{\overline{(\bar a.\bar b.c)}\ \overline{(\bar a.\bar d.)}}=\bar a.\bar b.c+a.d.$$
$$D=\bar a.b.d.$$

Then make the truth tables for NBCD, XS-3, Gray XS-3 and BCD 1242, fill in the variables of abc and d and compare. It will be seen that the converter is XS-3 Gray to NBCD (answer C, which is confirmed by the truth table below).

	XS-3 Gray				NBCD			
	d	c	b	a	D	C	B	A
0	0	0	1	0	0	0	0	0
1	0	1	1	0	0	0	0	1
2	0	1	1	1	0	0	1	0
3	0	1	0	1	0	0	1	1
4	0	1	0	0	0	1	0	0
5	1	1	0	0	0	1	0	1
6	1	1	0	1	0	1	1	0
7	1	1	1	1	0	1	1	1
8	1	1	1	0	1	0	0	0
9	1	0	1	0	1	0	0	1

A.8.4 It cannot of course be an NBCD-to-decimal converter, so it is either Decimal to NBCD or to 1242 code. The differences between NBCD and 1242 code are in the numbers above 4. So let's look at the output for Decimal 9, in which case all outputs are "1", so it must be answer B. (In case of NBCD only bits A and D are set!)

A.8.5 The type of BCD code of course has nothing to do with the number of output lines. Using a BCD code means 4 lines for each digit. And as $2^6=64$, two decimal digits are required, which means 8 lines (Answer C).

A.8.6 The type of logic does not define the value of the binary number. In negative logic if only means that a "0" has a higher voltage than a "1", but the "1" **remains** "1", and the **"0" remains** "0", so 0110 = decimal 6. (Answer A.)

A.8.7 The decimal value in the BCD down counter is 100, so after 100 clock pulse the counter is at 000, whereas at the same moment its binary counter has the position 1100100 (LSB). Thus the conversion needed 100 clock pulses (Answer C).

A.8.8 As the 1242 BCD-code is a true weighted code this can be done with a DAC whose weighting resistors have the factor 1, 2, 4 and 2 (Answer A).

A.8.9 In both cases the words are already serial, the only difference is the transport of the bits. When the bits are transported in parallel, then P lines are used, which means P times faster, (Answer B).

Printed in The Netherlands — 7201.20.3010.11