Digital Circuits And Logic Design

DCAP108

Edited by:
Dr. Avinash Bhagat

LOVELY PROFESSIONAL UNIVERSITY

DIGITAL CIRCUITS AND LOGIC DESIGN

Edited By
Dr. Avinash Bhagat

Printed by
EXCEL BOOKS PRIVATE LIMITED
A-45, Naraina, Phase-I,
New Delhi-110028
for
Lovely Professional University
Phagwara

SYLLABUS

Digital Circuits and Logic Design

Objectives: To enable the student to understand digital circuits and design of logic gates. Student will learn Boolean algebra, various types of logic gates, combinational circuits, sequential circuits. Student will also learn the storage mechanism of various memories and conversion system.

S. No.	Description
1.	**Basic Concepts:** Number systems - Binary, Octal, Decimal, Hexadecimal, conversion from one to another.
2.	**Boolean Algebra:** Complement arithmetic, Boolean theorems of Boolean algebra.
3.	**Minimization of Boolean Algebra:** Sum of products and product of sums, Minterms and Maxterms, Karnaugh map, Tabulation and computer aided minimization procedures.
4.	**Logic Gates:** RTL, DTL, TTL, ECL, ICL, HTL, NMOS & CMOS logic gates, Circuit diagram and analysis characteristics and specifications, tri-state gates.
5.	**Combinational Circuits:** Problem formulation and design of combinational circuits, Adder / Subtractor, Encoder / decoder, Mux /Demux, Code-converters, Comparators, Implementation of combinational logic using standard ICs.
6.	**Memories:** ROM, EPROM, EEPROM, PAL, PLA and their use in combinational circuit design.
7.	**Sequential Circuits:** Flipflops - SR, JK, T, D, Master/Slave FF, Triggering of FF, Analysis of clocked sequential circuits - their design.
8.	State minimization, state assignment, Circuit implementation, Registers-Shift registers, Ripple counters, Synchronous counters, Timing signal, RAM, Memory decoding, Semiconductor memories.
9.	**A/D and D/A Converters:** Principle of analog to digital conversion, Weighted resistor and ladder networks, single slope, dual slope, successive approximation and flash converters.

CONTENT

Unit 1: Number Systems

Notes

CONTENTS

Objectives

After studying this unit, you will be able to:

- Discuss the decimal number system
- Explain about binary number system
- Discuss the hexadecimal number system
- Discuss the octal number system
- Understand the method of conversion from one number system to another

Introduction

We are familiar with the decimal number system in which digits are 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9. The computer uses binary digits for its operation. In the binary system there are only two digits 0 and 1. The programmer feeds instruction and data in alphabets and decimal digits. But for the operation of the computer these are converted to binary bits. This chapter deals with the conversion of binary numbers to decimal numbers and vice versa. It also deals with hexadecimal and octal system. Computer circuitry is usually designed to process hexadecimal or octal number.

Number Systems are two types:

1. Non-Positional Number System

2. Positional Number System

Notes

Non-Positional Number Systems

In early days, human beings counted on fingers. When ten fingers were not adequate, stones, pebbles or sticks were used to indicate values. This method of counting uses an additive approach or the non-positional number system. In this system, we have symbols such as I for 1, II for 2, III for 3, IIII for 4, IIIII for 5, etc. Each symbol represents the same value regardless of its position in the number and the symbols are simply added to find out the value of a particular number. Since it is very difficult to perform arithmetic with such a number system, positional systems were developed.

Positional Number Systems

In a positional number system, there are only a few symbols called digits, and these symbols represent different values depending on the position they occupy in the number. The value of each digit in such a number is determined by three considerations:

1. The digit itself,

2. The position of the digit in the number, and

3. The base of the number system.

Example:

Decimal, Binary, Octal, Hexadecimal number systems.

1.1 Decimal Number System

The decimal number system is composed of 10 numerals or symbols. These 10 symbols are 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9; using these symbols as digitals of a number, we can express any quantity. The decimal system, also called the base-10 system because it has 10 digits, has evolved naturally as a result of the fact that man has 10 fingers.

The decimal number system is a positional-value system in which the value of a digit depends on its position. For example, consider the decimal number 453. We know that the digit 4 actually represents 4 hundreds, the 5 represents 5 tens and the 3 represents 3 units. In essence, the 4 carries the most weight of the three digital; it is referred to as the most significant digit (MSD). The 3 carries the least weight and is called the least significant digit (LSD).

Consider another example, 27.35. This number is actually equal to 2 tens plus 7 units plus 3 tenths plus 5 hundredths, or 2 * 10 + 7 * 1 + 3 * 0.1 + 5 * 0.01. The decimal point is used to separate the integer and fractional parts of the number.

More rigorously, the various positions relative to the decimal point carry weights that can be express as powers of 10. This is illustrated in Figure 1.1, where the number 2745.214 is represented. The decimal point separates the positive powers of 10 from the negative powers. The number 2745.214 is thus equal to $(2*10^{+3}) + (7*10^{+2}) + (4*10^{+1}) + (5*10^{0}) + (2*10^{-1}) + (1*10^{-2}) + (4 * 10^{-3})$. In general, any number W simply the sum of the products of each digit value and its positional value.

***Decimal Counting*:** When counting, in the decimal system, we start with 0 in the unit's position and take each symbol (digit) in progression until we reach 9. Then we add a 1 to the next higher position and start over with zero in the first position (see Figure 1.1). This process continues until

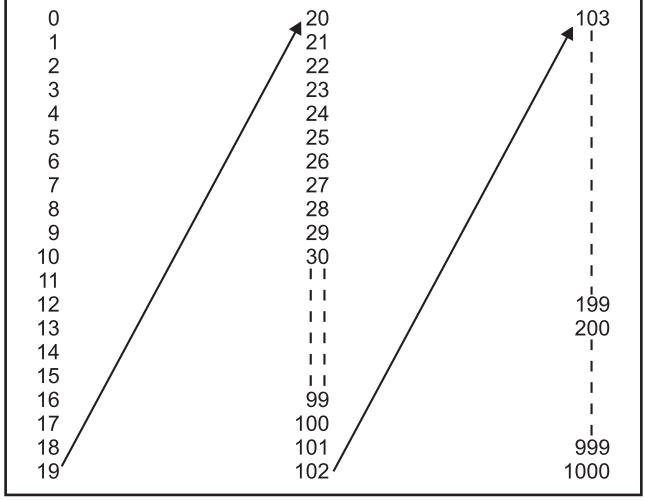
Notes

the count of 99 is reached. Then we add first to the third position and start over with zeros in the first two positions. The same pattern is followed continuously as high as we wish to count.

It is important to note that in decimal counting the units position (LSD) changes upward with each step in the count, the tens position changes upward every 10 steps in the count, the hundreds position changes upward every 100 steps in the count, and so on.

Another characteristic of the decimal system is that using only two decimal places we can count through 10^2 = 100 different numbers (0 to 99). With three places we can count through 1000 numbers (0 to 999); and so on.

Figure 1.1: Decimal Counting



In general, with N places or digits we can count through 10^N different numbers, starting with and including zero. The largest number will always be 10^{N-1}.

Did u know?

Blaise Pascal (French) invented the first adding machine in 1642. Twenty years later, an Englishman, Sir Samuel Moreland, developed a more compact device that could multiply, add, and subtract.

1.2 Binary Number System

Unfortunately, the decimal number system does not lend itself to convenient implementation in digital systems. For example, it is very difficult to design electronic equipment so that it can work with 10 different voltage levels (each one representing one decimal character, 0 through 9). On the other hand, it is very easy to design simple, accurate electronic circuits that operate with only two voltage levels. For this reason, almost every digital system uses the binary number system (base 2) as the basic number system of its operations, although other systems are often used in conjunction with binary.

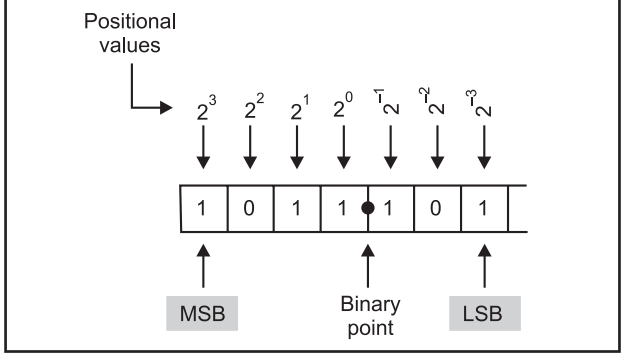
In the binary system there are only two symbols or possible digit values, 0 and 1. Even so, this base-2 system can be used to represent any quantity that can be represented in decimal or other number systems. In general though, it will take a greater number of binary digits to express a given quantity.

All the statements made earlier concerning the decimal system are equally applicable to the binary system. The binary system is also a positional-value system, wherein each binary digit has its

Notes

own value or weight expressed as a power of 2. This is illustrated in Figure 1.2. Here, places to the left of the binary point (Counterpart of the decimal point) are positive powers of 2 and places to the right are negative powers of 2.

Figure 1.2: Binary Position Values as Powers of 2



The number 1011.101 is shown represented in the Figure 1.2. To find its equivalent in the decimal system we simply take the sum of the products of each digit value (0 or 1) and its positional value:

$$1011.101_2 = (1 * 2^3) + (0 * 2^2) + (1 * 2^1) + (1 * 2^0)$$
$$+ (0 * 2^{-1}) + (0 * 2^{-2}) + (1 * 2^{-3})$$
$$= 8 + 0 + 2 + 1 + 0.5 + 0 + 0.125 = 11.625_{10}$$

Notice in the preceding operation that subscripts (2 and 10) were used to indicate the base in which the particular number is expressed. This convention is used to avoid confusion whenever more than one number system is being employed.

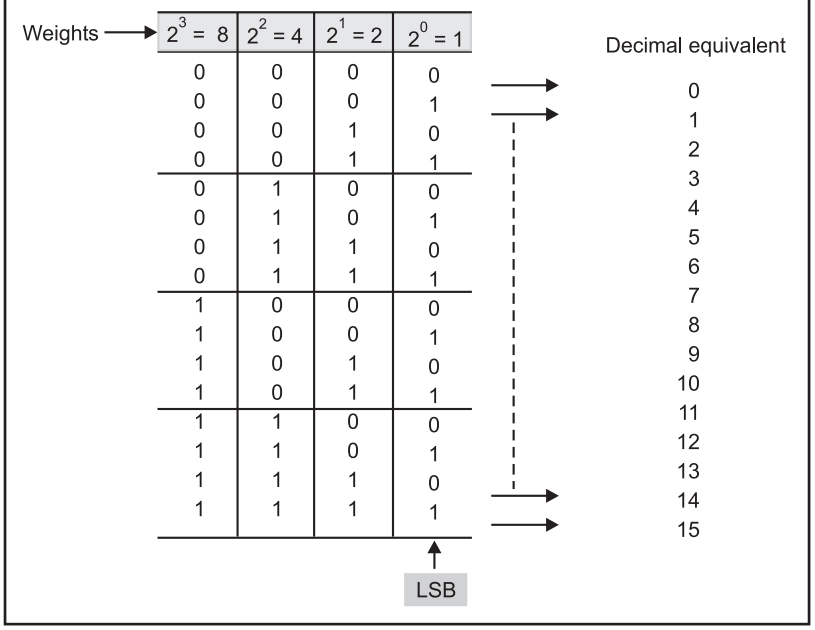
In the binary system, the term binary digit is often abbreviated to the term bit, which we will use henceforth. Thus, in the number expressed in Figure 1.2 there are 4 bits to the left of the binary point, representing the integer part of the number, and 3 bits to the right of the binary point, representing the fractional part. The most significant bit (MSB) is the leftmost bit (largest weight). The least significant bit. (LSB) is the rightmost bit (smallest weight). These are indicated in Figure 1.2.

***Binary Counting*:** When we deal with binary numbers, we will usually be restricted to a specific number of bits. This restriction is based on the circuitry that is used to represent these binary numbers. Let us use 4-bit binary numbers to illustrate the method for counting in binary.

The sequence (shown in Figure 1.3) begins with all bits at 0; this is called the zero count. For each successive count, the units (2^0) position toggles; that is, it changes from one binary value to the other. Each time the units bit changes from a 1 to a 0, the twos (2^1) position will toggle (change states). Each time the two's position changes from 1 to 0, the fours (2^2) position will toggle (change states). Likewise each time the fours position goes from 1 to 0, the eights (2^3) position toggles. This same process would be continued for the higher-order bit positions if the binary number had more than 4-bits.

Notes

Figure 1.3: Binary Counting Sequence



The binary counting sequence has an important characteristic, as shown in Figure 1.3 the units bit (LSB) changes either from 0 to 1 or 1 to 0 with each count. The second bit (two's position) stays at 0 for two counts, then at 1 for two counts, then at 0 for two counts, and so on. The third bit (fours position) stays at 0 for four counts, then at 1 for four counts, and so on. The fourth bit (eights position) stays at 0 for eight counts, then at 1 for eight counts. If we wanted to count further we would add more places, and this pattern would continue with 0s and 1s alternating in groups of 2^{N-1}. For example, using a fifth binary place, the fifth bit would alternate sixteen 0s, then sixteen 1s, and so on.

As we saw for the decimal system, it is also true for the binary system that by using ***N***-bits or places we can go through 2^N counts. For example, with 2-bits we can go through $2^2 = 4$ counts (00_2 through 11_2); with 4-bits we can go through $2^4 = 16$ counts (0000_2 through 1111_2); and so on. The last count will always be all is and is equal to 2^N–1 in the decimal system. For example, using 4-bits, the last count is 11112 = *V* – 1= 1510.

Self Assessment

Choose the correct answer:

1. The decimal number system is composed of ..... numerals or symbols.
   (*a*) 8 (*b*) 10
   (*c*) 11 (*d*) 15
2. The base of the decimal number is ..... .
   (*a*) 0 (*b*) 10
   (*c*) 11 (*d*) 8
3. Binary system there are only two symbols or possible digit values, and ..... .
   (*a*) 1 and 2 (*b*) 8 and 10
   (*c*) 0 and 1 (*d*) 8 and 2

Notes

4. A digital system is a combination of devices designed to manipulate physical quantities or information that are represented in digital form; that is, they can take on only discrete values.

   (*a*) True (*b*) False

1.3 Hexadecimal Number System

The hexadecimal number system is used as an intermediary system in computers, such as are presentation of memory addresses or a representation of colours. The hexadecimal number system is also known as the base-16 number system, because each position in the number represents an incremental number with a base of 16 (see Table 1.1). For example, the first position (the furthest right) is represented as 16, the second position (one from furthest right) is represented as 16, and so forth. To determine what the actual number is in "decimal" representation, take the number that appears in the position, and multiply it by 16x, where x is the power representation. For example, if a number appears in the furthest right position, take the number in the furthest right position and multiply it by 16. If there are multiple positions in the number (ex: 17AF), add all the results together.

Since the number system is represented in "sixteens", there are only 10 numbers and 5 letters that can be a value in each position of the base-16 number. Below are the numbers that each position can hold:

Table 1.1: Comparing Number Hexadecimal to Decimal Values

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

Caution

Be careful to use ten different symbols like: 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9 for hexadecimal number.

1.4 Octal Number System

Just as the decimal system with its ten digits is a base-ten system, the octal number system with its 8 digits, '0', '1', '2', '3', '4', '5', '6' and '7', is a base-eight system. Table 1.2 shows the weighting for

Notes

the octal number system up to 3 decimal places before and 2 decimal places after the octal point (.).

Table 1.2: Octal Weights

Weights	8^2	8^1	8^0	.	8^{-1}	8^{-2}

Just like the other counting conventions discussed previously, the LSB begins with zero (0) and is incremented until the maximum digit value is reached. The adjacent bit positions are then filled appropriately as the iterative counting process continues. Thus the counting convention for octal is 0, 1, 2, 3, 4, 5, 7, 10, 11, 12, 13, 14, 15, 16, 17, 20 .....

The octal, or base 8, number system is a common system used with computers. Because of its relationship with the binary system, it is useful in programming some types of computers.

Look closely at the comparison of binary and octal number systems in Table 1.3. You can see that one octal digit is the equivalent value of three binary digits. The following examples of the conversion of octal 2258 to binary and back again further illustrate this comparison:

Octal to Binary

2	2	5_8
010	010	101_2

Binary to Octal

010	010	101_2
2	2	5_8

Table 1.3: Binary and Octal Comparison

	BINARY	OCTAL	
2^0	0	0	8^0
	1	1	
2^1	10	2	
	11	3	
2^2	100	4	
	101	5	
	110	6	
	111	7	
2^3	1000	10	8^1
	1001	11	
	1010	12	
	1011	13	
	1100	14	
	1101	15	
	1110	16	
	1111	17	
2^4	10000	20	
	10001	21	
	10010	22	
	10011	23	
	10100	24	
	10101	25	
	10110	26	
	10111	27	
	11000	30	

Notes

Unit and Number

The terms that you learned in the decimal and binary sections are also used with the octal system.

The unit remains a single object, and the number is still a symbol used to represent one or more units.

***Base* (*Radix*)**

As with the other systems, the radix, or base, is the number of symbols used in the system. The octal system uses eight symbols—0 through 7. The base, or radix, is indicated by the subscript 8.

Positional Notation

The octal number system is a positional notation number system. Just as the decimal system uses powers of 10 and the binary system uses powers of 2, the octal system uses power of 8 to determine the value of a number's position. The following bar graph shows the positions and the power of the base:

Remember that the power, or exponent, indicates the number of times the base is multiplied by itself. The value of this multiplication is expressed in base 10 as shown below:

$$8^3 = 8 \times 8 \times 8, \text{ or } 512_{10}$$
$$8^2 = 8 \times 8, \text{ or } 64_{10}$$
$$8^1 = 8_{10}$$
$$8^0 = 1_{10}$$
$$8^{-1} = \frac{1}{8}, \text{ or } .125_{10}$$
$$8^{-2} = \frac{1}{8 \times 8}, \text{ or } \frac{1}{64}, \text{ or } .015625_{10}$$
$$8^{-3} = \frac{1}{8 \times 8 \times 8}, \text{ or } \frac{1}{512}, \text{ or } .0019531_{10}$$

All numbers to the left of the radix point are whole numbers, and those to the right are fractional numbers.

Did u know?

About 1672, Gottfried Wilhelm von Leibniz (German) perfected a machine that could perform all the basic operations (add, subtract, multiply, divide), as well as extract the square root. Modern electronic digital computers still use von Leibniz's principles.

1.5 Conversion from One Number System to Another

Numbers expressed in decimal number system are much more meaningful to us than are numbers expressed in any other number system. This is because we have been using decimal numbers in our day-to-day life, right from childhood; however, we can represent any number in one number system in any other number system. Because the input and final output values are to be in decimal, computer professionals are often required to convert number in other systems to decimal and vice versa. Many methods can be used to convert numbers from one base to another. A method of converting from another base to decimal, and a method of converting from decimal to another base are described here:

1.5.1 Converting from Another Base to Decimal

The following steps are used to convert a number in any other base to a base 10 (decimal) number:

Step 1: Determine the column (positional) value of each digit (this depends on the position of the digit and the base of the number system).

Notes

Step 2: Multiply the obtained column values (in Step 1) by the digits in the corresponding columns.

Step 3: Sum up the products calculated in Step 2. The total is the equivalent value in decimal.

Example:

11001_2 = ?10

Solution:

Step 1: Determine column values

Column Number	Column (From Right) $2^0 = 1$
1	$2^1 = 2$
2	$2^2 = 4$
3	$2^3 = 8$
4	$2^4=16$

Step 2: Multiply the column values by the corresponding column digits

16	8	4	2	1
*1	*1	*0	*0	*1
16	8	0	0	1

Step 3: Sum up the products

16+8+0+0+1=25

Hence, $11001_2=25_{10}$

Example:

4706_8=? $_{10}$

Solution:

Step 1: Determine column values

Column Number (From Right)	Column
1	$8^0 = 1$
2	$8^1 = 8$
3	$8^2 = 64$
4	$8^3 = 512$

Step 2: Multiply the column values by the corressponding column digits

512	64	8	1
* 4	* 7	* 0	* 6
2048	448	0	6

Step 3: Sum up the products

2048 + 448 + 0 + 6 = 2502

Hence, $4706_8 = 2502_{10}$

Notes

Example:

$1AC_{16} = ?_{10}$

Solution:

$$1AC_{16} = 1*16^2 + A*161 + C*16^0$$
$$= 1*256 + 10*16 + 12*1$$
$$= 256 + 160 + 12$$
$$= 428_{10}$$

Example:

$4052_7 = ?_{10}$

Solution:

$$4052_7 = 4*7^3 + 0*7^2 + 5*7^1 + 2*7^0$$
$$= 4*343 + 0*49 + 5*7 + 2*1$$
$$= 1372 + 0 + 35 + 2$$
$$1409_{10}$$

Example:

$4052_6 = ?_{10}$

Solution:

$$4052_6 = 4*6^3 + 0*6^2 + 5*6^1 + 2*6^0$$
$$= 4*216 + 0*36 + 5*6 + 2*1$$
$$= 864 + 0 + 30 + 2$$
$$= 896_{10}$$

Example:

Solution:

$$11001_4 = 1*4^4 + 1*4^3 + 0*4^2 + 1*4^1 + l*4^0$$
$$= 1*256 + 1*64 + 0*16 + 0*4 + l*l$$
$$= 256 + 64 + 0 + 0 + 1$$
$$= 321_{10}$$

Example:

$1AC_{13} = ?_{10}$

Solution:

$$lAC_{13} = 1*13^2 + A*13^1 + C*13^0$$
$$= 1*169 + 10*13 + 12*1$$
$$= 311_{10}$$

Notes

1.5.2 Converting from Decimal to Another Base (Division-Remainder Technique)

The following steps are used to convert a base 10 (decimal) number to a number in another base

Step 1: Divide the decimal number by the value of the new base.

Step 2: Record the remainder from Step 1 as the rightmost digit (least significant digit) of the new base number.

Step 3: Divide the quotient of the previous division by the new base.

Step 4: Record the remainder from Step 3 as the next digit (to the left) of the new base number. Repeat Steps 3 and 4, recording remainders from right to left, until the quotient becomes zero in Step 3.

Note that the last remainder, thus obtained, will be the most significant digit of the new base number.

Example:

25_{10}

Solution:

Steps 1: 25/2 = 12 and remainder 1

Steps 2: 12/2 = 6 and remainder 0

Steps 3: 6/2 = 3 and remainder 0

Steps 4: 3/2 = 1 and remainder 1

Steps 5: 1/2 = 0 and remainder 1

The remainders are now arranged in the reverse order, making the first remainder the least significant digit (LSD) and the last remainder the most significant digit (MSD).

Hence, $25_{10} = 11001_2$

1.5.3 Converting from a Base Other Than 10 to Another Base Other Than 10

The following steps are used to convert a number in a base other than 10, to number base other than 10:

Step 1: Convert the original number to a base (decimal) number.

Step 2: Convert the decimal number obtained in step 1 to the new base number.

Example:

$545_6 = ?_4$

Solution:

Step 1: Convert from base 6 to base 10

$$545 = 5*6^2 + 4*6^1 + 5*6^0$$

$$= 5*36 + 4*6 + 5*1$$

$$= 180 + 24 + 5$$

$$= 209_{10}$$

Notes

Step 2: Convert 209_{10} to base 4

		Remainder
4	209	1
	52	0
	13	1
	3	3
	0	

$209_{10} = 3101_4$

Therefore, $545_6 = 209_{10} = 3101_4$

Hence, $546_6 = 3101_4$

Above example illustrates the method of converting a binary number to an octal number.

1.5.4 Short-cut Method for Binary to Octal Conversion

The following steps are used in this method:

Step 1: Divide the binary digits into groups of three (starting from the right)

Step 2: Convert each group of three binary digit to one octal digit. Since there are only 8 digits (0 to 7) in octal number system, 3 bits ($2^3 = 8$) are sufficient to represent any octal number in binary to decimal conversion method in this step.

Example:

$101110_2 = ?_8$

Solution:

Step 1: Divide the binary digits into groups of 3, starting the right (LSD).

101 110

Step 2: Convert each group into one digit of octal (use binary-to-decimal conversion method).

$101_2 = 1*2^2 + 0*2^1 + 1*2^0$ $\qquad$ $110_2 = 1*2^2 + 1*2^1 + 0*2^0$

$= 4 + 0 + 1$ $\qquad$ $4 + 2 + 0$

$= 5_8$ $\qquad$ 6_8

Hence, $101110_2 = 56_8$

Caution Always check that analogs method is used to convert fractions while conversion of number representations.

1.5.5 Short-cut Method for Octal to Binary Conversion

The following steps are used in this method:

Step 1: Convert each octal digit to a 3 digit binary number (the octal digits may be treated as decimal numbers for this conversion).

Step 2: Combine all the resulting binary groups (of 3 digits each) into a single binary number.

Notes

Example:

$562_8 = ?_2$

Solution:

Step 1: Convert each octal digit to 3

Binary digits

$5_8 = 101_2$

$6_8 = 110_2$

$2_8\text{-}010_2$

Step 2: Combine the binary groups.

$562_8 = \frac{101}{5}\ \frac{110}{6}\ \frac{010}{2}$

Hence, $562_8 = 101110010_2$

1.5.6 Short-cut Method for Binary to Hexadecimal Conversion

The following steps are used in this method:

Step 1: Divide the binary digits into groups of four (starting from the right).

Step 2: Convert each group of four binary digits to one hexadecimal digit. Remember that hexadecimal digits 0 to 9 are equal to decimal digits 0 to 9 and hexadecimal digits A to F are equal to decimal values 10 to 15. Hence, for this step, we use binary to decimal conversion procedure and represent decimal values 10 to 15 as hexadecimal A to F.

Example:

$11010011_2 = ?_{16}$

Solution:

Step 1: Divide the binary digit into groups of 4, starting from the right (LSD)

1101 0011

Step 2: Convert each group of 4 binary digits to 1 hexadecimal digit.

$1101_2 = 1*2^3 + 1*2^2 + 0*2^1 + 1*2^0$

$= 8 + 4 + 0 + 1$

$= 13_{10}$

$= D_{16}$

$0011_2 = 0*2^3 + 0*2^2 + 1*2^1 + 1*2^0$

$= 0 + 0 + 2 + 1$

$= 3_{16}$

Hence, $11010011_2 = D3_{16}$

Task Convert the number $1001011010011_2 = ?_{16}$

1.5.7 Short-cut Method for Hexadecimal to Binary Conversion

The following steps are used in this method:

Step 1: Convert decimal equivalent of each hexadecimal digit to 4 binary digits.

Step 2: Combine all resulting binary groups (4 digits each) into a single binary number.

Notes

Example:

$2AB_{16}=?_{2}$

Solution:

Step 1: Convert decimal equivalent each hexadecimal digit to 4 binary digits.

$$2_{16} = 2_{10} = 0010_2$$

$$A_{16} = 10_{10} = 1010_2$$

$$B_{16} = 11_{10} = 1011_2$$

Step 2: Combine the binary groups.

$$2AB_{16} = \frac{0010}{2}\ \frac{0010}{A}\ \frac{1011}{B}$$

Hence, $2AB_{16} = 001010101011_2$

Table 1.4 summarizes the relationship among decimal, hexadecimal, binary, and octal number systems. Note that the maximum value for a single digit of octal (7) is equal to the maximum value of three digits of binary. The value range of one digit of octal duplicates the value range of three digits of binary. If we substitute octal digits for binary digits, the substitution is on a one-to-three basis. Hence, computers that print octal numbers instead binary, while taking memory dump, save one-third of printing space and time.

Similarly, note that the maximum value of one digit in hexadecimal is equal to the maximum value of four digits in binary. Hence, the value range of one digit of hexadecimal is equivalent to the value range of four digits of binary. Therefore, hexadecimal shortcut notation is a one-to-four reduction in space and time required for memory dump.

Table 1.4: Relationship among Decimal, Hexadecimal, Binary, and Octal Number Systems

Decimal	Hexadecimal	Binary	Octal
0	0	0	0
1	1	1	1
2	2	10	2
3	3	11	3
4	4	100	4
5	5	111	5
6	6	110	6
7	7	111	7
8	8	1000	10
9	9	1001	11
10	A	1010	12
11	B	1011	13
12	C	1100	14
13	D	1101	15
14	E	1110	16
15	F	1111	17
16	10	10000	20

Notes

Task Convert the value $2AFCB_{16} = ?_{2}$

Case Study

The Indian Numeral System

Although the Chinese were also using a decimal based counting system, the Chinese lacked a formal notational system that had the abstraction and elegance of the Indian notational system, and it was the Indian notational system that reached the Western world through the Arabs and has now been accepted as universal. Several factors contributed to this development whose significance is perhaps best stated by French mathematician Laplace: "The ingenious method of expressing every possible number using a set of ten symbols (each symbol having a place value and an absolute value) emerged in India. The idea seems so simple nowadays that its significance and profound importance is no longer appreciated. Its is simplicity lies in the way it facilitated calculation and placed arithmetic foremost amongst useful inventions."

Brilliant as it was, this invention was no accident. In the Western world, the cumbersome Roman numeral system posed as a major obstacle, and in China the pictorial script posed as a hindrance. But in India, almost everything was in place to favour such a development. There was already a long and established history in the use of decimal numbers, and philosophical and cosmological constructs encouraged a creative and expansive approach to number theory. **Panini's** studies in linguistic theory and formal language and the powerful role of symbolism and representational abstraction in art and architecture may have also provided an impetus, as might have the rationalist doctrines and the exacting epistemology of the **Nyaya Sutras**, and the innovative abstractions of the **Syadavada** and Buddhist schools of learning.

Influence of Trade and Commerce, Importance of Astronomy

The growth of trade and commerce, particularly lending and borrowing demanded an understanding of both simple and compound interest which probably stimulated the interest in arithmetic and geometric series. **Brahmagupta's** description of negative numbers as debts and positive numbers as fortunes points to a link between trade and mathematical study. Knowledge of astronomy—particularly knowledge of the tides and the stars—was of great import to trading communities who crossed oceans or deserts at night. This is borne out by numerous references in the **Jataka** tales and several other folk tales. The young person who wished to embark on a commercial venture was inevitably required to first gain some grounding in astronomy. This led to a proliferation of teachers of astronomy, who in turn received training at universities such as at Kusumpura (Bihar) or Ujjain (Central India) or at smaller local colleges or *Gurukuls*. This also led to the exchange of texts on astronomy and mathematics amongst scholars and the transmission of knowledge from one part of India to another. Virtually every Indian state produced great mathematicians who wrote commentaries on the works of other mathematicians (who may have lived and worked in a different part of India many centuries earlier). Sanskrit served as the common medium of scientific communication.

The science of astronomy was also spurred by the need to have accurate calendars and a better understanding of climate and rainfall patterns for timely sowing and choice of crops. At the same time, religion and astrology also played a role in creating an interest in astronomy

Contd...

Notes

and a negative fallout of this irrational influence was the rejection of scientific theories that were far ahead of their time. One of the greatest scientists of the Gupta period—**Aryabhatta** (born in 476 AD, Kusumpura, Bihar)—provided a systematic treatment of the position of the planets in space. He correctly posited the axial rotation of the earth, and inferred correctly that the orbits of the planets were ellipses. He also correctly deduced that the moon and the planets shined by reflected sunlight and provided a valid explanation for the solar and lunar eclipses rejecting the superstitions and mythical belief systems surrounding the phenomenon. Although **Bhaskar** *I* (born Saurashtra, 6th C, and follower of the Asmaka school of science, Nizamabad, Andhra) recognized his genius and the tremendous value of his scientific contributions, some later astronomers continued to believe in a static earth and rejected his rational explanations of the eclipses. But in spite of such setbacks, Aryabhatta had a profound influence on the astronomers and mathematicians who followed him, particularly on those from the Asmaka school.

Mathematics played a vital role in **Aryabhatta's** revolutionary understanding of the solar system. His calculations on pi, the circumferance of the earth (62832 miles) and the length of the solar year (within about 13 minutes of the modern calculation) were remarkably close approximations. In making such calculations, **Aryabhatta** had to solve several mathematical problems that had not been addressed before including problems in algebra *(*beej-ganit*)* and trigonometry (trikonmiti).

Bhaskar *I* continued where **Aryabhatta** left off, and discussed in further detail topics such as the longitudes of the planets; conjunctions of the planets with each other and with bright stars; risings and settings of the planets; and the lunar crescent. Again, these studies required still more advanced mathematics and *Bhaskar I* expanded on the trigonometric equations provided by *Aryabhatta*, and like *Aryabhatta* correctly assessed pi to be an irrational number. Amongst his most important contributions was his formula for calculating the sine function which was 99% accurate. He also did pioneering work on indeterminate equations and considered for the first time quadrilaterals with all the four sides unequal and none of the opposite sides parallel.

Another important astronomer/mathematician was *Varahamihira* (6th C, Ujjain) who compiled previously written texts on astronomy and made important additions to *Aryabhatta's* trigonometric formulas. His works on permutations and combinations complemented what had been previously achieved by Jain mathematicians and provided a method of calculation of *n*Cr that closely resembles the much more recent *Pascal's Triangle*. In the 7th century, *Brahmagupta* did important work in enumerating the basic principles of algebra. In addition to listing the algebraic properties of zero, he also listed the algebraic properties of negative numbers. His work on solutions to quadratic indeterminate equations anticipated the work of Euler and Lagrange.

Emergence of Calculus

In the course of developing a precise mapping of the lunar eclipse, *Aryabhatta* was obliged to introduce the concept of infinitesimals—i.e. *tatkalika gati* to designate the infinitesimal, or near-instantaneous motion of the moon—and express it in the form of a basic differential equation. *Aryabhatta's* equations were elaborated on by *Manjula* (10th C) and *Bhaskaracharya* (12th C) who derived the differential of the sine function. Later mathematicians used their intuitive understanding of integration in deriving the areas of curved surfaces and the volumes enclosed by them.

Applied Mathematics, Solutions to Practical Problems

Developments also took place in applied mathematics such as in creation of trigonometric tables and measurement units. *Yativrsabha's* work *Tiloyapannatti* (6th C) gives various units for measuring distances and time and also describes the system of infinite time measures.

Contd...

Notes

In the 9th C, *Mahaviracharya* (Mysore) wrote *Ganit Saar Sangraha* where he described the currently used method of calculating the Least Common Multiple (LCM) of given numbers. He also derived formulae to calculate the area of an ellipse and a quadrilateral inscribed within a circle (something that had also been looked at by *Brahmagupta*). The solution of indeterminate equations also drew considerable interest in the 9th century, and several mathematicians contributed approximations and solutions to different types of indeterminate equations.

In the late 9th C, *Sridhara* (probably Bengal) provided mathematical formulae for a variety of practical problems involving ratios, barter, simple interest, mixtures, purchase and sale, rates of travel, wages, and filling of cisterns. Some of these examples involved fairly complicated solutions and his *Patiganita* is considered an advanced mathematical work. Sections of the book were also devoted to arithmetic and geometric progressions, including progressions with fractional numbers or terms, and formulas for the sum of certain finite series are provided. Mathematical investigation continued into the 10th C. *Vijayanandi* (of Benares, whose *Karanatilaka* was translated by *Al-Beruni* into Arabic) and *Sripati* of Maharashtra are amongst the prominent mathematicians of the century.

The leading light of 12th C Indian mathematics was *Bhaskaracharya* who came from a long-line of mathematicians and was head of the astronomical observatory at Ujjain. He left several important mathematical texts including the *Lilavati* and *Bijaganita* and the *Siddhanta Shiromani*, an astronomical text. He was the first to recognize that certain types of quadratic equations could have two solutions. His **Chakrawaat** method of solving indeterminate solutions preceded European solutions by several centuries, and in his **Siddhanta Shiromani** he postulated that the earth had a gravitational force, and broached the fields of infinitesimal calculation and integration. In the second part of this treatise, there are several parts relating to the study of the sphere and its properties and applications to geography, planetary mean motion, eccentric epicyclical model of the planets, first visibilities of the planets, the seasons, the lunar crescent etc. He also discussed astronomical instruments and spherical trigonometry. Of particular interest are his trigonometric equations: $\sin(a + b) = \sin a \cos b + \cos a \sin b$; $\sin(a - b) = \sin a \cos b - \cos a \sin b$;

The Spread of Indian Mathematics

The study of mathematics appears to slow down after the onslaught of the Islamic invasions and the conversion of colleges and universities to *madrasahs*. But this was also the time when Indian mathematical texts were increasingly being translated into Arabic and Persian. Although Arab scholars relied on a variety of sources including Babylonian, Syriac, Greek and some Chinese texts, Indian mathematical texts played a particularly important role. Scholars such as Ibn Tariq and Al-Fazari (8th C, Baghdad), Al-Kindi (9th C, Basra), Al-Khwarizmi (9th C. Khiva), Al-Qayarawani (9th C, Maghreb, author of Kitab fi al-hisab al-hindi), Al-Uqlidisi Ibn-Sina (Avicenna), Ibn al-Samh (Granada, 11th C, Spain), Al-Nasawi (Khurasan, 11th C, Persia), Al-Beruni (11th C, born Khiva, died Afghanistan), Al-Razi (Teheran), and Ibn-Al-Saffar (11th C, Cordoba) were amongst the many who based their own scientific texts on translations of Indian treatises. Records of the Indian origin of many proofs, concepts and formulations were obscured in the later centuries, but the enormous contributions of Indian mathematics was generously acknowledged by several important Arabic and Persian scholars, especially in Spain. Abbasid scholar *Al-Gaheth* wrote: "*India is the source of knowledge, thought and insight*". *Al-Maoudi* (AD 956) who travelled in Western India also wrote about the greatness of Indian science. *Said Al-Andalusi*, an 11th C Spanish scholar and court historian, was amongst the most enthusiastic in his praise of Indian civilization, and specially remarked on Indian achievements in the sciences and in mathematics. Of course, eventually, Indian algebra and trigonometry reached Europe through a cycle of translations, travelling from the Arab world to Spain and Sicily, and eventually penetrating all of Europe. At the same time, Arabic and Persian translations of Greek and Egyptian scientific texts become more readily available in India.

Contd...

Notes

Questions:

1. What are the differences between the Indian mathematics and Chinese mathematics?
2. Why the invention of calculus was needed? How it affected mathematics?

Self Assessment

Choose the correct answer:

5. The hexadecimal number system is also known as the base-16 number system, because each position in the number represents an incremental number with a base of 16.

   (*a*) True (*b*) False

6. In Hexadecimal number system A stands for:

   (*a*) 12 (*b*) 10

   (*c*) 11 (*d*) 13

7. In Hexadecimal number system F stands for:

   (*a*) 14 (*b*) 10

   (*c*) 15 (*d*) 13

8. The octal, or base ....., number system is a common system used with computers.

   (*a*) 0 (*b*) 10

   (*c*) 11 (*d*) 8

1.6 Summary

- A number system is a way of representing a number. Every number system has a base (the number of digits available).
- A number system does not change the value of the number, but only the manner in which it is represented. What we mean to say is that the value of the number remains the same, but the digits we use and how we use them decides the representation of that number.
- The decimal system is a positional-value system in which the value of a digit depends on its position.
- The decimal point separates the positive powers of 10 from the negative powers.
- The binary system is also a positional-value system, wherein each binary digit has its own value or weight expressed as a power of 2.
- The hexadecimal number system is also known as the base-16 number system, because each position in the number represents an incremental number with a base of 16.

1.7 Keywords

***Binary system*:** It is also a positional-value system, wherein each binary digit has its own value or weight expressed as a power of 2.

***Digital systems*:** Digital systems process digital signals which can take only a limited number of values (discrete steps)—usually just two values are used: the positive supply voltage (+Vs) and zero volts (0V).

***Hexadecimal number system*:** It is also known as the base-16 number system, because each position in the number represents an incremental number with a base of 16.

Notes

***Number system*:** It is a basic counting various items. On hearing the word number all of us immediately think of the familiar decimal number system with its 10 digit: 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9.

***Octal*:** The octal, or base 8, number system is a common system used with computers. Because of its relationship with the binary system, it is useful in programming some types of computers.

Lab Exercise

1. Give an example and write the steps of converting from octal number system to decimal value.
2. Convert 336 from base 6 to base 4 value.

1.8 Review Questions

1. Discuss the number system with suitable example.
2. Explain the decimal number system and which decimal is equivalent of 1101011_2.
3. What is the largest number that can be represented using 8 bits?
4. Explain the method of conversion from octal number to binary number system.
5. What is the next binary after 10111_2 in the counting sequence?
6. Explain the method of conversion from hexadecimal to binary number system.
7. What is the hexadecimal representation for the binary number 100100101011?
8. What is the largest decimal value that can be represented using 12 bits?
9. Convert the following binary number system to their equivalent decimal value.
   (*a*) 11001_2 = ... $_{10}$
   (*b*) 1001.1001_2 = ... $_{10}$
   (*c*) 10011011001.10110_2 = ... $_{10}$
10. Convert from one number system to another
    (*a*) 5766_8 = ? $_{10}$
    (*b*) $21AFC_{16}$ = ? $_{10}$
    (*c*) 3552642_7 = ?$_{10}$
    (*d*) 3214052_6 = ? $_{10}$
    (*e*) $1DACC_{13}$ = ? $_{10}$
    (*f*) 32545_6 = ?$_4$

Answers to Self Assessment

1. (*b*) 2. (*b*) 3. (*c*) 4. (*a*)
5. (*a*) 6. (*b*) 7. (*c*) 8. (*d*)

1.9 Further Reading

Books

Digital Systems: *Principles and Applications*, by Ronald J. Tocc.

Online link

http://books.google.co.in/books?

Notes

Unit 2: Logic Gates

CONTENTS

Objectives

After studying this unit, you will be able to:

- Explain the types of logic gates
- Describe the circuit diagram of gates
- Discuss tri-state gates

Introduction

Logic gates are the basic components in digital electronics. They are used to create digital circuits and even complex integrated circuits. For example, complex integrated circuits may bring already a complete circuit ready to be used – microprocessors and microcontrollers are the best example – but inside them they were projected using several logic gates. In this tutorial we will teach you everything you need to know about logic gates, with several examples.

As you may already know, digital electronics accept only two numbers, "0" and "1". Zero means a 0 V voltage, while "1" means 5 V voltages or 3.3 V voltages on newer integrated circuits. You can think "0" and "1" as a light bulb turned off or on or as a switch turned off or on.

Notes

A letter, also known as variable, represents a binary number. So "A" can be "0" or "1". So, if A is connected to a switch, A will be "0" when the switch is turned off and "1" when the switch is turned on. A line drawn right above the variable name means that the variable is inverted. For example, if A = 0, /A will be "1", and if A = 1, /A will be "0". On text processors we need to substitute the line drawn above the letter with a slash because it is hard to draw a line above the letter. So here we will be using this kind of notation due to a limitation on our text processor.

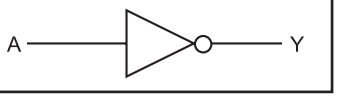
2.1 Types of Logic Gates

There are many types of logic gates, are illustrated with drawings of their relay-and-switch implementations:

2.1.1 The NOT (Inverter) Gate

As the name implies, inverter will invert the number entered. If you enter "0", you will get a "1"on its output, and if you enter a "1", you will get a "0" on its output. The inverter symbol you can see in Figure 2.1. Inverter gate is also known as NOT and its output is Y = /A.

Figure 2.1: Inverter



On the truth table below you can see a summary of how this circuit works.

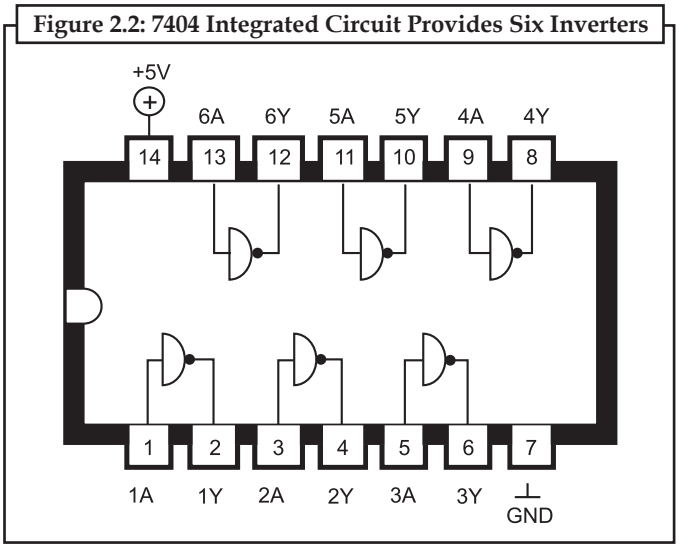
Table 2.1: Truth Table

A (Input)	Y (Output)
0	1
1	0

On logic circuits, an "o" symbol is a short for inverter. You will see that on logic gates like NAND, NOR and XNOR.

The most famous inverter integrated circuit is 7404 and you see its pinout in Figure 2.2. It has six inverters inside. To make this integrated circuit work, you need to connect it to a 5 V power supply.

Figure 2.2: 7404 Integrated Circuit Provides Six Inverters



Notes

2.1.2 The AND Gate

As its name implies, an AND logic gate performs an "AND" logic operation, which is a multiplication. It has at least two inputs. So, if A and B are its inputs, at the output we will find A * B (also represented as A • B). So, AND logic gate can be summarized by the formula Y = A* B (or Y = A * B).

Figure 2.3: AND Logic Gate

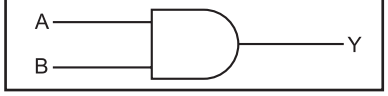


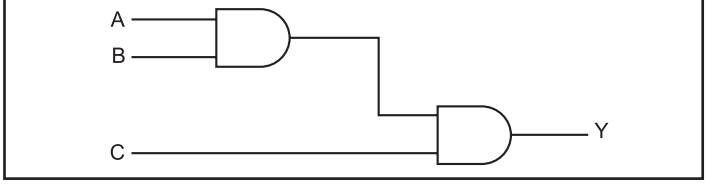
Table 2.2: Truth Table

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

Another way to understand AND logic gate: its output will only be at "1" when all its inputs are also at "1". Otherwise its output will be "0".

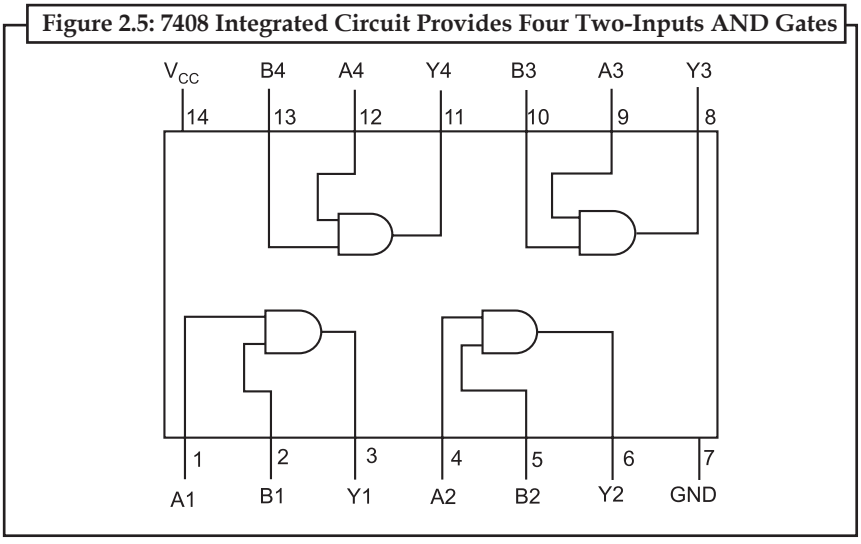
If you are projecting a circuit and need an AND logic gate with more inputs, you can go ahead and simple draw an AND logic gate like the one in Figure 2.3 and put more inputs on it. But if you are working with integrated circuits with AND logic gates with fewer inputs that you need, you can expand the number of inputs by connecting them like shown in Figure 2.4.

Figure 2.4: Expanding AND Inputs Using 2-Input Gates



The most famous AND gate integrated circuit is 7408 and you can its pinout in Figure 2.5. Of course there are several other integrated circuits that provide AND gates with more inputs. For example, 7411 provides three three-input AND gates.

Figure 2.5: 7408 Integrated Circuit Provides Four Two-Inputs AND Gates



Notes

2.1.3 The OR Gate

As its name implies, an OR logic gate performs an "OR" logic operation, which is an addition. It has at least two inputs. So, if A and B are its inputs, at the output we will find A + B. So, OR logic gate can be summarized by the formula Y = A + B. You can see its symbol in Figure 2.6 and its truth table right below it.

Figure 2.6: OR Logic Gate

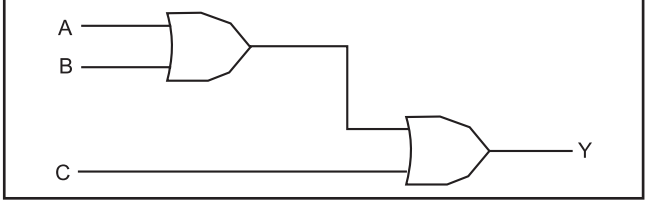
Table 2.3: Truth Table

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

Another way to understand OR logic gate: its output will only be at "0" when all its inputs are also at "0". Otherwise its output will be "1".

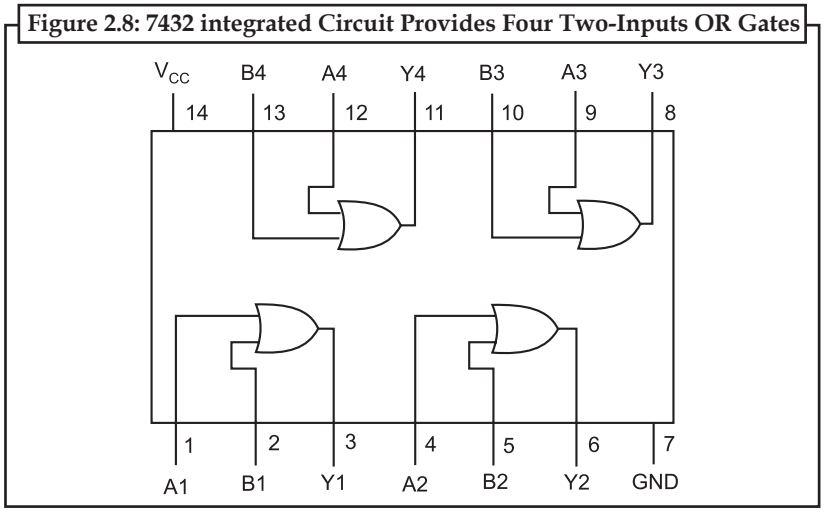
If you need more than two inputs, the same idea applies. If you are projecting a circuit and need an OR logic gate with more inputs, you can go ahead and simple draw an OR logic gate like the one in Figure 2.6 and put more inputs on it. But if you are working with integrated circuits with OR logic gates with fewer inputs that you need, you can expand the number of inputs by connecting them like shown in Figure 2.7.

Figure 2.7: Expanding OR Inputs Using 2-Input Gates



The most famous OR gate integrated circuit is 7432 and you can its pinout in Figure 2.8. Of course there are several other integrated circuits that provide OR gates with more inputs. For example, 7427 provides three three-input OR gates.

Figure 2.8: 7432 integrated Circuit Provides Four Two-Inputs OR Gates



Notes

2.1.4 NAND Gate

The "N" letter on NAND stands for NOT, meaning that NAND logic gate is an AND gate with an inverter attached. So, its output is the opposite from AND. Its symbol is the same of AND but with a "o" on its output, meaning that the output is inverted. You can build yourself a NAND gate by connecting an AND gate to an inverter.

Figure 2.9: NAND Logic Gate

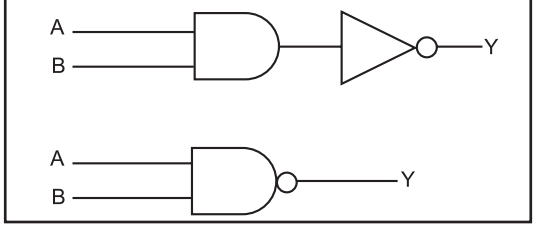


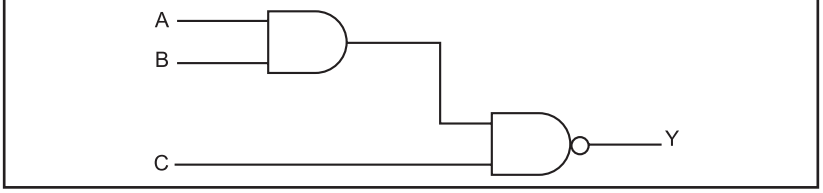
Table 2.4: Truth table

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

Another way to understand NAND logic gate: its output will only be at"0" when all its inputs are also at "1". Otherwise its output will be "1".

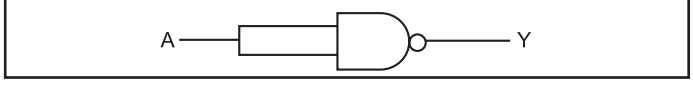
If you need more inputs, just draw them on the symbol shown in Figure 2.9. However, if you want to create more inputs using gates with fewer inputs, you cannot connect them using the same idea shown in Figure 2.4. You need to use AND gates for the "extra" inputs (you can link them together to expand the number of inputs like shown in Figure 2.4) and a NAND gate for the "final" gate, see Figure 2.10.

Figure 2.10: How to Expand the Number of Inputs of a NAND Gate



You can also easily transform NAND and NOR gates into inverters by shorting their inputs, like shown in Figure 2.11. This is a very common trick. For example, you need an inverter in your circuit and have some NAND gates available at a given integrated circuit. Instead of adding another integrated circuit to your project just to have one inverter (what would increase the final circuit size and also the cost), you may want to use one of the available NAND gates.

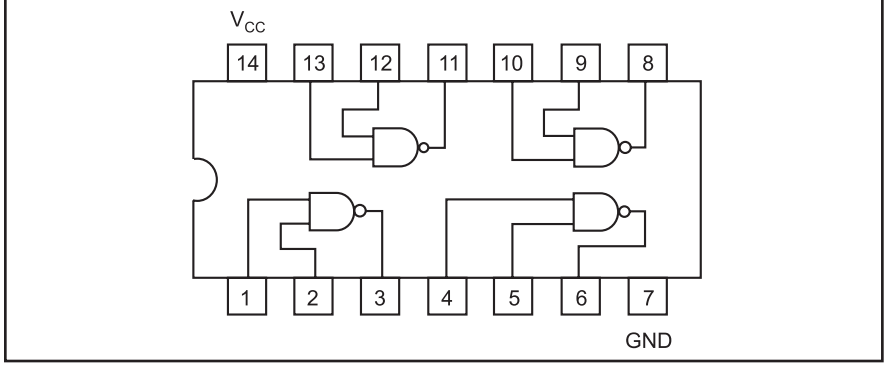
Figure 2.11: Transforming a NAND Gate into an Inverter



The most famous NAND gate integrated circuit is 7400 and you can its pinout in Figure 2.12. Of course there are several other integrated circuits that provide NAND gates with more inputs. For example, 7411 provides three three-input NAND gates and 7430 provides one eight-input NAND gate.

Notes

Figure 2.12: 7400 Integrated Circuit Provides Four Two-Inputs NAND Gates



Caution To avoid the short circuiting use caution when wiring chip and be sure to check which pins are outputs and inputs.

2.1.5 The NOR Gate

The "N" letter on NOR stands for NOT, meaning that NOR logic gate is an OR gate with an inverter attached. So, its output is the opposite from OR. Its symbol is the same of OR but with a "o" on its output, meaning that the output is inverted. You can build yourself a NOR gate by connecting an OR gate to an inverter.

Figure 2.13: NOR Logic Gate

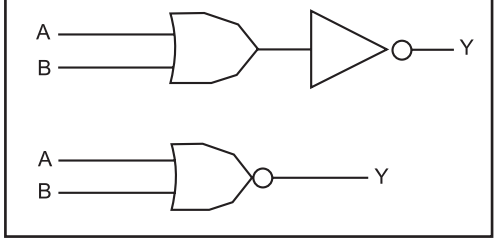


Table 2.5: Truth Table

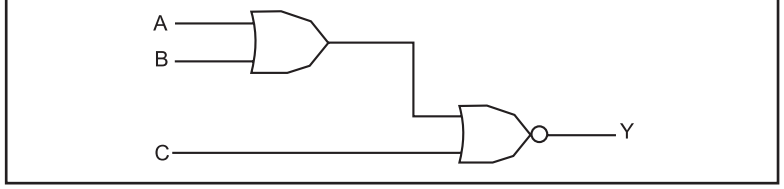
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

Another way to understand NOR logic gate: its output will only be at "1" when all its inputs are at "0". Otherwise its output will be "0".

If you need more inputs, just draw them on the symbol shown in Figure 2.13. However, if you want to create more inputs using gates with fewer inputs, you cannot connect them using the same idea shown in Figure 2.11. You need to use OR gates for the "extra" inputs (you can put them together to expand the number of inputs like shown in Figure 2.11) and a NOR gate for the "final" gate, see Figure 2.14.

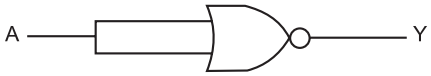
Notes

Figure 2.14: How to Expand the Number of Inputs of a NOR Gate



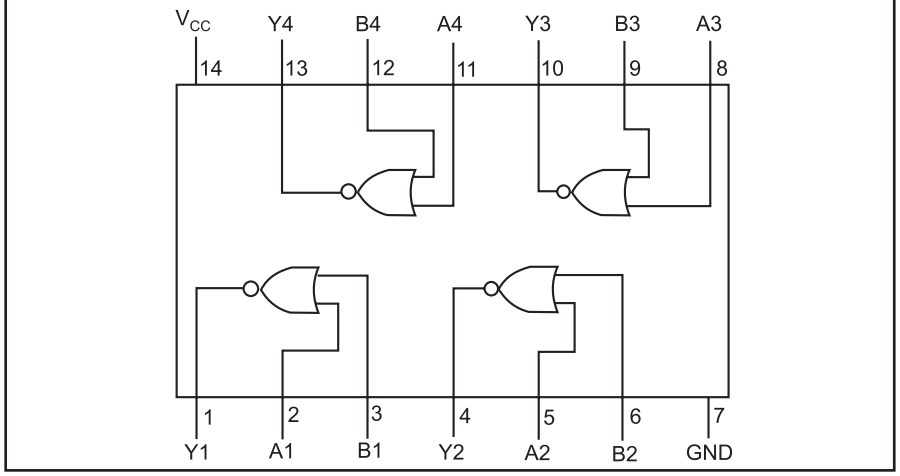
You can also easily transform NAND and NOR gates into an inverter by shorting their inputs, like shown in Figure 2.15. This is a very common trick. For example, you need an inverter in your circuit and have some NOR or NAND gates available at a given integrated circuit. Instead of adding another integrated circuit to your project just to have one inverter (what would increase the final circuit size and also the cost), you may want to use one of the available NAND or NOR gates.

Figure 2.15: Transforming a NOR Gate into an Inverter



The most famous NOR gate integrated circuit is 7402 and you can its pinout in Figure 2.16. Pay attention because the location of the inputs and outputs on this integrated circuit is different from the other integrated circuits we have seen before. Of course there are several other integrated circuits that neither provide NOR gates with more inputs. For example, 7427 provides three three-input NOR gates.

Figure 2.16: 7402 Integrated Circuit Provides Four Two-Inputs NOR Gates



Task Implement a 2-input AND gate using NAND gate only and implement a 2-input AND gate using NOR gates only.

Self Assessment

Choose the correct answer:

1. The output of a logic gate is 1 when all its inputs are at logic 0 the gate is either ..... .

   (*a*) a NAND or an EX-OR (*b*) an OR or an EX-NOR

   (*c*) an AND or an EX-OR (*d*) a NOR or an EX-NOR

Notes

2. The NAND gate output will be low if the two inputs are ..... .

   (*a*) 00 (*b*) 01

   (*c*) 10 (*d*) 11

3. The gates required to build a half adder are ..... .

   (*a*) EX-OR gate and NOR gate (*b*) EX-OR gate and OR gate

   (c) EX-OR gate and AND gate (*d*) Four NAND gates.

4. How many AND gates are required to realize Y = CD+EF+G ?

   (*a*) 4 (*b*) 5

   (*c*) 3 (*d*) 2

5. When an input signals A=11001 is applied to a NOT gate serially, its output signal is..... .

   (*a*) 00111 (*b*) 00110

   (*c*) 10101 (*d*) 11001

6. How many two-input AND gates and two-input OR gates are required to realize Y = BD+CE+AB

   (*a*) 1, 1 (*b*) 4, 2

   (*c*) 3, 2 (*d*) 2, 3

7. Which of the following are known as universal gates ..... .

   (*a*) NAND & NOR (*b*) AND & OR

   (*c*) XOR & OR (*d*) None

2.1.6 XOR Gate

XOR stands for exclusive OR. XOR gate compares two values and if they are different its output will be "1." XOR operation is represented by the symbol ⊕. So Y = A ⊕ B. You can see XOR logic gate symbol in Figure 2.6 and its truth table right below it.

Figure 2.17: XOR Logic Gate

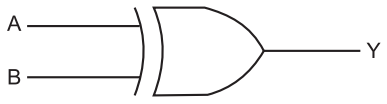


Table 2.6: Truth Table

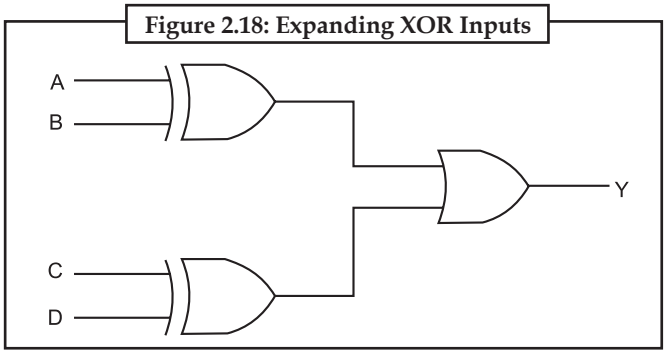
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

So its output will only be at "0" when all its inputs have the same value. Otherwise its output will be "1."

If you need more than two inputs, you will need to add an OR gate like shown in Figure 2.18.

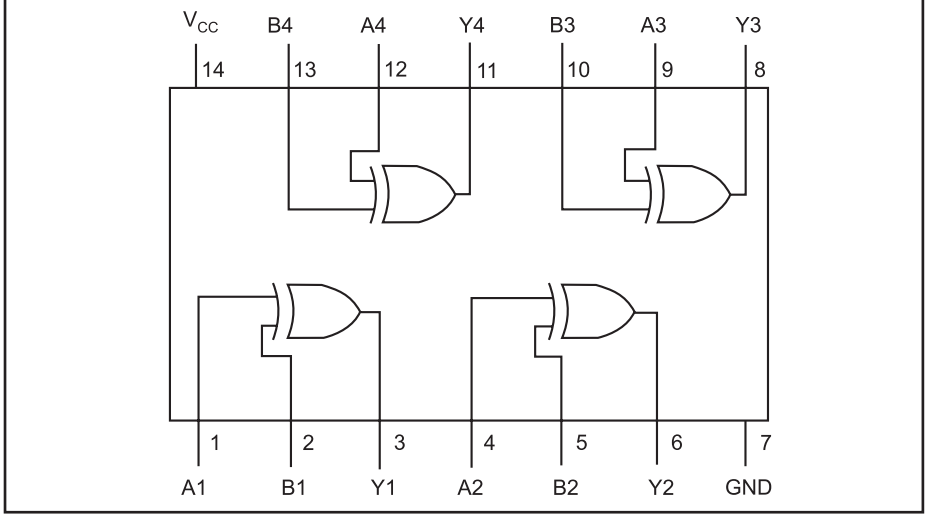
Notes

Figure 2.18: Expanding XOR Inputs



The most famous XOR gate integrated circuit is 7486 and you can its pinout in Figure 2.19.

Figure 2.19: 7486 Integrated Circuit Provides Four Two-Inputs XOR Gates



2.1.7 XNOR Gate

XNOR stands for exclusive NOR and is an XOR gate with its output inverted. So, its output is at "1" when the inputs have the same value and "0" when they are different. XNOR operation is represented by the symbol (•). So Y = A (•) B. You can see XNOR logic gate symbol in Figure 2.20 and its truth table right below it.

Figure 2.20: XNOR Logic Gate

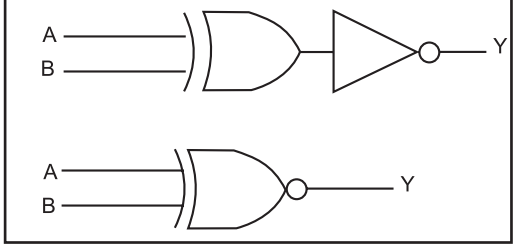


Table 2.7: Truth Table

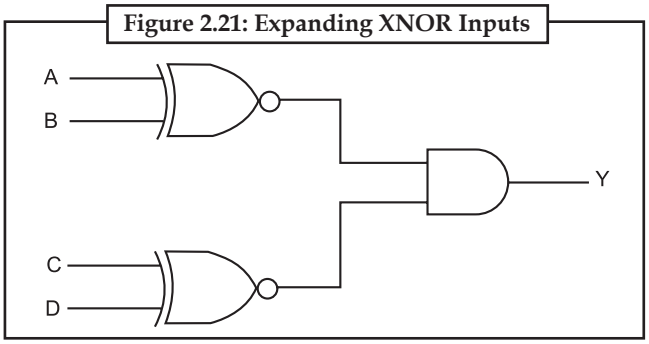
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

Notes

So its output will only be at "1" when all its inputs have the same value. Otherwise its output will be "0".

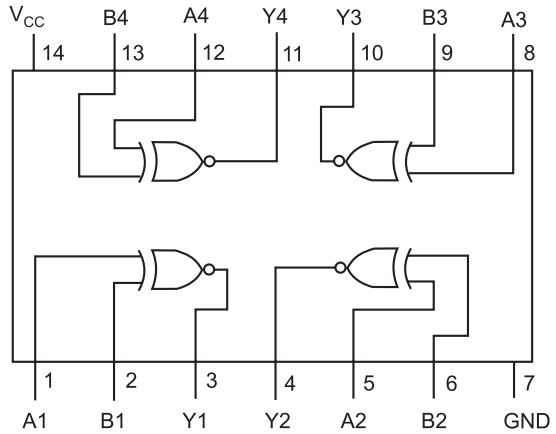
If you need more than two inputs, you will need to add an AND gate like shown in Figure 2.21. Another way is to use the circuit shown in Figure 2.18 adding an inverter on its output.

Figure 2.21: Expanding XNOR Inputs



As an example of integrated circuit with XNOR gates we have 747266 and you can check its pinout in Figure 2.22.

Figure 2.22: 747266 Integrated Circuit Provides Four Two-Inputs XNOR Gates



2.1.8 Non-Inverter or Buffer

At a non-inverter, also known as buffer, the value entered on its input will be found on its output. You may think that this is a crazy logic gate, since it does nothing. That is not true, it has several important applications on digital electronics, as we will explain below:

Figure 2.23: Non-Inverter or Buffer

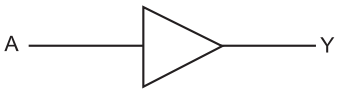


Table 2.8: Truth Table

A	Y
0	0
1	1

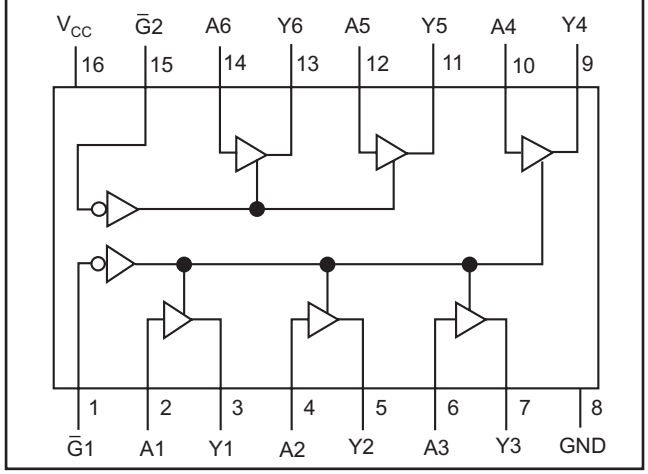
A typical application for a buffer is to increase the fan-out of a given logic gate. Fan-out is the maximum number of gates a given integrated circuit is capable of being connected to. For example, if a given logic gate has a fan-out of 3 gates; its output can be only connected directly to three

Notes

other logic gates. If you need to connect its output to more logic gates, you can use a buffer to increase the number of logic gates you can connect this output to.

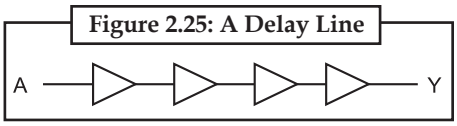
Another application for the buffer is the use of a circuit where the buffer is controlled by a gate (74367 integrated circuit is a good example). In this application this logic gate will work like a gate: it will only replicate what is on its input when its gate pin is activated.

Figure 2.24: 74367 Integrated Circuit Pinout



One more application for a non-inverter is to create a delay line. Since each integrated circuit delays a little bit to replicate what is on its input on its output, a non-inverter can be used to delay the signal. This idea is used on some digital oscillator circuits, for example. If you take the circuit in Figure 2.25, if each gate delays the signal 10 ns (nanoseconds), with four gates we will have a 40 ns delay line.

Figure 2.25: A Delay Line



Another very common application for both non-inverters and inverters is to drive circuits that need more current or need to work with voltages different from 5 V as ”1“.

2.1.9 Open Collector and Open Drain

As we said before,”1“ means 5 V. Sometimes you need a higher voltage for controlling a device that does not work with 5 V. You may want to control a 12 V relay, for example. Also, sometimes you may want to control a 5 V circuit but it drives more current than a standard integrated circuit can deliver. In those cases, you can use open-collector configuration.

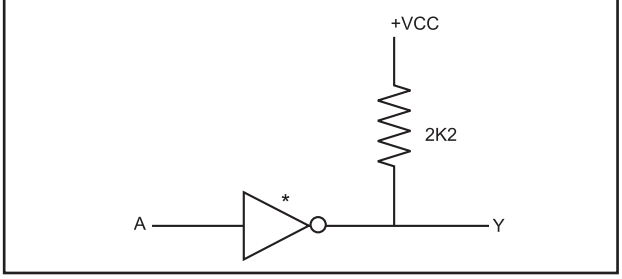
Integrated circuits from the 74xxx series (all integrated circuit examples we are giving on this tutorial) are based on a technology called TTL, Transistor-Transistor Logic. Open collector means that the transistor used on the output of the gate does not have its collector internally connected to the integrated circuits VCC (voltage). So, you have to do this connection by yourself. This means that you need to install an external resistor (called ”pull-up“) between the output and VCC to make the circuit work. The good thing is that VCC does not need to be the +5 V power supply. You can install it to a +12 V power supply and feed your 12 V relay, for example.

The term open-drain is used for CMOS integrated circuits and is exactly the same thing.

Open collector or open-drain gates are usually marked with an asterisk.

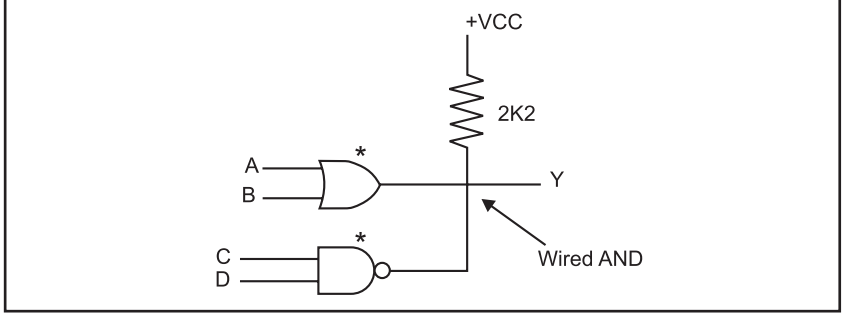
Notes

Figure 2.26: Example of Open Collector Circuit



Open collector configuration has far more applications. The most common is a technique called wired-AND, where the junction works as an AND gate. See the example in Figure 2.27. The output Y will be equal to (A OR B) AND (C NAND D). The junction will work as an AND gate.

Figure 2.27: Example of Wired-AND Using Open Collector Logic Gates



Some open collector integrated circuits include 7403 (NAND gates, same pinout as 7400), 7405 (inverters, same pinout as 7404), 7409 (AND gates, same pinout as 7408) and 7433 (NOR gates, same pinout as 7402), just to mention a few examples.

Task Connect the two circuits using a modulo-2 adder (EX-OR gate, i.e. IC7486).

2.2 Tri-State Gates

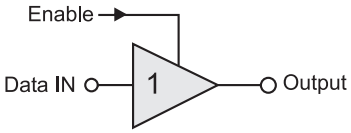
A Tri-state Gate can be thought of as an input controlled switch which has an output that can be electronically turned "ON" or "OFF" by means of an external "Control" or "Enable" signal input. This control signal can be either logic "0" or a logic "1" type signal resulting in the Tri-state gate being in one state allowing its output to operate normally giving either logic "0" or logic "1" output. But when activated in the other state it disables or turns "OFF" its output producing an open circuit condition that is neither "high" or "low", but instead gives an output state of very high impedance, high-Z, or more commonly Hi-Z. Then this type of device has two logic state inputs, "0" or a "1" but can produce three different output states, "0", "1" or "Hi-Z" which is why it is called a "3-state" device.

There are two different types of tri-state gate, one whose output is controlled by an "Active-HIGH" control signal and the other which is controlled by an "Active-LOW" control signal, as shown below.

Active "HIGH" Tri-state Gates

Notes

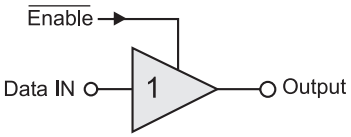
Table 2.9: Symbol and Truth Table

Symbol	Truth Table		
 **Tri-state Gate**	Enable	A	Q
	1	0	0
	1	1	1
	0	0	Hi-Z
	0	1	Hi-Z
Read as Output = Input if Enable is equal to "1"			

An Active-high tri-state gate is activated when a logic level "1" is applied to its "enable" control line and the data passes through from its input to its output. When the enable control line is at logic level "0", the gate output is disabled and a high impedance condition, Hi-Z, is present on the output.

2.2.1 Active "LOW" Tri-state Gate

Table 2.10: Symbol and Truth Table

Symbol	Truth Table		
 **Tri-state Gate**	Enable	A	Q
	0	0	0
	0	1	1
	1	0	Hi-Z
	1	1	Hi-Z
Read as Output = Input if Enable is NOT equal to "1"			

An Active-low Tri-state Gate is the opposite to the above, and is activated when a logic level "0" is applied to its "enable" control line. The data passes through from its input to its output. When the enable control line is at logic level "1", the gate output is disabled and a high impedance condition, Hi-Z, is present on the output.

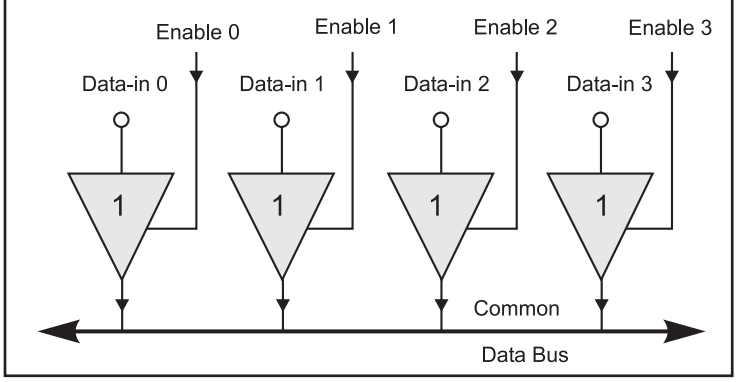
2.2.2 Tri-state Gate Control

The Tri-state Gate is used in many electronic and microprocessor circuits as they allow multiple logic devices to be connected to the same wire or bus without damage or loss of data. For example, suppose we have a data line or data bus with some memory, peripherals, I/O or a CPU connected to it. Each of these devices is capable of sending or receiving data onto this data bus. If these devices start to send or receive data at the same time a short circuit may occur when one device outputs to the bus a logic "1" the supply voltage, while another is set at logic level "0" or ground, resulting in a short circuit condition and possibly damage to the devices.

Then, the Tri-state Gate can be used to isolate devices and circuits from the data bus and one another. If the outputs of several Tri-state Buffers are electrically connected together Decoders are used to allow only one Tri-state Buffer to be active at any one time while the other devices are in their high impedance state. An example of Tri-state Gates connected to a single wire or bus is shown in Figure 2.28.

Notes

Figure 2.28: Tri-state Gate Control



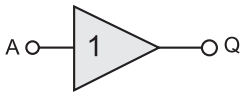
It is also possible to connect Tri-state Gate "back-to-back" to produce a Bi-directional Gate circuit with one "active-high buffer" connected in parallel but in reverse with one "active-low buffer". Here, the "enable" control input acts more like a directional control signal causing the data to be both read "from" and transmitted "to" the same data bus wire.

A **digital gate** is another single input device that does not invert or perform any type of logical operation on its input signal as its output exactly matches that of its input signal. In other words, its output equals its input. It is a "non-inverting" device and so will give us the Boolean expression of: Q = A.

Then we can define the operation of a single input digital gate as being:

"If A is true then Q is true"

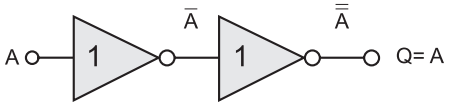
Table 2.11: Truth Table

Symbol	Truth Table	
	A	Q
	0	0
A Tri-state Gate	1	1
Boolean Expression **Q = A**	Read as **A** gives **Q**	

The Digital Tri-state Gate can also be made by connecting together two NOT gates as shown below: The first will "invert" the input signal A and the second will "re-invert" it back to its original level.

2.2.3 Double Inversion Using NOT Gates

Figure 2.29: Tri-state Gate Control

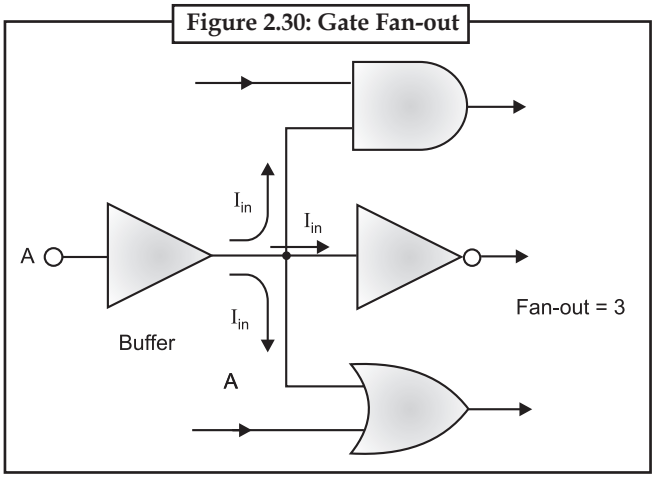


You may think "what is the point of a Digital Gate", if it does not alter its input signal in any way or make any logical operations like the AND or OR gates, then why not use a piece of wire instead and that is a good point. But a non-inverting digital Gate has many uses in digital electronic circuits, as they can be used to isolate other gates or circuits from each other or they can be used to drive high current loads such as transistor switches because their output drive capability is much higher than their input signal requirements, in other words gates are used for power amplification giving them a high fan-out capability.

Notes

2.2.4 Gate Fan-out Example

Figure 2.30: Gate Fan-out



Fan-out is the output driving capability or output current capability of a logic gate giving greater power amplification of the signal. It may be necessary to connect more than just one logic gate to the output of another or to switch a high current load such as an LED, then a Gate will allow us to do just that by having a high fan-out rating of up to 50.

Caution

Beware while connecting a variable DC voltage source to the NAND gate, input must always stay in the range 0 to +5 V otherwise the circuit will be destroyed.

Case Study

Digital Logic Circuit

Binary logic was first proposed by 19th-century British logician and mathematician George Boole, who in 1847 invented a two-valued system of algebra that represented logical relationships and operations. This system of algebra, called Boolean algebra, was used by German engineer Konrad Zuse in the 1930s for his Z1 calculating machine. It was also used in the design of the first digital computer in the late 1930s by American physicist John Atanasoff and his graduate student Clifford Berry. During 1944–1945 Hungarian-born American mathematician John von Neumann suggested using the binary arithmetic system for storing programs in computers. In the 1930s and 1940s British mathematician Alan Turing and American mathematician Claude Shannon also recognized how binary logic was well suited to the development of digital computers.

Functions Performed by Logic Circuits

"True" can be represented by a 1 and "false" by a 0, and in logic circuits the numerals appear as signals of two different voltages. Logic circuits are used to make specific true-false decisions based on the presence of multiple true-false signals at the inputs. The signals may be generated by mechanical switches or by solid-state transducers. Once the input signal has been accepted and conditioned (to remove unwanted electrical signals, or "noise"), it is processed by the digital logic circuits. The various families of digital logic devices, usually integrated circuits, perform a variety of logic functions through logic gates, including "OR", "AND", and "NOT", and combinations of these (such as "NOR", which includes both OR and NOT).

Contd...

Notes

Types of Logic Components

One widely used logic family is the transistor-transistor logic (TTL). Another family is the complementary metal oxide semiconductor logic (CMOS), which performs similar functions at very low power levels but at slightly lower operating speeds. Several other, less popular families of logic circuits exist, including the currently obsolete resistor-transistor logic (RTL) and the emitter coupled logic (ECL), the latter used for very-high-speed systems.

Logical Gates

The elemental blocks in a logic device are called digital logic gates.

An AND gate has two or more inputs and a single output. The output of an AND gate is true only if all the inputs are true.

An OR gate has two or more inputs and a single output. The output of an OR gate is true if any one of the inputs is true and is false if all of the inputs are false.

An INVERTER has a single input and a single output terminal and can change a true signal to a false signal, thus performing the NOT function.

An NAND gate has two or more inputs and a single output. The output of an NAND gate is true if any one of the inputs is false and is false if all the inputs are true.

An NOR gate has two or more inputs and a single output. The output of an NOR gate is true if all the inputs are false and is false if the inputs are different.

An EXCLUSIVE OR gate has two or more inputs and a single output. The output of an EXCLUSIVE OR gate is true if the inputs are different and is false if the inputs are the same.

Obs: You can easily observe how the NAND gate can be emulated by two other gates (AND, NOT). The output of the AND gate is connected to the input of the NOT gate.

The output of the NAND gate is true if one of the inputs is false. Now let's verify: We put true and false on the inputs of the AND gate. This gate now returns false answer (0).This false answer is the input of the NOT gate. This gate returns true (opposite of false). You can now see that the answer is similar to the one witch NAND returned.

Questions:

1. Give the cause through which digital logic circuit was developed.
2. What are the logic components?

Self Assessment

Choose the correct answer:

8. The universal gates only are NOT, AND, OR.
   (*a*) True (*b*) False
9. How many two-input AND and OR gates are required to realize Y=CD+EF+G ?
   (*a*) 2,2 (*b*) 2,3
   (*c*) 3,3 (*d*) none of these
10. A universal logic gate is one which can be used to generate any logic function. Which of …….the following is a universal logic gate?
    (*a*) OR (*b*) AND
    (*c*) XOR (*d*) NAND

Notes

2.3 Summary

- Logic gates are the basic components in digital electronics. They are used to create digital circuits and even complex integrated circuits.
- The AND logic gate performs an"AND" logic operation, which is a multiplication.
- An OR logic gate performs an"OR" logic operation, which is an addition.
- A NAND logic gate is an AND gate with an inverter attached.
- A NOR logic gate is an OR gate with an inverter attached. So, its output is the opposite from OR.
- The XOR stands for exclusive OR. XOR gate compares two values and if they are different its output will be "1."
- The XNOR stands for exclusive NOR and is an XOR gate with its output inverted. So, its output is at "1" when the inputs have the same value and "0" when they are different.
- Fan-out is the maximum number of gates a given integrated circuit is capable of being connected to.
- A Tri-state Gate can be thought of as an input controlled switch which has an output that can be electronically turned "ON" or "OFF" by means of an external "Control" or "Enable" signal input.
- The control signal of tri-state gate can be either a logic "0" or a logic "1" type signal resulting in the Tri-state gate being in one state allowing its output to operate normally giving either logic "0" or logic "1" output.

2.4 Keywords

***AND gate*:** The AND gate is so named because, if 0 is called "false" and 1 is called "true", the gate acts in the same way as the logical "and" operator.

***Inverter*:** A logical inverter, sometimes called a NOT gate to differentiate it from other types of electronic inverter devices, has only one input. It reverses the logic state.

***Logic gate*:** A logic gate is an elementary building block of a digital circuit. Most logic gates have two inputs and one output.

***NAND gate*:** The NAND gate operates as an AND gate followed by a NOT gate. It acts in the manner of the logical operation "and" followed by negation. The output is "false" if both inputs are "true". Otherwise, the output is "true".

***NOR gate*:** The NOR gate is a combination OR gate followed by an inverter. Its output is "true" if both inputs are "false". Otherwise, the output is "false".

***OR gate*:** The OR gate gets its name from the fact that it behaves after the fashion of the logical inclusive "or". The output is "true" if either or both of the inputs are "true". If both inputs are "false", then the output is "false".

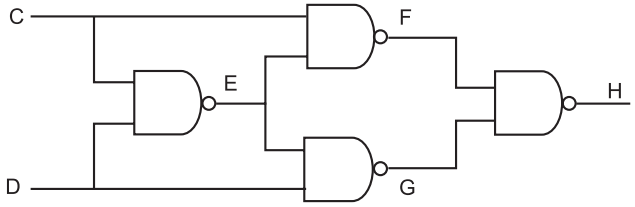
***XNOR gate*:** The XNOR (exclusive-NOR) gate is a combination XOR gate followed by an inverter. Its output is "true" if the inputs are the same and "false" if the inputs are different.

***XOR gate*:** The XOR (exclusive-OR) gate acts in the same way as the logical "either/or." The output is "true" if either, but not both, of the inputs are "true". The output is "false" if both inputs are "false" or if both inputs are "true".

Notes

Lab Exercise

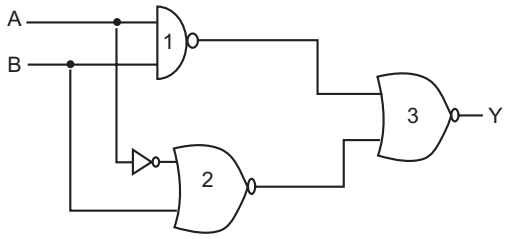
1. The circuit shown in the below figure are assembled from only NAND gates. Draw a truth table, including the intermediate points E, F and G, and state which single gate the output H represents.



2. Design a combination of gates that will accept four inputs A through D and provide a single output that will be true if two, three, or four of the inputs are set.

2.5 Review Questions

1. How many types of logic gates are there? Explain in detail.
2. What do you understand by inverter? Explain the suitable truth table.
3. Describe the double inversion using NOT gates.
4. Explain the tri-state gate control and draw the circuit diagram.
5. What is Tri-state logic and explain Tri-state logic inverter with the help of a circuit diagram. Give its Truth Table.
6. Write the truth table of NOR gate.
7. Find the Boolean expression for the logic circuit shown below:



8. What are universal gates? Construct a logic circuit using NAND gates only for the expression x = A (B + C).
9. With relevant logic diagram and truth table explain the working of a two input EX-OR gate.
10. What is a universal gate? Give examples. Realize the basic gates with any one universal gate.

Answers to Self Assessment

1. (*a*) 2. (*a*) 3. (*d*) 4. (*b*) 5. (*d*)
6. (*d*) 7. (*d*) 8. (*a*) 9. (*a*) 10. (*c*)

2.6 Further Readings

Books

Digital Electronics – Principles and Applications, by R.L. Tokheim.

Modern Digital Electronics, by R.P. Jain.

Online link

www.watson.ibm.com/leo/introelect/IntroElectro_handout.pdf

Notes

Unit 3: Boolean Algebra

CONTENTS

Objectives

After studying this unit, you will be able to:

- Explain the Boolean algebra
- Describe the Boolean theorems
- Explain the DeMorgan's theorems

Introduction

Boolean logic forms the basis for computation in modern binary computer systems. You can represent any algorithm, or any electronic computer circuit, using a system of Boolean equations. This provides a brief introduction to Boolean algebra, truth tables, canonical representation, of Boolean functions, Boolean function simplification, logic design, combinatorial and sequential circuits, and hardware/software equivalence.

The material is especially important to those who want to design electronic circuits or write software that controls electronic circuits. Even if you never plan to design hardware or write software than controls hardware, the introduction to Boolean algebra this unit provides is still important since you can use such knowledge to optimize certain complex conditional expressions within IF, WHILE, and other conditional statements.

On minimizing (optimizing) logic functions uses Veitch Diagrams or Karnaugh Maps. The optimizing techniques this unit uses reduce the number of terms in a Boolean function. You should realize that many people consider this optimization technique obsolete because reducing the number of terms in an equation is not as important as it once was. This unit uses the mapping method as an example of Boolean function optimization, not as a technique one would regularly

Notes

employ. If you are interested in circuit design and optimization, you will need to consult a text on logic design for better techniques.

3.1 Boolean Algebra

You have seen logic gates and how they can operate on binary numbers. Now, we need a mathematical framework for describing the relationship between logic gates and binary numbers. That framework is Boolean algebra. This document of course provides only an introduction to Boolean algebra, refer to dedicated texts for a detailed discussion of the subject.

The English mathematician George Boole (1815–1864) sought to give symbolic form to Aristotle's system of logic. Boole wrote a treatise on the subject in 1854, titled *An Investigation of the Laws of Thought on Which Are Founded the Mathematical Theories of Logic and Probabilities*, which codified several rules of relationship between mathematical quantities limited to one of two possible values: true or false, 1 or 0. His mathematical system became known as Boolean algebra.

All arithmetic operations performed with Boolean quantities have but one of two possible outcomes: either 1 or 0. There is no such thing as "2" or "–1" or "1/2" in the Boolean world. It is a world in which all other possibilities are invalid by fiat. As one might guess, this is not the kind of math you want to use when balancing a chequebook or calculating current through a resistor. However, Claude Shannon of MIT fame recognized how Boolean algebra could be applied to on-and-off circuits, where all signals are characterized as either "high" (1) or "low" (0). His 1938 thesis, titled *A Symbolic Analysis of Relay and Switching Circuits*, put Boole's theoretical work to use in a way Boole never could have imagined, giving us a powerful mathematical tool for designing and analyzing digital circuits.

In this section, you will find a lot of similarities between Boolean algebra and "normal" algebra, the kind of algebra involving so-called real numbers. Just bear in mind that the system of numbers defining Boolean algebra is severely limited in terms of scope, and that there can only be one of two possible values for any Boolean variable: 1 or 0. Consequently, the "Laws" of Boolean algebra often differ from the "Laws" of real-number algebra, making possible such statements as 1 + 1 = 1, which would normally be considered absurd. Once you comprehend the premise of all quantities in Boolean algebra being limited to the two possibilities of 1 and 0, and the general philosophical principle of Laws depending on quantitative definitions, the "nonsense" of Boolean algebra disappears.

3.1.1 Boolean Arithmetic

Let us begin our exploration of Boolean algebra by adding numbers together:

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = 1$$

The first three sums make perfect sense to anyone familiar with elementary addition. The last sum, though, is quite possibly responsible for more confusion than any other single statement in digital electronics, because it seems to run contrary to the basic principles of mathematics. Well, it does contradict principles of addition for real numbers, but not for Boolean numbers. There is no such thing as "2" within the scope of Boolean values. Since the sum "1 + 1" certainly is not 0, it must be 1 by process of elimination.

It does not matter how many or few terms we add together, either. Consider the following sums:

Notes

$$0 + 1 + 1 = 1$$

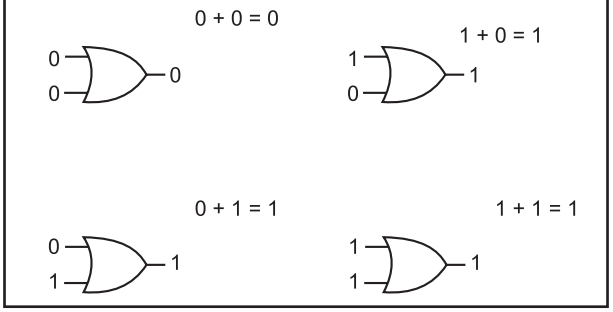
$$1 + 1 + 1 = 1$$

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

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

Take a close look at the two-term sums in the first set of equations. Does that pattern look familiar to you? It should! It is the same pattern of 1's and 0's as seen in the truth table for an OR gate. In other words, Boolean addition corresponds to the logical function of an "OR" gate, as shown in Figure 3.1

Figure 3.1: Boolean Addition and the OR Gate



There is no such thing as subtraction in the realm of Boolean mathematics. Subtraction implies the existence of negative numbers: 5 – 3 is the same thing as 5 + (–3), and in Boolean algebra negative quantities are forbidden. There is no such thing as division in Boolean mathematics, either, since division is really nothing more than compounded subtraction, in the same way that multiplication is compounded addition.

Multiplication is valid in Boolean algebra, and thankfully it is the same as in real-number algebra: anything multiplied by 0 is 0, and anything multiplied by 1 remains unchanged:

$$0 \times 0 = 0$$

$$0 \times 1 = 0$$

$$1 \times 0 = 0$$

$$1 \times 1 = 1$$

This set of equations should also look familiar to you: it is the same pattern found in the truth table for an AND gate. In other words, Boolean multiplication corresponds to the logical function of an "AND" gate.

Like "normal" algebra, Boolean algebra uses alphabetical letters to denote variables. Unlike "normal" algebra, though, Boolean variables are always UPPERCASE letters, never lower-case. Because they are allowed to possess only one of two possible values, either 1 or 0, each and every variable has a complement: the opposite of its value. For example, if variable "A" has a value of 0, then the complement of A has a value of 1. Boolean notation uses a bar above the variable character to denote complementation, like this.

$$\text{If: } A = 0$$

$$\text{Then: } \bar{A} = 1$$

$$\text{If: } A = 1$$

$$\text{Then: } \bar{A} = 0$$

Notes

In written form, the complement of "A" denoted as "A-not" or "A-bar". Sometimes a "prime" symbol is used to represent complementation ('A'). Boolean complementation finds equivalency in the form of the NOT gate.

The basic definition of Boolean quantities has led to the simple rules of addition and multiplication, and has excluded both subtraction and division as valid arithmetic operations. We have symbols for denoting Boolean variables, and their complements.


Caution

Remember that in the world of Boolean algebra, there are only two possible values for any quantity and for any arithmetic operation: 1 or 0.

3.1.2 Boolean Algebraic Identities

In mathematics, an identity is a statement true for all possible values of its variable or variables. The algebraic identity of $x + 0 = x$ tells us that anything (x) added to zero equals the original "anything," no matter what value that "anything" (x) may be. Like ordinary algebra, Boolean algebra has its own unique identities based on the bivalent states of Boolean variables. If A is a Boolean variable, Figure 3.2 below shows the basic Boolean algebraic identities.

Figure 3.2: Basic Boolean Algebraic Identities

Additive	Multiplicative
A + 0 + = A	0A = 0
A + 1 = 1	1A = A
A + A = A	AA = A
$A + \bar{A} = 1$	$A\bar{A} = 0$

3.1.3 Boolean Algebraic Properties

Another type of mathematical identity, called a "property" or a "law," describes how differing variables relate to each other in a system of numbers. Assuming A and B are Boolean numbers, Figure 3.3 lists the Boolean algebraic properties.

Figure 3.3: Basic Boolean Algebraic Properties of Additive Association, Multiplicative Association and Distribution

Additive	Multiplicative
A + B = B + A	AB = BA
A + (B + C) = (A + B) + C	A(BC) = (AB) C
Distributive A(B + C) = AB + AC	

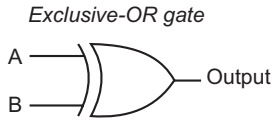
3.1.4 Translating Truth Tables into Boolean Expressions

In designing digital circuits, the designer often begins with a truth table describing what the circuit should do. The design task is largely to determine what type of circuit will perform the function described in the truth table. While some people seem to have a natural ability to look at a truth table and immediately envision the necessary logic gate or relay logic circuitry for the task, there are procedural techniques available for the rest of us. Here, Boolean algebra proves its utility in a most dramatic way.

Notes

As an example, let us take a look at the exclusive-OR (XOR) gate. (Figure 3.4 for convenience).

Figure 3.4: XOR Gate Schematic Symbol and Truth Table Revisited



A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

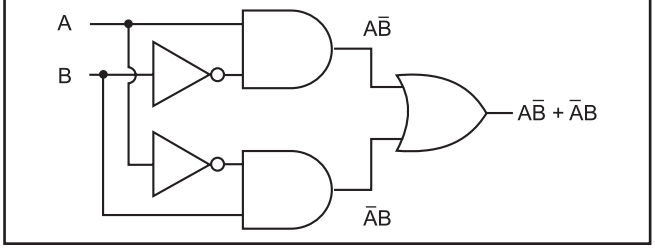
It is not necessarily obvious what kind of logic circuit would satisfy the truth table. However, a simple method for designing such a circuit is found in a standard form of Boolean expression called the Sum-Of-Products, or SOP, form. As you might suspect, a Sum-Of-Products Boolean expression is literally a set of Boolean terms added (summed) together, each term being a multiplicative (product) combination of Boolean variables. An example of an SOP expression would be something like this: ABC + BC + DF, the sum of products "ABC", "BC", and "DF".

Sum-Of-Products expressions are easy to generate from truth tables. All we have to do is examine the truth table for any rows where the output is "high" (1), and write a Boolean product term that would equal a value of 1 given those input conditions. In Figure 3.4, rows 2 and 3 have output high. The product term corresponding to row 2 would be A'B since the term would have a value of 1 if and only if A = 0 and B = 1. Similarly, the product term corresponding to row 3 would be AB'. Now, we join our Boolean products together by addition to create a single Boolean expression for the truth table as a whole. The Δ is the symbol for XOR.

$$A \oplus B = A'B + AB'$$

Now, we can easily translate the right-hand side of the equation above into a circuit, (refer to Figure 3.5).

Figure 3.5: Simplified Gate-Level Schematic of XOR



An alternative to generating a Sum-Of-Products expression to account for all the "high" (1) output conditions in the truth table is to generate a Product-Of-Sums, or POS, expression, to account for all the "low" (0) output conditions instead. For the XOR gate above, a POS expression is (A' + B'). (A + B).

Both the Sum-Of-Products and Products-Of-Sums standard Boolean forms are powerful tools when applied to truth tables. They allow us to derive a Boolean expression and, ultimately, an actual logic circuit from nothing but a truth table, which is a written specification for what we want a logic circuit to do. To be able to go from a written specification to an actual circuit using simple, deterministic procedures means that it is possible to automate the design process for a digital circuit. In other words, a computer could be programmed to design a custom logic circuit from a

Notes

truth table specification! The steps to take from a truth table to the final circuit are so unambiguous and direct that it requires little, if any, creativity or other original thought to execute them.


Caution

Be careful that "Truth table" must be considered while designing the digital circuits.


Did u know?

Stone's representation theorem for Boolean algebra states that every Boolean algebra is isomorphic to a field of sets, as stated by Marshall Harvey Stone in 1936.

3.1.5 Boolean Rules for Simplification

Boolean algebra finds its most practical use in the simplification of logic circuits. If we apply certain algebraic rules to a Boolean equation resulting from a truth table, we will get a simpler equation. The simplified equation may be translated into circuit form for a logic circuit performing the same function with fewer components. If equivalent function may be achieved with fewer components, the result will be increased reliability and decreased cost of manufacture. A few of the Boolean rules for simplification are shown in Figure 3.6.

Figure 3.6: Useful Boolean Rules for Simplification

$$A + AB = A$$

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

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

Proving the rules above requires the use of concepts learned in sections 5 (*a*), (*b*) and (*c*). It proves the first two below and leave the third as an exercise.

$$\begin{aligned} A + AB &= A.(1 + B) \text{ [Distributive property]} \\ &= A \\ A + A'B &= (A + AB) + A'B \text{ [From rule above: A = A+B]} \\ &= A + (AB + A'B) \text{ [Additive association property]} \\ &= A + B.(A + A') \text{ [Distributive property]} \\ &= A + B \end{aligned}$$

Now, we will encapsulate what we learned about the binary number systems, logic gates and Boolean algebra by designing a few common building blocks of digital systems.

3.2 Boolean Theorems

We have seen how Boolean algebra can be used to help analyze a logic circuit and express its operation mathematically. We will continue our study of Boolean algebra by investigating the various Boolean theorems (rules) that can help us to simplify logic expressions and logic circuits. The first group of theorems is given in Figure 3.7. In each theorem, x is a logic variable that can be either a 0 or a 1. Each theorem is accompanied by a logic-circuit diagram that demonstrates its validity.

Theorem (1) states that if any variable is ANDed with 0, the result has to be 0. This is easy to remember because the AND operation is just like ordinary multiplication, where we know that anything multiplied by 0 is 0. We also know that the output of an AND gate will be 0 whenever any input is 0, regardless of the level on the other input.

Notes

Theorem (2) is also obvious by comparison with ordinary multiplication.

Theorem (3) can be proved by trying each case. If $x = 0$, then $0 \cdot 0 = 0$; if $.x = 1$, then $1 \cdot 1 = 1$. Thus, $x \quad x = x$.

Theorem (4) can be proved in the same manner. However, it can also be reasoned that at any time either x or its inverse x has to be at the 0 level, and so there AND product always has to be 0.

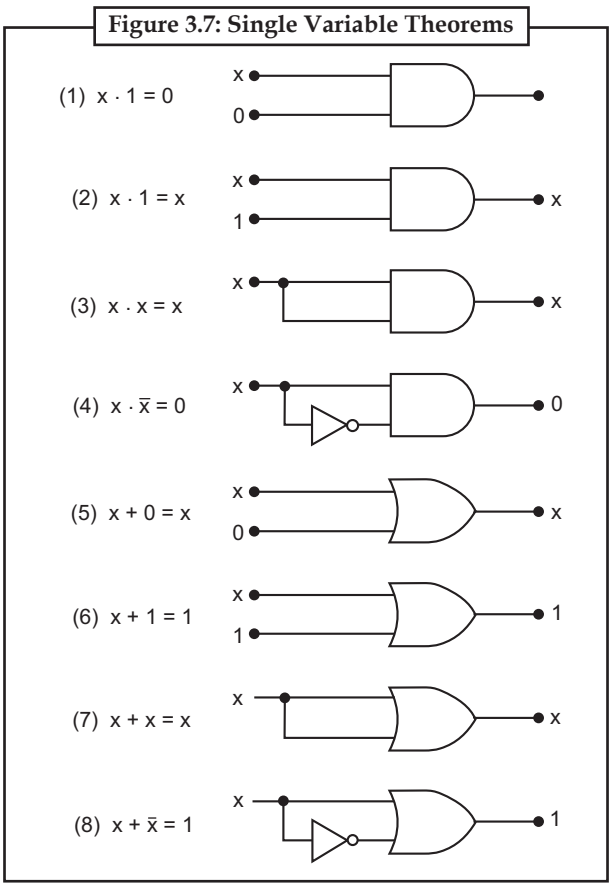
Theorem (5) is straightforward, since 0 added to anything does not affect its value, either in regular addition or in OR addition.

Theorem (6) states that if any variable is ORed with 1, the result will always be 1. Checking this for both values of x: 0 + 1 = 1 and 1 + 1 = 1. Equivalently, we can remember that an OR gate output will be 1 when any input is 1, regardless of the value of the other input.

Theorem (7) can be proved by checking for both values of x: 0 + 0 = 0 and 1 + 1 = 1.

Theorem (8) can be proved similarly, or we can just reason that at any time either x or $\bar{x}$ has to be at the 1 level so that we are always ORing a 0 and a 1, which always results in 1.

Figure 3.7: Single Variable Theorems



Before introducing any more theorems, it should be pointed out that in applying theorems (1) through (8) the variable x may actually represent an expression containing more than one variable. For example, if we have $A\bar{B}\,(\overline{A\bar{B}})$, we can invoke theorem (4) by letting $x = \bullet\, A\bar{B}$. Thus, we can say that $A\bar{B}\,(\overline{A\bar{B}} = 0$. The same idea can be applied to the use of any of these theorems.

Notes

3.2.1 Multivariable Theorems

The theorems presented below involve more than one variable:

(9) $x + y = y + x$

(10) $x \bullet y = y \bullet x$

(11) $x + (y + z) = (x + y) + z = x + y + z$

(12) $x(yz) = (xy)z = xyz$

(13*a*) $x(y + z) = xy + xz$

(13*b*) $(w + x)(y + z) = wy + xy + wz + xz$

(14) $x + xy = x$

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

Theorems (9) and (10) are called the commutative laws. These laws indicate that the order in which we OR Or AND two variables is unimportant; the result is the same.

Theorems (11) and (12) are the associative laws, which state that we can group the variables in an AND expression or OR expression any way we want.

Theorem (13) is the distributive law, which states that an expression can be expanded by multiplying term by term just the same as in ordinary algebra. This theorem also indicates that we can factor an expression. That is, if we have a sum of two (or more) terms, each of which contains a common variable, the common variable can be factored out just like in ordinary algebra. For example, if we have the expression $A\bar{B}C + \bar{A}\bar{B}\bar{C}$, we can factor out the $\bar{B}$ variable:

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

As another example, consider the expression ABC + ABD. Here the two terms have the variables A and B in common, and so $A \bullet B$ can be factored out of both terms. That is,

$$ABC + ABD = AB(C + D)$$

Theorems (9) to (13) are easy to remember and use since they are identical to those of ordinary algebra. Theorems (14) and (15), on the other hand, do not have any counterparts in ordinary algebra. Each can be proved by trying all possible cases for x and y. This is illustrated for theorem (14) as follows:

Case 1. For $x = 0$, $y = 0$,

$$\begin{aligned} x + xy &= x \\ 0 + 0 \bullet 0 &= 0 \\ 0 &= 0 \end{aligned}$$

Case 2. For $x = 0$, $y = 1$,

$$\begin{aligned} x + xy &= x \\ 0 + 0 \bullet 1 &= 0 \\ 0 + 0 &= 0 \\ 0 &= 0 \end{aligned}$$

Case 3. For $x = 1$, $y = 0$,

$$\begin{aligned} x + xy &= x \\ 1 + 1 \bullet 0 &= 1 \\ 1 + 0 &= 1 \\ 1 &= 1 \end{aligned}$$

Notes

Case 4. For $x = 1$, $y = 1$,

$$\begin{aligned} x + xy &= x \\ 1 + 1 \bullet 1 &= 1 \\ 1 + 1 &= 1 \\ 1 &= 1 \end{aligned}$$

Theorem, (14) can also be proved by factoring and using theorems (6) and (2) as follows:

$$\begin{aligned} x + xy &= x\,(1 + y) \\ &= x\,1 \qquad \text{[using theorem (6)]} \\ &= x \qquad \text{[using theorem (2)]} \end{aligned}$$

All of these Boolean theorems can be useful in simplifying a logic expression—that is, in reducing the number of terms in the expression. When this is done, the reduced expression will produce a circuit that is less complex than the one which the original expression would have produced. For now, the following examples will serve to illustrate how the Boolean theorems can be applied.

Example:

Simplify the expression $y = A\bar{B}D + A\bar{B}\bar{D}$.

Solution:

Factor out the common variables $A\bar{B}$ using theorem (13):

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

Using theorem (8), the term in parentheses is equivalent to 1. Thus, $y = A\bar{B} \bullet 1$

$$y = A\bar{B} \quad \text{[using theorem (2)]}$$

Example:

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

Solution:

The expression can be expanded by multiplying out the terms [theorem (13)]:

$$\begin{aligned} y &= A\bar{B}\,(D + \bar{D}) \\ z &= \bar{A} \bullet A + \bar{A} \bullet B + B \bullet A + B \bullet B \end{aligned}$$

Invoking theorem (4), the term $\bar{A} \bullet A = 0$. Also, $B \bullet B = B$ [theorem (3)]:

$$z = O + \bar{A} \bullet B + B \bullet A + B = \bar{A}B + AB + B$$

Factoring out the variable B [theorem (13)], we have

$$z = B(\bar{A} + A + 1)$$

Finally, using theorems (2) and (6),

$$z = B$$

Example:

Simplify $x = ACD + \bar{A}BCD$.

Solution:

Factoring out the common variables CD, we have

$$x = CD(A + \bar{A}B)$$

Notes

Utilizing theorem (15), we can replace $A + \bar{A}B$ by $A + B$, so

$$x = CD(A + B)$$
$$= ACD + BCD$$

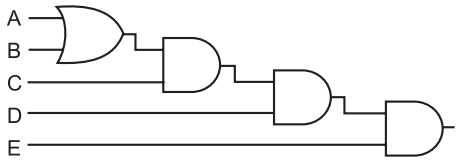
Task Simplify the Boolean expression $F = C\ (B + C)\ (A + B + C)$.

Did u know? Boolean logic forms are the basis for computation in modern binary computer systems.

Self Assessment

Choose the correct answer:

1. Give the relationship that represents the dual of the Boolean property A + 1 = 1.
   (*a*) A * 1 = 1 (*b*) A * 0 = 0
   (*c*) A + 0 = 0 (*d*) A * A = A
2. The best definition of a literal is:
   (*a*) A Boolean variable (*b*) The complement of a Boolean variable
   (*c*) 1 or 2 (*d*) A Boolean variable interpreted literally
3. On simplifying the Boolean expression (A + B + C)(D + E)′ + (A + B + C)(D + E), the best answer is:
   (*a*) A + B + C (*b*) D + E
   (*c*) A′B′C′ (*d*) D′E′
4. What is the Boolean expression for the logic circuit shown below:



   (*a*) C(A+B)DE (*b*) [C(A+B)D+$\bar{E}$]
   (*c*) [[C(A+B)]D]$\bar{E}$ (*d*) ABCDE

3.3 DeMorgan's Theorems

Two of the most important theorems of Boolean algebra were contributed by a great mathematician named DeMorgan. DeMorgan's theorems are extremely useful in simplifying expressions in which a product or sum of variables is inverted.

DeMorgan's first theorems: In words, the complement of a logical sum equals the logical product of the complements. In the term of circuits, a NOR gate equals a bubbled AND gate.

DeMorgan's second theorems: In words, the complement of a logical product equals the logical sum equals the complements. In the term of circuits, a NAND gate is equivalent to a bubbled OR gate.

The two theorems are:

1. $(\overline{x + y}) = \bar{x}\,\bar{y}$

Notes

2. $(\overline{x \bullet y}) = \bar{x} + \bar{y}$

Theorem (1) says that when the OR sum of two variables is inverted, this is the same as inverting each variable individually and then ANDing these inverted variables. Theorem (2) says that when the AND product of two variables is inverted, this is the same as inverting each variable individually and then ORing them. Each of DeMorgan's theorems can be readily proven by checking for all possible combinations of x and y.

Although these theorems have been stated in terms of single variables x and y, they are equally valid for situations where x and/or y are expressions that contain more than one variable. For example, let's apply them to the expression

$(\overline{A\bar{B} + C})$ as shown below:

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

Note that here we treated $A\bar{B}$ as x and C as y. The result can be further simplified since we have a product $A\bar{B}$ that is inverted. Using theorem (2), the expression becomes

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

Notice that we can replace $\bar{\bar{B}}$ by B, so that we finally have

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

This final result contains only inverter signs that invert a single variable.

When using DeMorgan's theorems to reduce an expression, we may break an inverter sign at any point in the expression and change the operator at that point in the expression to its opposite (+ is changed to •, and vice versa). This procedure is continued until the expression is reduced to one in which only single variables are inverted. Two or more examples are given below:

1.
$$\begin{aligned} z &= \overline{A + \bar{B}.C} \\ &= \bar{A} \bullet (\overline{\bar{B} \bullet C}) \\ &= \bar{A} \bullet (\bar{\bar{B}} + \bar{C}) \\ &= \bar{A} \bullet (B + \bar{C}) \end{aligned}$$

2.
$$\begin{aligned} \omega &= \overline{(A + BC) \bullet (D + EF)} \\ &= (\overline{A + BC}) + (\overline{D + EF}) \\ &= (\bar{A} \bullet \overline{BC}) + (\bar{D} \bullet \bar{E}\bar{F}) \\ &= [\bar{A} \bullet (\bar{B} + \bar{C})] + [\bar{D} \bullet (\bar{E} + \bar{F})] \\ &= \bar{A}\bar{B} + \bar{A}\bar{C} + \bar{D}\bar{E} + \bar{D}\bar{F} \end{aligned}$$

DeMorgan's theorems are easily extended to more than two variables. For example, it can be proved that

$$\begin{aligned} \overline{x + y + z} &= \bar{x} \bullet \bar{y} \bullet \bar{z} \\ \bar{x} \bullet \bar{y} \bullet \bar{z} &= \overline{x + y + z} \end{aligned}$$

and so on for more variables. Again, realize that any one of these variables can be an expression rather than a single variable.

3.3.1 Implications of DeMorgan's Theorems

Let us examine these theorems (1) and (2) from the standpoint of logic circuits. First, consider theorem (1),

$$\overline{x + y} = \bar{x} \bullet \bar{y}$$

The left-hand side of the equation can be viewed as the output of a NOR gate whose inputs are x and y. The right-hand side of the equation, on the other hand, is the result of first inverting both x and y and then putting them through an AND gate. These two representations are equivalent and are illustrated in Figure 3.8 (*a*).

Notes

Figure.3.8 (*a*): Equivalent Circuits Implied by Theorem (1): (*b*) Alternative Symbol for the NOR Function

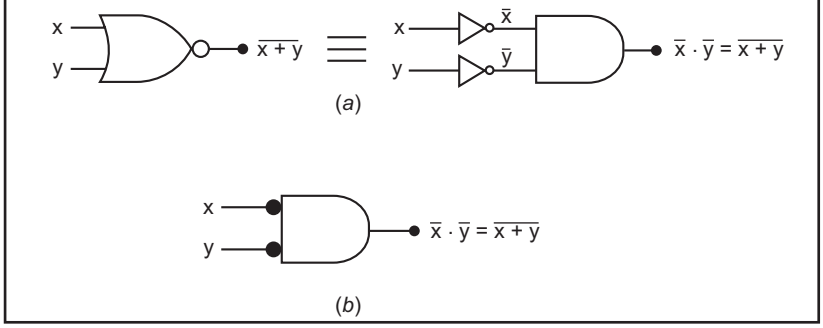
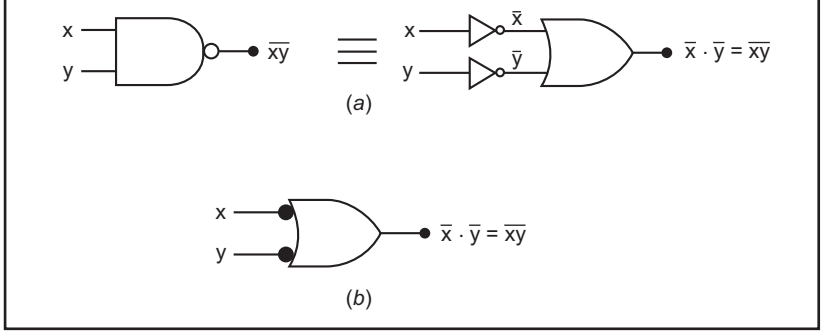


Figure 3.9 (*a*): Equivalent Circuits Implied by Theorem (2); (*b*) Alternative Symbol for the NAND Function



What this means is that an AND gate with INVERTERs on each of its inputs is equivalent to a NOR gate. In fact, both representations are used to represent the NOR function. When the AND gate with inverted inputs is used to represent the NOR function, it is usually drawn as shown in Figure 3.8 (*b*), where the small circles on the inputs represent the inversion operation.

Now consider theorem (2),

$$x\,y \;=\; x + y$$

The left side of the equation can be implemented by a NAND gate with inputs x and y. The right side can be implemented by first inverting inputs x and y and then putting them through an OR gate. These two equivalent representations are show in Figure 3.9(*a*). The OR gate with INVERTERS on each of its inputs is equivalent to the NAND gate. In fact, both representations are used to represent the NAND function, while the OR gate with inverted inputs is used to represent the NAND function.

Example:

Simplify the expression z = $(\bar{A} + C) \bullet (B + \bar{D})$ to one having only single variables inverted.

Solution:

Using theorem (2), we can rewrite this as

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

We can think of this as breaking the large inverter sign down the middle and changing the AND sign (•) to an OR sign (+). Now the term $\overline{(\bar{A} + C)}$ can be simplified by applying theorem (1). Likewise, $\overline{(B + \bar{D})}$ can be simplified:

Notes

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

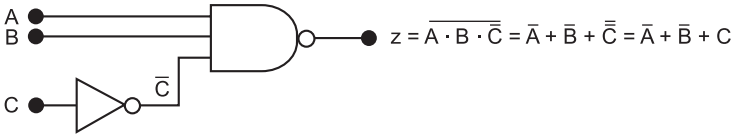
$$= (\bar{\bar{A}} \cdot \bar{C}) + \bar{B} \cdot \bar{\bar{D}}$$

Here we have broken the larger, inverter signs down the middle and replaced the (+) with a(•). Cancelling out the double inversions, we have finally

$$z = A\bar{C} + \bar{B}D$$

Example:

Determine the output expression for the circuit and simplify it using DeMorgan's theorems.



Solution:

The expression for z is $z = \overline{AB\bar{C}}$. Use DeMorgan's theorem to break the large inversion sign:

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

Cancel the double inversions over C to obtain

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

Task Evaluate $x = \bar{A}.B + C(\overline{A.D})$ using the convention A = True and B = False.

Case Study

History of Boolean Algebra

Boolean Algebra is a two-valued algebra, applied earlier to statements and sets which were either true or false and now to switches which are either closed or open, i.e. On or Off respectively. George Boole developed this branch of mathematics in his book "*An Investigation of the Laws of Thought*" now known as symbolic logic. This provided the basic logic for operations on binary numbers (1 and 0). Since modern business machines are based on binary system, the symbolic logic of George Boole was found extremely useful and is being considered as the base of Modern Mathematics.

In the 19th century Symbolic Logic was invented, it was used much later when in the 20th century Claude Shannon discovered the similarity of structure between it and telephone switching circuits. His paper "*A Symbolic Analysis of Relay and Switching Circuits*" made an important contribution to the use of Boolean algebra towards the designing of modern Business Machine based on Binary Number.

Contd...

Notes

Structure:

- Basic Properties
- Boolean Functions
- Boolean Addition
- Electrical Switching System
- Derived Properties
- Canonical Form
- Boolean Multiplication
- Circuits with Composite Operations

$\bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$

↓ Factoring BC out of 1st and 4th terms

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

↓ Applying identity $A + \bar{A} = 1$

$BC(1) + A\bar{B}C + AB\bar{C}$

↓ Applying identity $1A = A$

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

↓ Factoring B out of 1st and 3rd terms

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

↓ Applying rule $\mathbf{A + \bar{A}B = A + B}$ to the $C + A\bar{C}$ term

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

↓ Distributing terms

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

↓ Factoring **A** out of 2nd and 3rd terms

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

↓ Applying rule $\mathbf{A + \bar{A}B = A + B}$ to the $B + \bar{B}C$ term

$BC + A(B + C)$

↓ Distributing terms

$BC + AB + AC$

or Simplified result

$AB + BC + AC$

There are three basic operations in Boolean algebra AND, OR and NOT. These are symbolized by ∩, U and {} respectively in case of the theory of sets. In this unit the more common symbolic plus+, dot '.' and prime (') would be used for the three operations respectively. The similarity would become obvious in the way the present chapter would synthesize and generalize what we have studied earlier and apply it to the end purpose of the designing of the electric circuits.

Questions:

1. Who was the founder of Boolean algebra?
2. Explain the basic structures of Boolean algebra.

Self Assessment

Choose the correct answer:

5. One of DeMorgan's theorems states that $\overline{x + y} = \bar{x}\bar{y}$ and $\overline{x \cdot y} = \bar{x} + \bar{y}$. Simply stated, this means that logically there is no difference between:

   (*a*) a NOR and an AND gate with inverted inputs

Notes

(*b*) a NAND and an OR gate with inverted inputs

(*c*) an AND and a NOR gate with inverted inputs

(*d*) a NOR and a NAND gate with inverted inputs

6. The commutative law of Boolean addition states that A + B = A × B.

(*a*) True (*b*) False

7. The systematic reduction of logic circuits is accomplished by:

(*a*) using Boolean algebra (*b*) symbolic reduction

(*c*) TTL logic (*d*) using a truth table

8. An AND gate with schematic "bubbles" on its inputs performs the same function as a (n) ..... gate.

(*a*) NOT (*b*) OR

(*c*) NOR (*d*) NAND

3.4 Summary

- Boolean algebra are truth tables, canonical representation, of Boolean functions, Boolean function simplification, logic design, combinatorial and sequential circuits, and hardware/ software equivalence.
- Multiplication is valid in Boolean algebra, and thankfully it is the same as in real-number algebra: anything multiplied by 0 is 0, and anything multiplied by 1 remains unchanged.
- Boolean theorems (rules) that help us to simplify logic expressions and logic circuits.
- Sum-Of-Products expressions are easy to generate from truth tables. All we have to do is examine the truth table for any rows where the output is "high" (1), and write a Boolean product term that would equal a value of 1 given those input conditions.
- Products-Of-Sums standard Boolean forms are powerful tools when applied to truth tables. They allow us to derive a Boolean expression and, ultimately, an actual logic circuit from nothing but a truth table, which is a written specification for what we want a logic circuit to do.

3.5 Keywords

***Boolean algebra*:** It is used to help analyze a logic circuit and express its operation mathematically.

***Boolean quantities*:** It has led to the simple rules of addition and multiplication, and has excluded both subtraction and division as valid arithmetic operations.

***Boolean theorem*:** It is useful in simplifying a logic expression—that is, in reducing the number of terms in the expression.

***DeMorgan's theorem*:** These are extremely useful in simplifying expressions in which a product or sum of variables is inverted.

***Distributive law*:** It states that an expression can be expanded by multiplying term by term just the same as in ordinary algebra.

Lab Exercise

1. Construct an OR gate connecting NAND gate.
2. Prepare an OR gate using AND and NOR gate.

Notes

3.6 Review Questions

1. Define Boolean algebra.
2. What is the difference between Boolean arithmetic and Boolean algebraic identities?
3. Explain the Boolean algebraic properties and draw the circuit diagram.
4. Write a Boolean expression for an OR gate having A and B as inputs and y as the output.
5. Write a Boolean expression for an AND gate having A and B as inputs and y as the output.
6. Explain how to translate truth tables into Boolean expressions. Write the procedure.
7. Write the Boolean expression for a 2-input NOR gate.
8. Write the Boolean expression for a 2-input NAND gate. Draw the circuit diagram.
9. Write the Boolean theorems.
10. Write first and second DeMorgan's theorems.

Answers to Self Assessment

1. (*b*) 2. (*c*) 3. (*a*) 4. (*a*)
5. (*a*) 6. (*b*) 7. (*a*) 8. (*c*)

3.7 Further Reading

Books *Boolean Algebra and Its Applications*, by J. Eldon Whitesitt.

Online link http://books.google.co.in/books?id=VDIuSvrDVxwC&printsec=frontcover&dq=boolean+algebra&hl=en&sa=X&ei=mpn5ToDoN5C3rAen1fD8Dw&ved=0CC8Q6AEwAA#v=onepage&q=boolean%20algebra&f=false

Notes

Unit 4: Minimization of Boolean Algebra

CONTENTS

Objectives

After studying this unit, you will be able to:

- Explain the minterms and maxterms
- Understand sum of product and product of sum
- Discuss the Karnaugh map

Introduction

Boolean expression is expressed in terms of either sum of product or product of sum. With the sum of product method the design starts with a truth table that summarises the desired input-output condition. The next step is to convert the truth table into an equivalent sum of product equation.

Product of sum equation, you can simplify it with Boolean algebra. Alternatively, you may prefer simplification based on the Karnaugh map. Introduces two standard forms of expressing Boolean functions, the minterms and maxterms, also known as standard products and standard sums respectively. A procedure is also presented to show how one can convert one form to the other.

Karnaugh maps are used for many small design problems. It is true that many larger designs are done using computer implementations of different algorithms. However designs with a small number of variables occur frequently in interface problems and that makes learning Karnaugh maps worthwhile. In addition, if you study Karnaugh maps you will gain a great deal of insight into digital logic circuits.

A Karnaugh map comprises a box for every line in the truth table; the binary value for each box is the binary value of the input terms in the corresponding table row.

Notes

Unlike a truth table, in which the input values typically follow a standard binary sequence (00, 01, 10, 11), the Karnaugh map's input values must be ordered such that the values for adjacent columns vary by only a single bit, for example, 00, 01, 11, and 10. This ordering is known as a Gray code. We use a Karnaugh map to obtain the simplest possible Boolean expression that describes a truth table.

In this section we will examine some Karnaugh maps for three and four variables. As we use them be particularly tuned in to how they are really being used to simplify Boolean functions.

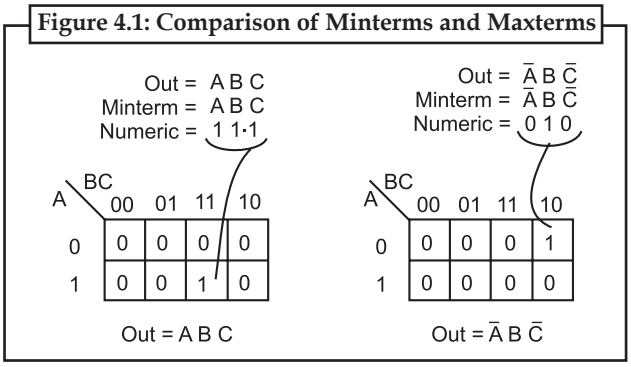
4.1 Minterms and Maxterms

Any Boolean expression may be expressed in terms of either minterms or maxterms. To do this we must first define the concept of a literal. A literal is a single variable within a term which may or may not be complemented. For an expression with N variables, minterms and maxterms are defined as follows:

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

The establish a formal procedure for minterms for comparison to the new procedure for maxterms.

Figure 4.1: Comparison of Minterms and Maxterms



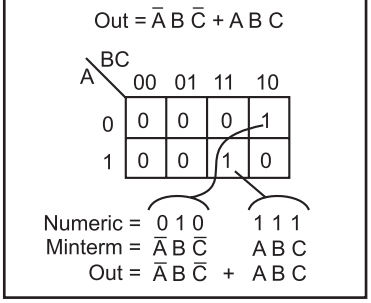
A minterm is a Boolean expression resulting in 1 for the output of a single cell, and 0s for all other cells in a Karnaugh map, or truth table. If a minterm has a single 1 and the remaining cells as 0s, it would appear to cover a minimum area of 1s. The illustration above left shows the minterm ABC, a single product term, as a single 1 in a map that is otherwise 0s. We have not shown the 0s in our Karnaugh maps up to this point, as it is customary to omit them unless specifically needed. Another minterm A'BC' is shown in Figure 4.1 right. The point to review is that the address of the cell corresponds directly to the minterm being mapped. That is, the cell 111 corresponds to the minterm ABC above left. The minterm A'BC' corresponds directly to the cell 010. A Boolean expression or map may have multiple minterms.

Referring to the Figure 4.1, let's summarize the procedure for placing a minterm in a K-map:

- Identify the minterm (product term) to be mapped.
- Write the corresponding binary numeric value.
- Use binary value as an address to place a 1 in the K-map
- Repeat steps for other minterms (P-terms within a Sum-Of-Products).

Notes

Figure 4.2: Minterms



A Boolean expression will more often than not consist of multiple minterms corresponding to multiple cells in a Karnaugh map as shown above. The multiple minterms in this map are the individual minterms which we examined in the previous Figure 4.2. The point we review for reference is that the 1s come out of the K-map as a binary cell address which converts **directly** to one or more product terms. By directly we mean that a 0 corresponds to a complemented variable, and a 1 corresponds to a true variable. Example: 010 converts **directly** to A'BC'. There was no reduction in this example. Though, we do have a Sum-Of-Products result from the minterms.

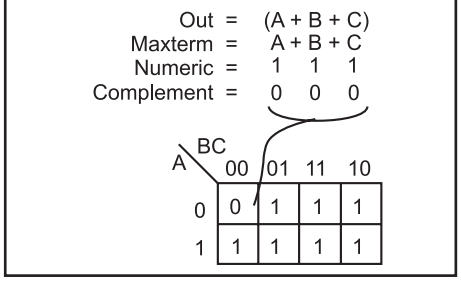
Referring to the Figure 4.2, Let's summarize the procedure for writing the Sum-Of-Products reduced Boolean equation from a K-map:

- Form largest groups of 1s possible covering all minterms. Groups must be a power of 2.
- Write binary numeric value for groups.
- Convert binary value to a product term.
- Repeat steps for other groups. Each group yields a p-terms within a Sum-Of-Products.

Nothing new so far, a formal procedure has been written down for dealing with minterms. This serves as a pattern for dealing with maxterms.

Next we attack the Boolean function which is 0 for a single cell and 1s for all others.

Figure 4.3: Maxterm in the K-map



A *maxterm* is a Boolean expression resulting in a **0** for the output of a single cell expression, and 1s for all other cells in the Karnaugh map, or truth table. The illustration above left shows the maxterm($A + B + C$), a single sum term, as a single 0 in a map that is otherwise 1s. If a maxterm has a single 0 and the remaining cells as 1s, it would appear to cover a maximum area of 1s.

There are some differences now that we are dealing with something new, maxterms. The maxterm is a 0, not a 1 in the Karnaugh map. A maxterm is a sum term, ($A + B + C$) in our example, not a product term.

It also looks strange that ($A + B + C$) is mapped into the cell 000. For the equation Out=($A + B + C$) = 0, all three variables (A, B, C) must individually be equal to 0. Only (0 + 0 + 0) = 0 will equal 0. Thus we place our sole 0 for minterm ($A + B + C$) in cell A, B, C = 000 in the K-map, where the inputs are all 0. This is the only case which will give us a 0 for our maxterm.

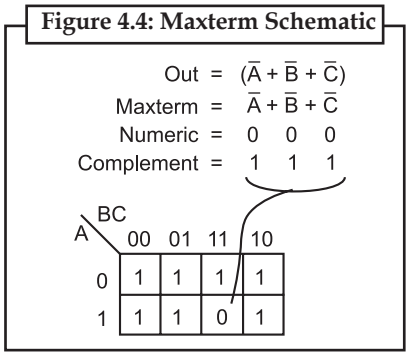
Notes

All other cells contain 1s because any input values other than ((0,0,0) for $(A + B + C)$ yields 1s upon evaluation.

Referring to the Figure 4.3, the procedure for placing a maxterm in the K-map is:

- Identify the Sum term to be mapped
- Write corresponding binary numeric value
- Form the complement
- Use the complement as an address to place a 0 in the K-map
- Repeat for other maxterms (Sum terms within Product-of-Sums expression).

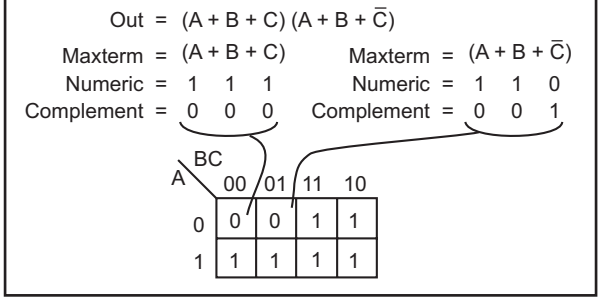
Figure 4.4: Maxterm Schematic



Another maxterm A′ + B′ + C′ is shown. Numeric 000 corresponds to A′ + B′ + C′. The complement is 111. Place a 0 for maxterm (A′ + B′ + C′) in this cell (1,1,1) of the K-map as shown above.

Why should (A′ + B′ + C′) cause a 0 to be in cell 111? When A′ + B′ + C′ is (1′ + 1′ + 1′), all 1s in, which is (0 + 0 + 0) after taking complements, we have the only condition that will give us a 0. All the 1s are complemented to all 0s, which is 0 when ORed.

Figure 4.5: Maxterms Schematic



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

Table 4.1: Two-variable Expression of Minterms and Maxterms

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

Notes

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

Table 4.2: Three-variable Expression of Minterms and Maxterms

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

This allows us to represent expressions in either Sum-of-Products or Product-of-Sums forms.

Did u know?

A Boolean expression only composed a combination of the Boolean constants true or false, Boolean-typed variables, Boolean-valued operators, and Boolean-valued functions.

4.2 Sum-of-Products and Product-of-Sums Forms of Logic Expressions

Logical functions are expressed in terms of logical variables. The values taken by logical variables and logical functions are in binary form. Any logical function may he expressed in the following two forms:

1. Sum-of-products (SOP) form
2. Product-of-sums (POS) form

4.2.1 Sum-of-Products Form

A sum-of-products expression consists of product terms logically added. A product term is a logical product of many variables. The variables used may or may not be complemented. Following are the examples of sum of products.

(*i*) $AB + \bar{A}B + A\bar{B}$

(*ii*) $AB + ABC + B\bar{C}$

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

(*iv*) $ABC + \bar{A}B + A\bar{B}C + \bar{A}B\bar{C}$

Task Prepare a truth table for minterms and maxterms.

4.2.2 Product-of-Sums Form

A product-of-sums expression consists of sum terms logically multiplied. A sum term is the logical addition of several variables. The variables may or may not be complemented.

Following are the examples of product of sums:

Notes

(*i*) $(A + B)(\overline{A} + B)$

(*ii*) $A(\overline{B} + \overline{C})(B + C)$

(*iii*) $(A + \overline{B})(A + B + C)(B + C)$

(*iv*) $(A + B + C)(\overline{A} + B + C)(A + \overline{B} + \overline{C})$

Sum-Of-Products expressions are easy to generate from truth tables. All we have to do is examine the truth table for any rows where the output is "high" (1), and write a Boolean product term that would equal a value of 1 given those input conditions. For instance, in the fourth row down in the truth table for our two-out-of-three logic system, where A = 0, B = 1, and C = 1, the product term would be A'BC, since that term would have a value of 1 if and only if A = 0, B = 1, and C = 1:

Figure 4.6: Sum-of-Products

sensor inputs

A	B	C	Output	
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	$\overline{A}BC = 1$
1	0	0	0	
1	0	1	1	
1	1	0	1	
1	1	1	1	

Three other rows of the truth table have an output value of 1, so those rows also need Boolean product expressions to represent them:

Figure 4.7: Boolean Product Expressions

sensor inputs

A	B	C	Output	
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	$\overline{A}BC = 1$
1	0	0	0	
1	0	1	1	$A\overline{B}C = 1$
1	1	0	1	$AB\overline{C} = 1$
1	1	1	1	$ABC = 1$

Finally, we join these four Boolean product expressions together by addition, to create a single Boolean expression describing the truth table as a whole:

Notes

Figure 4.8: Single Boolean

sensor inputs

A	B	C	Output	
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	$\bar{A}BC = 1$
1	0	0	0	
1	0	1	1	$A\bar{B}C = 1$
1	1	0	1	$AB\bar{C} = 1$
1	1	1	1	$ABC = 1$

Output $= \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$

Example:

Simplify the Product-Of-Sums Boolean expression below, providing a result in POS form.

$$\text{Out} = (A + B + C + \bar{D})(A + B + \bar{C} + D)(A + \bar{B} + C + \bar{D})(A + \bar{B} + \bar{C} + D)$$
$$(\bar{A} + \bar{B} + \bar{C} + D)(\bar{A} + B + C + \bar{D})(\bar{A} + B + \bar{C} + D)$$

Solution:

Transfer the seven maxterms to the map below as 0s. Be sure to complement the input variables in finding the proper cell location.

Figure 4.9: Maxterms

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

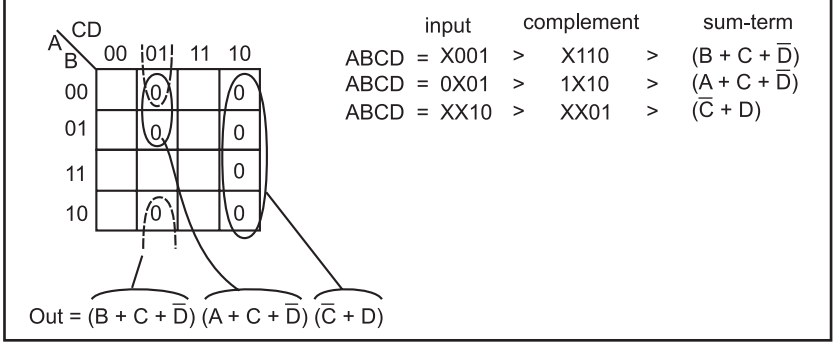
A \ B \ CD	00	01	11	10
00		0		0
01		0		0
11				0
10		0		0

We map the 0s as they appear left to right top to bottom on the map above. We locate the last three maxterms with leader lines..

Once the cells are in place above, form groups of cells as shown below. Larger groups will give a sum-term with fewer inputs. Fewer groups will yield fewer sum-terms in the result.

Notes

Figure 4.10: Three Sum-terms in our POS



We have three groups, so we expect to have three sum-terms in our POS result. The group of 4-cells yields a 2-variable sum-term. The two groups of 2-cells give us two 3-variable sum-terms. Details are shown for how we arrived at the sum-terms above. For a group, write the binary group input address, then complement it, converting that to the Boolean sum-term. The final result is product of the three sums.

Example:

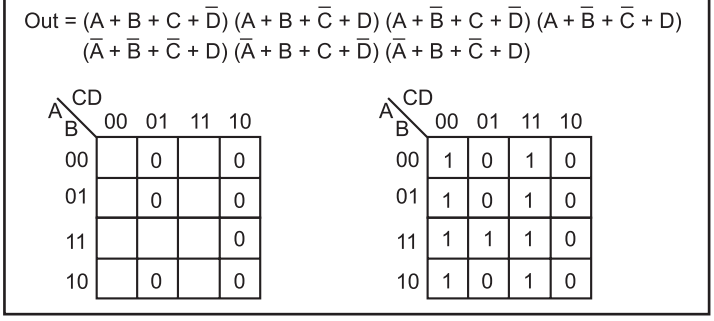
Simplify the Product-Of-Sums Boolean expression below, providing a result in SOP form:

$$\text{Out} = (A+B+C+\bar{D})(A+B+\bar{C}+D)(A+\bar{B}+C+\bar{D})(A+\bar{B}+\bar{C}+D)$$
$$(\bar{A}+\bar{B}+\bar{C}+D)(\bar{A}+B+C+\bar{D})(\bar{A}+B+\bar{C}+D)$$

Solution:

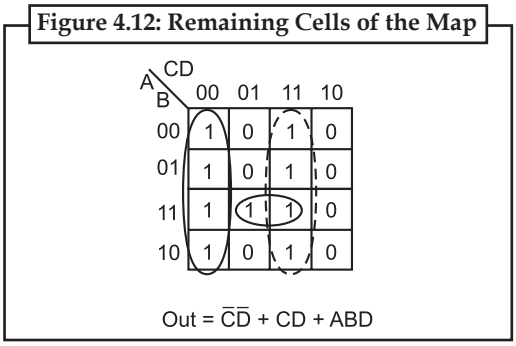
This looks like a repeat of the last problem. It is except that we ask for a Sum-Of-Products Solution instead of the Product-Of-Sums which we just finished. Map the maxterm 0s from the Product-Of-Sums given as in the previous problem, below left:

Figure 4.11: Sum-of-Products



Then fill in the implied 1s in the remaining cells of the map right.

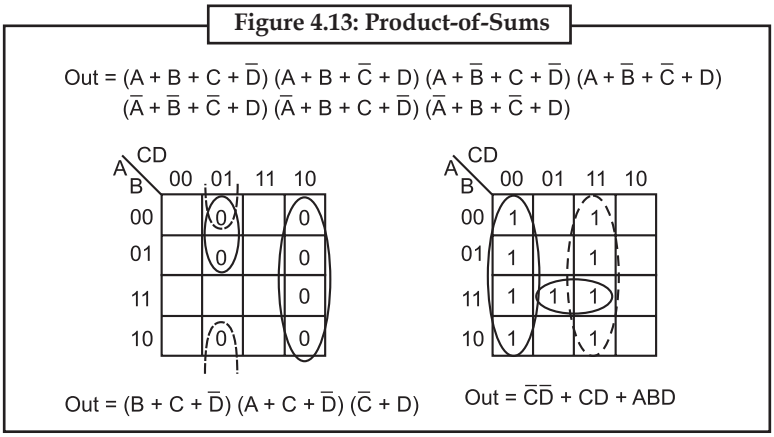
Figure 4.12: Remaining Cells of the Map



Form groups of 1s to cover all 1s.

Notes

Figure 4.13: Product-of-Sums



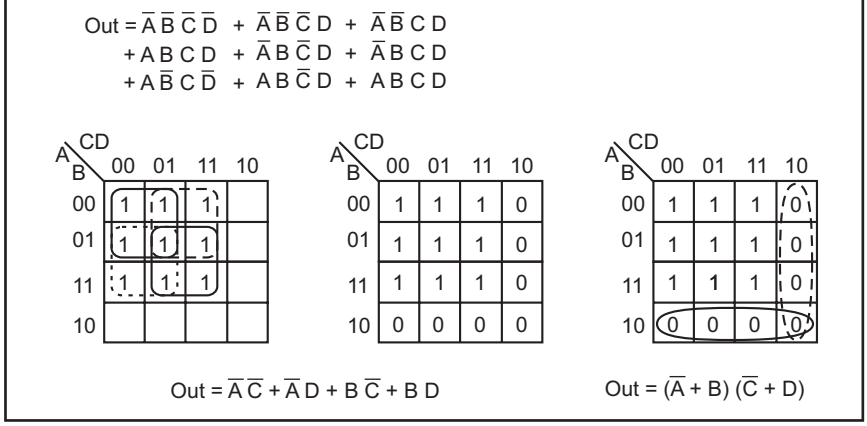
We show both the Product-Of-Sums solution, from the previous example, and the Sum-Of-Products solution from the current problem for comparison. Which is the simpler solution? The POS uses 3-OR gates and 1-AND gate, while the SOP uses 3-AND gates and 1-OR gate. Both use four gates each. Taking a closer look, we count the number of gate inputs. The POS uses 8-inputs; the SOP uses 7-inputs. By the definition of minimal cost solution, the SOP solution is simpler. This is an example of a technically correct answer that is of little use in the real world.

Task Draw the circuit diagram of Sum-Of-Products.

Example:

Let us revisit a previous problem involving an SOP minimization. Produce a Product-Of-Sums solution. Compare the POS solution to the previous SOP.

Figure 4.14: SOP Minimization



Solution:

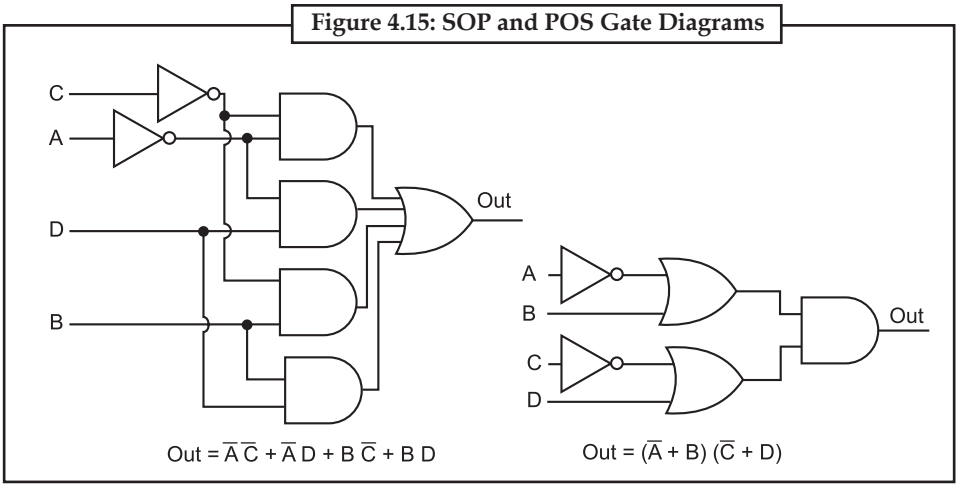
Above left we have the original problem starting with a 9-minterm Boolean unsimplified expression. Reviewing, we formed four groups of 4-cells to yield a 4-product-term SOP result, lower left.

In the middle Figure 4.14, we fill in the empty spaces with the implied 0s. The 0s form two groups of 4-cells. The solid blue group is (A'+B), the dashed red group is (C'+D). This yields two sum-terms in the Product-Of-Sums result, above right Out = (A'+B)(C'+D).

Notes

Comparing the previous SOP simplification, left, to the POS simplification, right, shows that the POS is the least cost solution. The SOP uses 5-gates total, the POS uses only 3-gates. This POS solution even looks attractive when using TTL logic due to simplicity of the result. We can find AND gates and an OR gate with 2-inputs.

Figure 4.15: SOP and POS Gate Diagrams



The SOP and POS gate diagrams are shown above for our comparison problem.

Did u know?

A tautology is a propositional formula that assigned truth value 1 by every truth assignment of its propositional variables to an arbitrary Boolean algebra (or, equivalently, every truth assignment to the two elements Boolean algebra).

Self Assessment

Choose the correct answer:

1. A maxterm is the product of N distinct literals where each literal occurs exactly once.
   (*a*) True (*b*) False
2. The Sum- of-Products form represents an expression as a sum of minterms.
   (*a*) True (*b*) False
3. ..... expressions are easy to generate from truth tables.
   (*a*) TTL (*b*) NAND
   (*c*) SOP (*d*) POS
4. A ..... is a logical product of many variables.
   (*a*) Maxterm (*b*) Minterms
   (*c*) Product term (*d*) Sum-terms
5. The examples of sum-of-products .....
   (*a*) $(A+B)\,(\overline{A}+B)$ (*b*) $A(\overline{B}+\overline{C})(B+C)$
   (*c*) $(A+B)(\overline{A}+B)$ (*d*) $AB+\overline{A}B+A\overline{B}$
6. The examples of product-of-sums .....
   (*a*) $AB+ABC+B\overline{C}$ (*b*) $A+A\overline{B}+\overline{B}C$
   (*c*) $(A+B)(\overline{A}+B)$ (*d*) $AB+\overline{A}B+A\overline{B}$
7. A product of sums expression consists of sum terms logically multiplied.
   (*a*) True (*b*) False

Notes

4.3 Karnaugh Maps

The Karnaugh map is a graphical device used to simplify a logic equation or to convert a truth table to its corresponding logic circuit in a simple, orderly process. Although a Karnaugh map (henceforth abbreviated *K - map*) can be used for problems involving any number of input variables, its practical usefulness is limited to six variables. The following discussion will be limited to problems with up to four inputs, since even five- and six-input problems are too involved and are best done by a computer program.

What Does a Karnaugh Map Look Like?

A Karnaugh map is a grid-like representation of a truth table. It is really just another way of presenting a truth table, but the mode of presentation gives more insight. A Karnaugh map has zero and one entries at different positions. Each position in a grid corresponds to a truth table entry. Here's an example taken from the voting circuit presented on Minterms. The truth table is shown first. The Karnaugh Map for this truth table is shown after the truth table.

Table 4.3: Karnaugh Map for Truth Table

A	B	C	V
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

		BC			
		00	01	11	10
A	0	0	0	1	0
	1	0	1	1	1

Did u know?

A Boolean algebra is called representable when it is isomorphic to a concrete Boolean algebra.

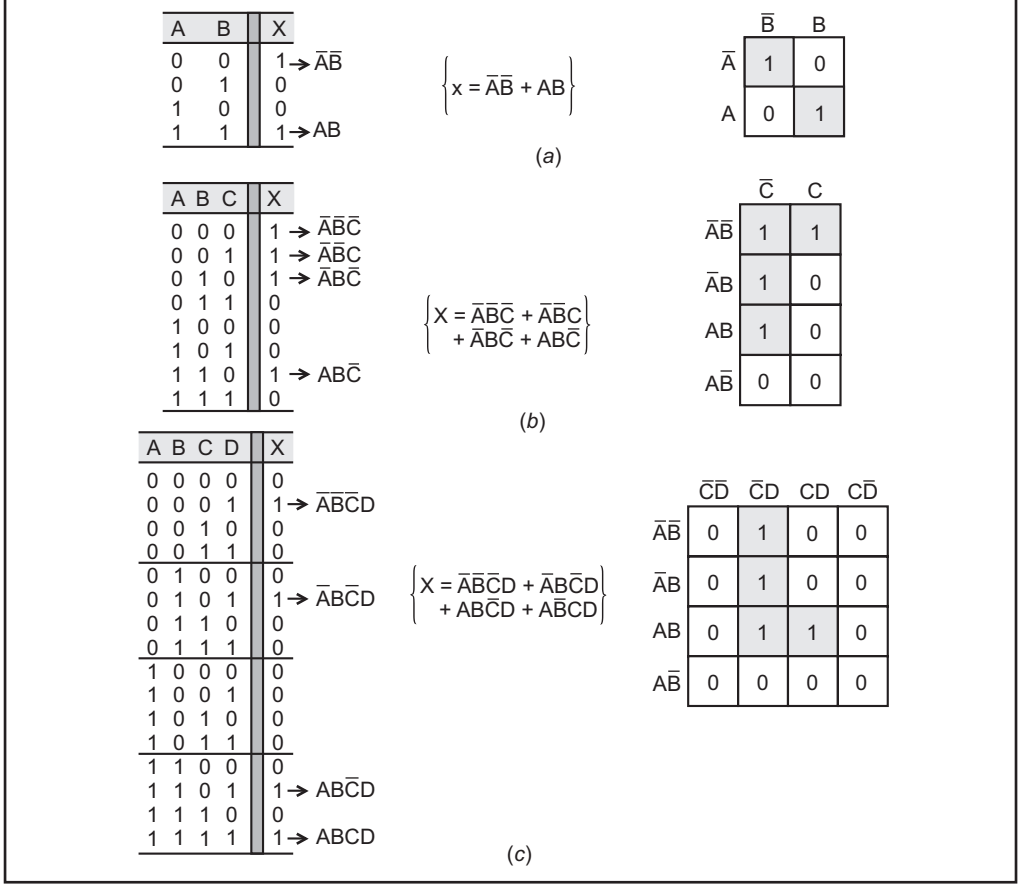
4.3.1 Karnaugh Map Format

The K-map, like a truth table, is a means for showing the relationship between logic inputs and the desired output. Figure 4.16 shows three examples of K-maps for two, three, and four variables, together with the corresponding truth tables. These examples illustrate the following important points:

1. The truth table gives the value of output X for each combination of input values. The K-map gives the same information in a different format. Each case in the truth table corresponds to a square in the K-map. For example, in Figure 4.16 (*a*), the $A = 0$, $B = 0$ condition in the truth table corresponds to the $\overline{A}\overline{B}$ square in the K map. Since the truth table shows $X = 1$ for this case, a 1 is placed in the $\overline{A}\overline{B}$ square in the K-map. Similarly, the $A = 1$, $B = 1$ condition in the truth table corresponds to the AB square of the K-map. Since $X = 1$ for this case, a 1 is placed in the AB square. All other squares are filled with Os. This same idea is used in the three- and four-variable maps shown in the Figure 4.16.

Notes

Figure 4.16: Karnaugh Maps and Truth Tables for (*a*) Two, (*b*) Three, and (*c*) Four Variables



2. The K-map squares are labelled so that horizontally adjacent squares differ only in one variable. For example, the upper left-hand square in the four-variable map is $\bar{A}\bar{B}\bar{C}\bar{D}$, while the square immediately to its right is $\bar{A}\bar{B}\bar{C}D$ (only the D variable is different). Similarly, vertically adjacent squares differ only in one variable. For example, the upper left-hand square is $\bar{A}\bar{B}\bar{C}D$ while the square directly below it is $\bar{A}B\bar{C}\bar{D}$ (only the *B* variable is different).

   Note that each square in the top row is considered to be adjacent to a corresponding square in the bottom row. For example, the $\bar{A}\bar{B}CD$ square in the top row is adjacent to the $A\bar{B}CD$ square in the bottom row, since they differ only in the A variable. The top of the map as being wrapped around to touch the bottom of the map. Similarly, squares in the leftmost column are adjacent to corresponding squares in the rightmost column.

3. In order for vertically and horizontally adjacent squares to differ in only one variable, the top-to-bottom labelling must be done in the order shown- $\bar{A}\bar{B}, \bar{A}B, AB, A\bar{B}$. The same is true of the left-to-right labeling.

4. Once a K-map has been filled with 0s and 1s, the sum-of-products expression for the output *X* can be obtained by 0Ring together those squares that contain a 1. In the three-variable map of Figure 4.17(*b*), the $\bar{A}\bar{B}\bar{C}, \bar{A}\bar{B}C$ $\bar{A}B\bar{C}$, and $AB\bar{C}$ squares contain a 1, so that $X = \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C + \bar{A}B\bar{C} + AB\bar{C}$.

   The magnificence of the Karnaugh map is that it has been cleverly designed so that any two adjacent cells in the map differ by a change in one variable. It is always a change of one variable any time you cross a horizontal or vertical cell boundaries. (It is not fair to go through the corners!)

Notes

4.3.2 Looping

The expression for output X can be simplified by properly combining those squares in the K-map which contain 1s. The process for combining these 1s is called looping.

Looping Groups of Two (Pairs)

Figure 4.18(*a*) is the K map for a particular three-variable truth table. This map contains a pair of 1s that are vertically adjacent to each other; the first represents $\bar{A}B\bar{C}$ and the second represents $AB\bar{C}$. Note that in these two terms only the *A* variable appears in both normal and complemented form (*B* and $\bar{C}$ remain unchanged). These two terms can be looped (combined) to give a resultant that eliminates the *A* variable since it appears in both uncomplemented and complemented forms. This is easily proved as follows:

$$X = \bar{A}B\bar{C} + AB\bar{C}$$
$$= B\bar{C}(\bar{A} + A)$$
$$= B\bar{C}(1) = B\bar{C}$$

This same principle holds true for any pair of vertically or horizontally adjacent 1s. Figure 4.18 (*b*) shows an example of two horizontally adjacent 1s. These two can be looped and the *C* variable eliminated since it appears in both its uncomplemented and complemented forms to give a resultant of $X = \bar{A}B$.

Another example is shown in Figure 4.18 (*c*). In a K-map the top row and bottom row of squares are considered to be adjacent. Thus, the two 1s in this map can be looped to provide a resultant of $\bar{A}\bar{B}\bar{C} + A\bar{B}\bar{C} = B\bar{C}$.

Figure 4.17: Examples of Looping Pairs of Adjacent 1s

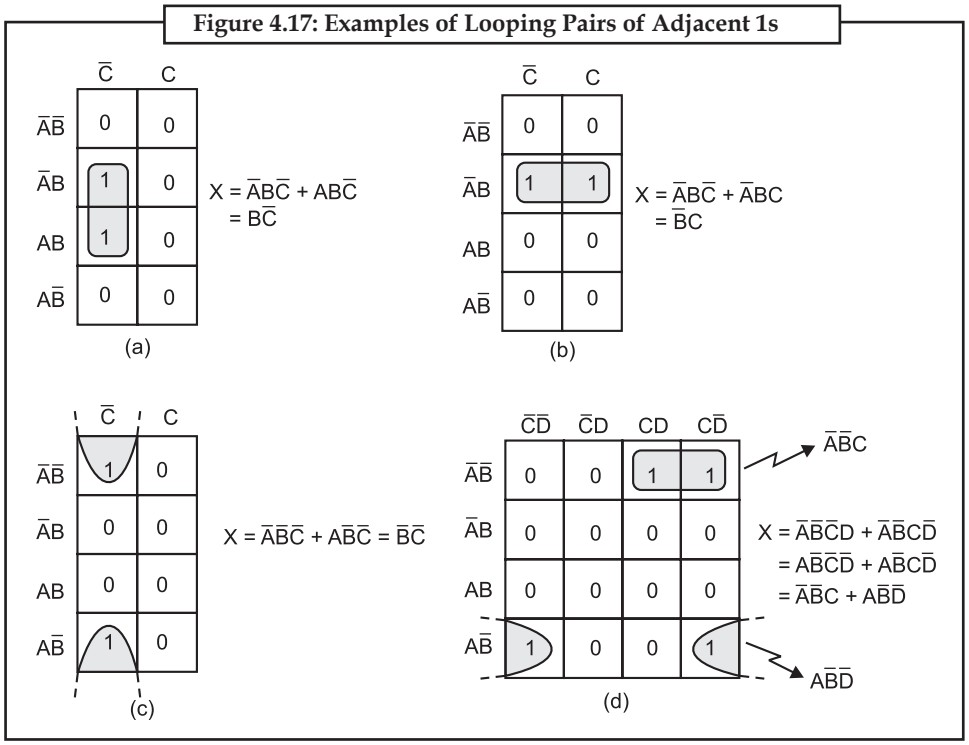


Figure 4.18 (*d*) shows a K-map that has two pairs of 1s which can be looped. The two 1s in the top row are horizontally adjacent. The two 1s in the bottom row are also adjacent, since in a K-map the leftmost column and rightmost column of sequence are considered to be adjacent. When the top pair of ls is looped, the *D* able is eliminated (since it appears as both *D* and $\bar{D}$) to give the term $\bar{A}B\bar{C}$. ping the bottom pair eliminates the *C* variable to give the term $A\bar{B}\bar{D}$. These terms are ORed to give the final result for *X*.

Notes

To summarize:

Looping a pair of adjacent 1s in a K-map eliminates the variable that appears in complemented and uncomplemented form.

Looping Groups of Four (Quads)

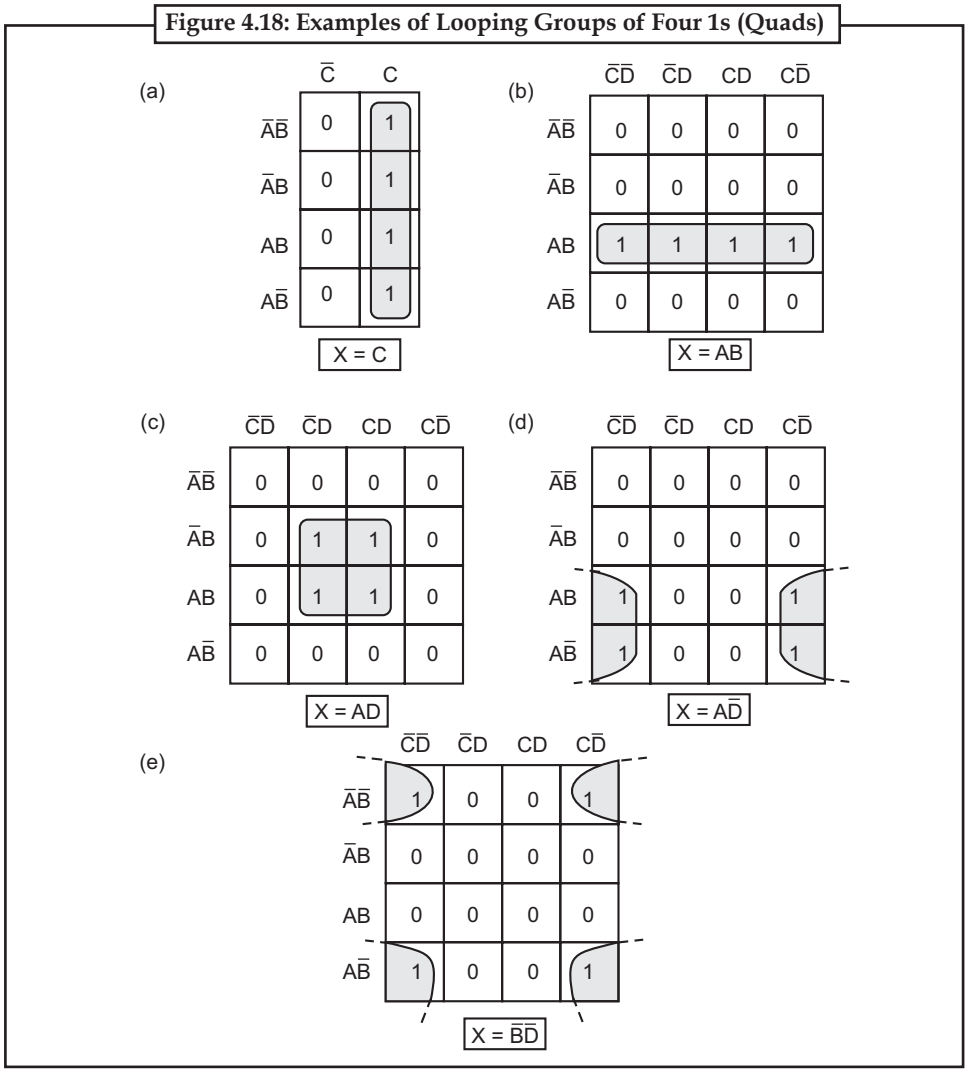
A K-map may contain a group four 1s that are adjacent to each other. This group is called a *quad*. Figure 4.19 shows several examples of quads. In part (*a*) the four 1s are vertically adjacent and in part (*b*) they are horizontally adjacent. The K-map in Figure 4.19(*c*) contains four 1s in a square, and they are considered adjacent to each other. The four 1s in Figure 4.19(*d*) are also adjacent, as are those in Figure 4.19 (*e*) because, as pointed out earlier, the top and bottom rows are considered to be adjacent to each other as are the leftmost and rightmost columns. ***Pair*** two horizontally or vertically adjacent 1s on a Karnaugh map.

Caution

Resistive divider must be constructed in such a way that a 1 in the 21 bit position will cause a change of +7 *1/7=+ 2 V in the analog output voltage.

When a quad is looped, the resultant term will contain only the variables–do not change form for all the squares in the quad. For example, in Figure 4.19 (the four squares which contain a 1 are $\bar{A}\bar{B}C$, $\bar{A}BC$, ABC, and $A\bar{B}C$. Examination of these terms reveals that only the variable C remains unchanged (both A and B) appear in complemented and uncomplemented form).

Figure 4.18: Examples of Looping Groups of Four 1s (Quads)



Notes

Thus, the resultant expression for X is simply $X = C$. This can be proved as follows:

$$\begin{aligned} X &= \bar{A}\bar{B}C + \bar{A}BC + ABC + A\bar{B}C \\ &= \bar{A}C(\bar{B} + B) + AC(B + \bar{B}) \\ &= \bar{A}C + AC \\ &= C(\bar{A} + A) = C \end{aligned}$$

As another example, consider Figure 4.19(d), where the four squares containing 1s are $AB\bar{C}\bar{D}$ $A\bar{B}\bar{C}\bar{D}$, $ABC\bar{D}$, and $A\bar{B}C\bar{D}$. Examination of these terms indicates that only the variables A and $\bar{D}$ remain unchanged, so that the simplified expression)r X is

$$X = A\bar{D}$$

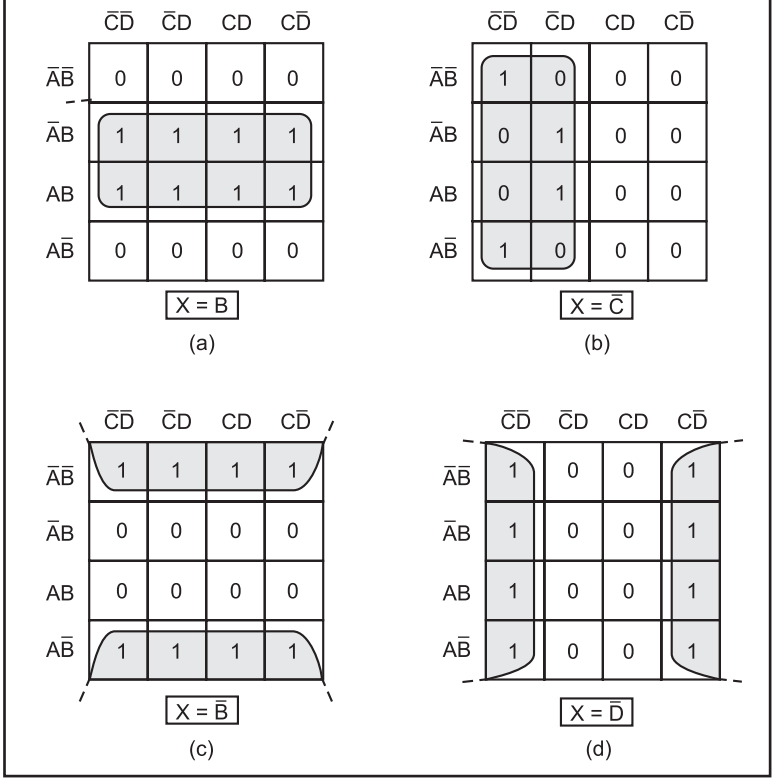
To summarize:

Looping a quad of 1s eliminates the two variables that appear in both complemented and uncomplemented form.

Looping Groups of Eight (Octets)

A group of eight 1s that are adjacent to one another is called an octet. Several examples of octets are shown in Figure 4.20. When an octet is looped in a four-variable map, three of the four

Figure 4.19: Examples of Looping Groups of Eight 1s (Octets)



variables are eliminated because only one variable remains unchanged. For example, examination of the eight looped squares in Figure 4.19 (a) shows that only the variable B is in the same form for all eight squares; the other variables appear in complemented and uncomplemented form. Thus, for this map, $X = B$. The reader can verify the results for the other examples in Figure 4.19.

Notes

To summarize:

Looping an octet of 1s eliminates the three variables that appear in both complemented and uncomplemented form.

Complete Simplification Process

We have seen how looping of pairs, quads, and octets on a K-map can be used to obtain a simplified expression. We can summarize the rule for loops of any size: When a variable appears in both complemented and uncomplemented form within a loop, that variable is eliminated from the expression. Variables that are the same for all squares of the loop must appear in the final expression.

It should be clear that a larger loop of 1s eliminates more variables. To be exact, a loop of two eliminates one variable, a loop of four eliminates two, and a loop of eight eliminates three. This principle will now be used to obtain a simplified logic expression from a K-map that contains any combination of 1s and 0s.

The procedure will first be outlined and then applied to several examples. The steps below are followed in using the K-map method for simplifying a Boolean expression:

1. Construct the K-map and place 1s in those squares corresponding to the 1s in the truth table. Place 0s in the other squares.
2. Examine the map for adjacent 1s and loop those 1s which are not adjacent to any other 1s. These are called *isolated 1s*.
3. Next, look for those 1s which are adjacent to only one other 1. Loop *any* pair containing such a 1.
4. Loop any octet even if it contains some 1s that have already been looped.
5. Loop any quad that contains one or more 1s which have not already been looped, *making sure to use the minimum number of loops*.
6. Loop any pair necessary to include any 1s that have not yet been looped, *making sure to use the minimum number of loops*.
7. Form the OR sum of all the terms generated by each loop.


Caution

Be careful as the analog output signal must be held between sampling periods, and the outputs must therefore be equipped with sample-and-hold amplifiers.

Example:

Figure 4.20 (*a*) shows the K-map for a four-variable problem. We will assume that the map was obtained from the problem truth table (Step 1). The squares are numbered for convenience in identifying each loop.

Step 2. Square 4 is the only square containing a 1 that is not adjacent to any other 1. It is looped and is referred to as loop 4.

Step 3. Square 15 is adjacent only to square 11. This pair is looped and referred to as loop 11, 15.

Step 4. There are no octets.

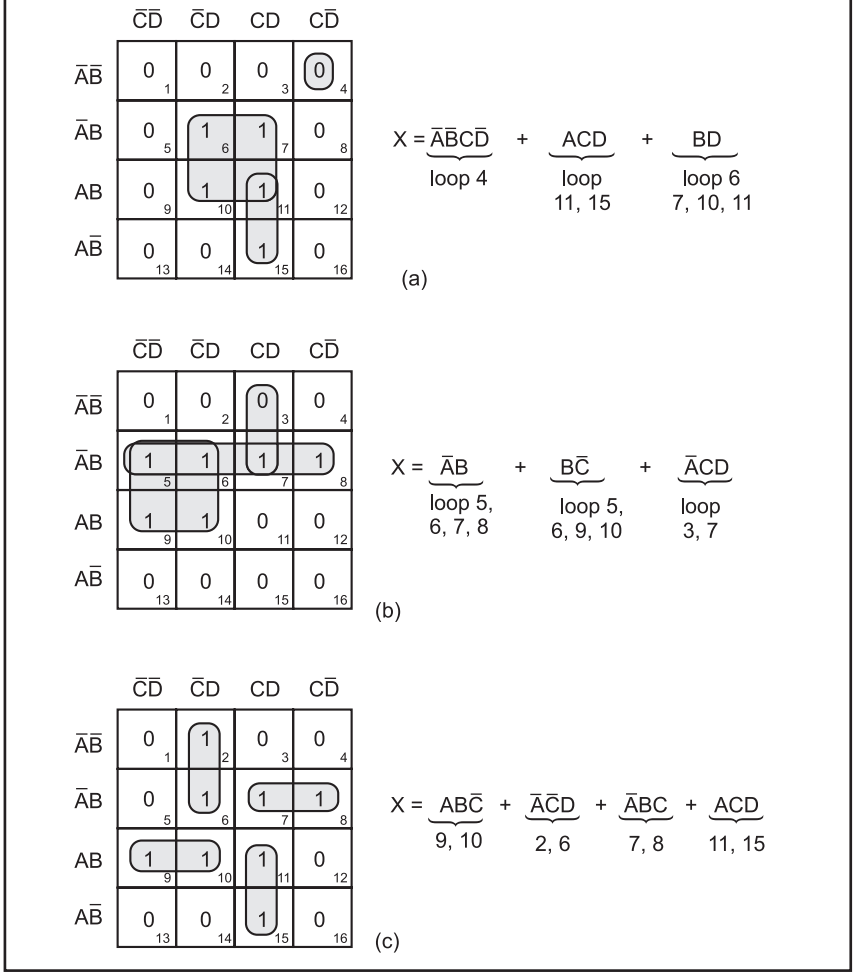
Notes

Step 5. Squares 6, 7, 10, and 11 form a quad. This quad is looped (loop 6, 7, 10, 11). Note that square 11 is used again, even though it was part of loop 11, 15.

Step 6. All 1s have already been looped.

Step 7. Each loop generates a term in the expression for *X*. Loop 4 is simply $\bar{A}\bar{B}C\bar{D}$. Loop 11, 15 is *ACD* (the *B* variable is eliminated). Loop 6, 7, 10, 11 is *BD* (*A* and *C* are eliminated).

Figure 4.20: The K-map



Example:

Consider the K map in Figure 4.21 (*b*). Once again we can assume that Step 1 has already been performed.

Step 2. There are no isolated 1s.

Step 3. The 1 in square 3 is adjacent *only* to the 1 in square 7. Looping d pair (loop 3, 7) produces the term $\bar{A}CD$.

Step 4. There are no octets.

Step 5. There are two quads. Squares 5, 6, 7, and 8 form one quad. Looping this quad produces the term $\bar{A}B$. The second quad is made up of squares 6, 9, and 10.

Notes

This quad is looped because it contains two squares that have not been looped previously. Looping this quad produces $B\bar{C}$.

Step 6. All 1s have already been looped.

Step 7. The terms generated by the three loops are ORed together to obtain the expression for *X*.

Example:

Consider the K-map in Figure 4.21 (*c*).

Step 2. There are no isolated 1s.

Step 3. The 1 in square 2 is adjacent only to the 1 in square 6. This pair looped to produce $\bar{A}\bar{C}D$. Similarly, square 9 is adjacent only to square 10.

Looping this pair produces $\bar{A}BC$. Likewise, loop 7, 8 and loop 11, 15 product the terms $\bar{A}BC$ and ACD, respectively.

Step 4. There are no octets.

Step 5. There is one quad formed by squares 6, 7, 10, and 11. This quad, however, is not looped, because all the Is in the quad have been included other loops.

Step 6. All 1s have already been looped.

Step 7. The expression for *X* is shown in the Figure 4.21.

Example:

Consider the K map in Figure 4.22(*a*).

Step 2. There are no isolated 1s.

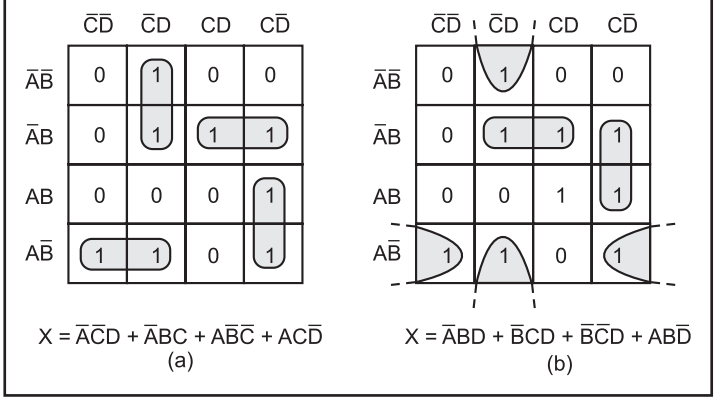
Step 3. There are no 1s that are adjacent to only one other 1.

Step 4. There are no octets.

Step 5. There are no quads.

Steps 6 and 7. There are many possible pairs. The looping must use the minimum number of loops to account for all the 1s. For this map there are two possible loopings, which require only four looped pairs. Figure 4.22 shows one solution and its resultant expression. Figure 4.22(b) shows the other. Note that both expressions are of the same complexity, and so neither is better than the other.

Figure 4.21: The Same K-Map with Two Equally Solutions



Notes

Example:

Use the K-map to simplify the expression $y = \bar{A}\bar{B}\bar{C} + \bar{B}C + \bar{A}B$.

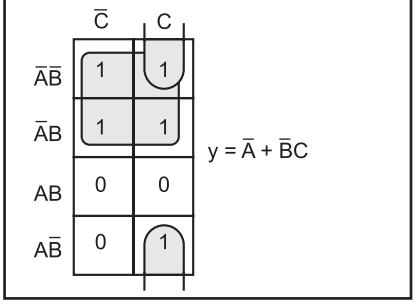
Solution:

In this problem we are not given the truth table from which to fill in the K-map instead, we must fill in the K-map by taking each of the product terms in the expression and placing 1s in the corresponding squares.

The first term, $\bar{A}\bar{B}\bar{C}$ tells us to enter a 1 in the $\bar{A}\bar{B}\bar{C}$ square of the map (see 4.23). The second term, $\bar{B}C$, tells us to enter a 1 in each square that contains a $\bar{B}C$, in its label. In Figure 4.23 this would be the $A\bar{B}C$ and $\bar{A}\bar{B}C$ squares. Likewise, the $\bar{A}B$ term tells us to place a 1 in the $\bar{A}BC$ and $\bar{A}B\bar{C}$ squares. All other squares will be filled with 0s.

Now the K-map can be looped for simplification. The result is $y = (\bar{A} + \bar{B}C,)$ as shown in the Figure 4.23.

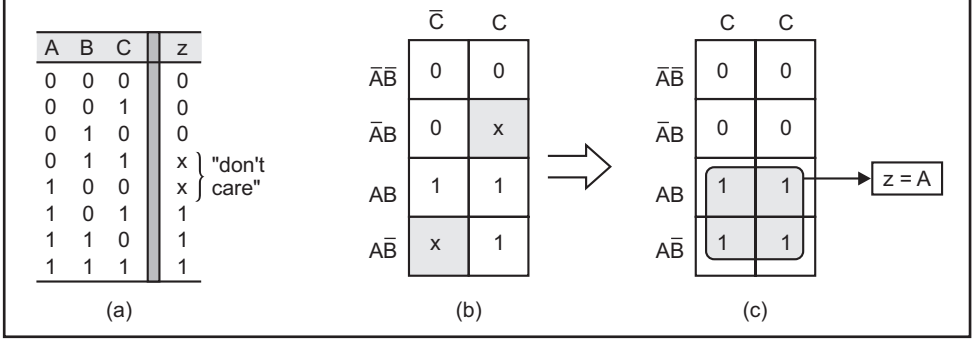
Figure 4.22: Example K-map



"Don't Care" Conditions

Some logic circuits can be designed so that there are certain input conditions for which there are no specified output levels, usually because these input conditions will never occur. In other words, there will be certain combinations of input levels where we "don't care" whether the output is HIGH or LOW. This is illustrated in the truth table of Figure 4.24 (*a*).

Figure 4.23: "Don't Care" Conditions should be Changed to 0 or 1 to Produce K-map Looping that Yields the Simplest Expression



Here the output z is not specified as either 0 or 1 for the conditions *A*,*B*,*C* = 1, 0, 0 and *A*, *B*, *C* = 0, 1, 1. Instead, an *x* is shown for these conditions. T represents the "don't care" condition. A "don't care" condition can come about several reasons, the most common being that in some situations certain input combinations can never occur, and so there is no specified output for these conditions.

Notes

A circuit designer is free to make the output for any "don't care" condition either a 0 or a 1 in order to produce the simplest output expression. For example the K map for this truth table is shown in Figure 4.24 (*b*) with an *x* placed in the $A\bar{B}\bar{C}$ and $\bar{A}BC$ squares. The designer here would be wise to change the *x* in the $A\bar{B}\bar{C}$ square to a 1 and the *x* in the $\bar{A}BC$ square to a 0, since this would produce quad that can be looped to produce $z = A$, as shown in Figure 4.24 (*c*).

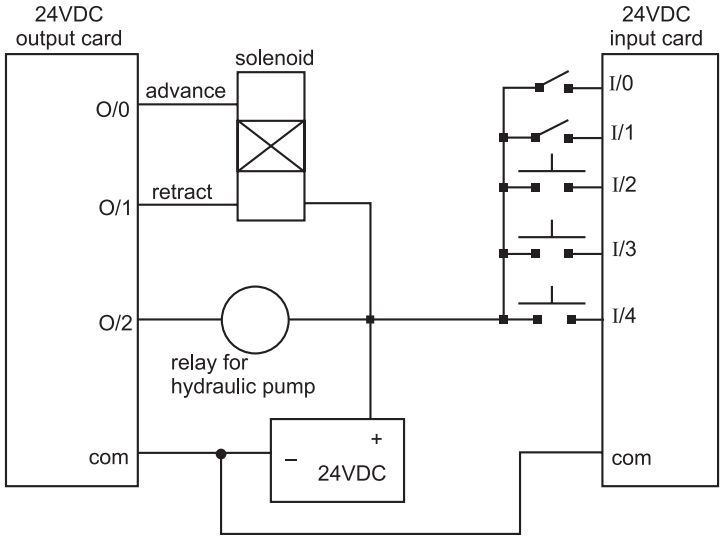
Whenever "don't care" conditions occur, we have to decide which one change to 0 and which to 1 to produce the best K-map looping (i.e. the simplest expression). This decision is not always an easy one.

Case Study

Press Wiring

An electrical layout is needed for a hydraulic press. The press uses a 24Vdc double actuated solenoid valve to advance and retract the press. This device has a single common and two input wires. Putting 24Vdc on one wire will cause the press to advance, putting 24Vdc on the second wire will cause it to retract. The press is driven by a large hydraulic pump that requires 220Vac rated at 20A, this should be running as long as the press is on. The press is outfitted with three push buttons, one is a NO stop button, the other is a NO manual retract button, and the third is a NO start automatic cycle button. There are limit switches at the top and bottom of the press travels that must also be connected.

SOLUTION



The input and output cards were both selected to be 24Vdc so that they may share a single 24Vdc power supply. In this case the solenoid valve was wired directly to the output card, while the hydraulic pump was connected indirectly using a relay (only the coil is shown for simplicity). This decision was primarily made because the hydraulic pump requires more current than any PLC can handle, but a relay would be relatively easy to purchase and install for that load. All of the input switches are connected to the same supply and to the inputs.

Questions:

1. Explain the need of hydraulic press.
2. What are limit switches at the top and bottom of the press?

Notes

Self Assessment

Choose the correct answer:

8. Karnaugh map is used for the purpose of
   (*a*) Reducing the electronic circuits used.
   (*b*) To map the given Boolean logic function.
   (*c*) To minimize the terms in Boolean expression.
   (*d*) To maximize the terms of a given Boolean expression.
9. The Karnaugh map is a graphical device used to simplify a .....
   (*a*) linear equation (*b*) logic equation
   (*c*) logix equation (*d*) None of these
10. A Karnaugh map has ..... and ..... entries at different positions.
   (*a*) zero, one (*b*) one, two
   (*c*) two, three (*d*) None of these

4.4 Summary

- Minterm is a Boolean expression resulting in 1 for the output of a single cell, and 0s for all other cells in a Karnaugh map, or truth table.
- Maxterm is a Boolean expression resulting in a 0 for the output of a single cell expression, and 1s for all other cells in the Karnaugh map, or truth table.
- Product of sums expression consists of sum terms logically multiplied. A sum term is the logical addition of several variables. The variables may or may not be complemented.
- Karnaugh map is a graphical device used to simplify a logic equation or to convert a truth table to its corresponding logic circuit in a simple, orderly process.
- K-map has been filled with 0s and 1s, the sum-of-products expression for the output X can be obtained by oRing together those squares that contain a 1.
- Looping an octet of 1s eliminates the three variables that appear in both complemented and uncomplemented form.
- Minimal cost solution is a valid logic design with the minimum number of gates with the minimum number of inputs.
- Don't-care condition input-output condition that never occurs during normal operation. Since the condition never occurs, you can use an X on the Karnaugh map. This X can be a 0 or a 1, whichever you prefer.

4.5 Keywords

- ***"Don't care" conditions*:** It is occur, we have to decide which one change to 0 and which to 1 to produce the best K-map looping
- ***AND gate*:** It is used to perform the logical multiplication also known as OR gate. An OR gate has two or more input signals and only one output signal.

Notes

- ***K-map squares*:** These are labelled so that horizontally adjacent squares differ only in one variable and similarly, vertically adjacent squares differ only in one variable.
- ***Looping*:** The expression for output X can be simplified by properly combining those squares in the 'K-map which contain 1s. The process for combining these 1s is called looping.
- ***Maxterm*:** It is the sum of N distinct literals where each literal occurs exactly once.
- ***Minterm*:** It is the product of N distinct literals where each literal occurs exactly once.
- ***OR gate*:** It is used to perform the logical addition also known as OR gate. An OR gate has two or more input signals and only one output signal.
- ***Quad*:** It is looped, the resultant term will contain only the variables do-not-change form for all the squares in the quad.
- ***Redundant group*:** A group of 1s on a Karnaugh map that are all part of other groups. You can eliminate any redundant group.

Lab Exercise

1. Prepare a Karnaugh map's truth table three values.
2. Write an equation of SOP minimization technique and solve it.

4.6 Review Questions

1. Explain the 6-variable K-map and give the example with circuit diagram.
2. Explain the 4-variable K-map and give the example.
3. What is the sum-of-products circuits for the truth table given below?

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

Notes

4. Draw the unsimplifid product of sum circuit for the table.

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

5. Simplify the given expression to its Sum-of-Products (SOP) form. Draw the logic circuit for the simplified SOP function $Y = (A + B)(A + \overline{AB})\ C + \bar{A}(B + \bar{C}) + \bar{A}B + ABC$

6. Write the expression for Boolean function

   $F(A, B, C) = \Sigma\, m\,(1, 4, 5, 6, 7)$ *in standard POS form.*

7. Which of the following expression is a sum of product?

   (*a*) $AB + CD + E$ (*b*) $AB(C + D)$

   (*c*) $(A + B)(C + D + E)$ (*d*) $(MN + PQ)$

8. State which of the following expressions are not in thesum of product form:

   (*a*) $RS\bar{T} + \bar{R}S\bar{T} + \bar{T}$ (*b*) $A\bar{D}C + \bar{A}DC$

   (*c*) $MN\bar{P} + (M + \bar{N})P$ (*d*) $AB + \bar{A}B\bar{C} + \overline{ABC}D$

9. Simplify the following expression into sum of products using Karnaugh map F(A,B,C,D) = Σ (1, 3, 4, 5, 6, 7, 9, 12, 13)

10. Simplify using K-map and draw the logic diagram for the given expression

    $F = \overline{ABC} + \overline{A\bar{B}}C + \bar{A}B\bar{C} + A\overline{BC}\ A\bar{B}C$

11. Minimize the logic function Y(A, B, C, D) = Σ M (0, 1, 2, 3, 5, 7, 8, 9, 11, 14). Use Karnaugh map. Draw logic circuit for the simplified function.

12. Use a Karnaugh map to simplify the equation F = Σ (1, 3, 5, 7, 9) with don't cares X = Σ (6, 12, 13).

Answer to Self Assessment

1. (*b*) 2. (*a*) 3. (*c*) 4. (*c*) 5. (*d*)

6. (*c*) 7. (*a*) 8. (*c*) 9. (*b*) 10. (*a*)

4.7 Further Reading

Notes

Books

Digital System – Design With Systemverilog, by Zwolinski Dr Mark.

Online link

http://babbage.cs.qc.edu/courses/Minimize/

Notes

Unit 5: Combinational Circuits

CONTENTS

Objectives

After studying this unit, you will be able to:

- Explain working of adder
- Discuss subtractor
- Understand encoder
- Discuss decoder

Introduction

Combinational (combinatorial) circuits realize Boolean functions and deal with digitized signals, usually denoted by 0s and 1s. The behaviour of a combinational circuit is memoryless; that is, given a stimulus to the input of a combinational circuit, a response appears at the output after some propagation delay, but the response is not stored or fed back. Simply put, the output depends solely on its most recent input and is independent of the circuit's past history. Design of a combinational circuit begins with a behavioural specification and selection of the implementation technique. These are then followed by simplification, hardware synthesis, and verification.

Combinational circuits can be specified via Boolean logic expressions, structural descriptions, or truth tables. Various implementation techniques, using fixed and programmable components, are outlined in the rest of this article. Combinational circuits implemented with fixed logic tend to be more expensive in terms of design effort and hardware cost, but they are often both faster and denser and consume less power. They are thus suitable for high-speed circuits and/or high-volume production. Implementations that use memory devices or programmable logic circuits,

Notes

on the other hand, are quite economical for low-volume production and rapid prototyping, but may not yield the best performance, density, or power consumption.

Simplification is the process of choosing the least costly implementation from among feasible and equivalent implementations with the targeted technology.

5.1 Adder

In electronics, an adder is a digital circuit that performs addition of binary numbers. In modern computers adders reside in the arithmetic logic unit (ALU).

Adders are important not only in computers but also in many types of digital systems in which the numeric data are processed.

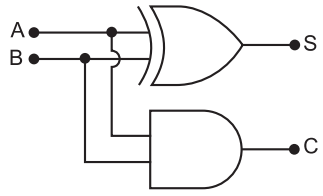
There are two types of adders:

- Half adder
- Full adder

5.1.1 Half Adder

The half adder is an example of a simple, functional digital circuit built from two logic gates. The half adder adds to one-bit binary numbers (AB). The output is the sum of the two bits (S) and the carry (C). Note how the same two inputs are directed to two different gates. The inputs to the XOR gate are also the inputs to the AND gate. The input "wires" to the XOR gate are tied to the input wires of the AND gate; thus, when voltage is applied to the A input of the XOR gate, the A input to the AND gate receives the same voltage.

Figure 5.1: (*a*) Logic Diagram of Half Adder (*b*) Truth Table of Half Adder



(*a*)

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

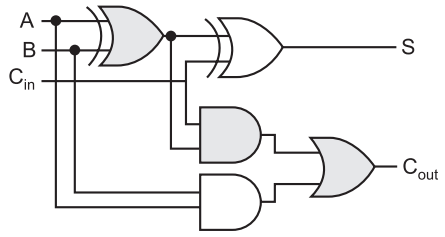
(*b*)

5.1.2 Full Adder

The full-adder circuit adds three one-bit binary numbers (C A B) and outputs two one-bit binary numbers, a sum (S) and a carry (C1). The full adder is usually a component in a cascade of adders, which add 8, 16, 32, etc. binary numbers. The carry input for the full-adder circuit is from the carry output from the circuit "above" itself in the cascade. The carry output from the full adder is fed to another full adder "below" itself in the cascade. If you look closely, you will see the full adder is simply two half adders joined by an OR.

Notes

Figure 5.2: (*a*) Logic Diagram of Full Adder (*b*) Truth Table of Full Adder



(*a*)

A	B	Cin	S	C
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

(*b*)

Did u know?

Each full adder requires three levels of logic. In a 32-bit [ripple carry] adder, there are 32 full adders, so the critical path (worst case) delay is 3 (for carry propagation in first adder) + 31 * 2 (for carry propagation in later adders) = 65 gate delays.

5.2 Subtractor

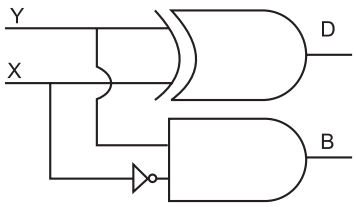
Subtractor circuits take two binary numbers as input and subtract one binary number input from the other binary number input. Similar to adders, it gives out two outputs, difference and borrows (carry–in the case of Adder). There are two types of subtractors:

- Half Subtractor
- Full Subtractor

5.2.1 Half Subtractor

The half-subtractor is a combinational circuit which is used to perform subtraction of two bits. It has two inputs, X (minuend) and Y (subtrahend) and two outputs D (difference) and B (borrow).

Figure 5.3: (*a*) Logic Diagram of Half Subtractor (*b*) Truth Table of Half Subtractor



(*a*)

Notes

Truth table

The truth table for the half subtractor is given below:

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

(*b*)

From the above table one can draw the Karnaugh map for "difference" and "borrow". So, Logic equations are:

$$D = X \oplus Y$$

$$B = \bar{X} \cdot Y$$

5.2.2 Full Subtractor

As in the case of the addition using logic gates, a full subtractor is made by combining two half subtractors and an additional OR gate. A full subtractor has the borrow–in capability and so allows cascading which results in the possibility of multi-bit subtraction. The circuit diagram for a full subtractor is given below:

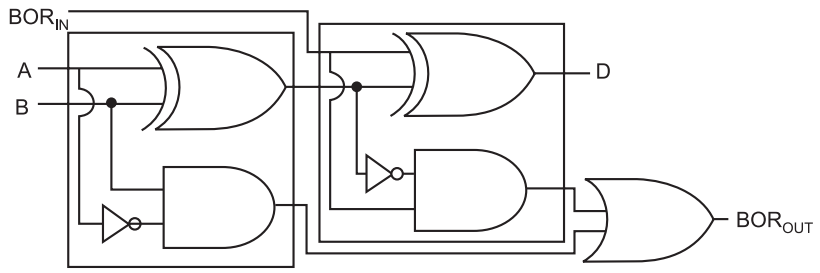
Easy way to write truth table D = X – Y – Z (don't bother about sign) B = 1 If X < (Y + Z)

So, Logic equations are:

$$D = X \oplus Y \oplus Z$$

$$B = Z \cdot (\overline{X \oplus Y}) + \bar{X} \cdot Y$$

Figure 5.4: (*a*) Logic Diagram of Full Subtractor (*b*) Truth Table of Full Subtractor



(*a*)

Truth table

The truth table for the full subtractor is given below:

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

(*b*)

Notes

Self Assessment

Choose the correct answer:

1. The gates required to build a half adder are:
   (*a*) EX-OR gate and NOR gate (*b*) EX-OR gate and OR gate
   (*c*) EX-OR gate and AND gate (*d*) Four NAND gates.
2. A full adder logic circuit will have:
   (*a*) Two inputs and one output. (*b*) Three inputs and three outputs.
   (*c*) Two inputs and two outputs. (*d*) Three inputs and two outputs.
3. In modern computer adders reside in the arithmetic logic unit (ALU).
   (*a*) True (*b*) False
4. Full-adder circuit adds only two binary numbers.
   (*a*) True (*b*) False
5. Binary encoder encodes information from ..... inputs into an n-bit code exactly one of the input signals.
   (*a*) n^2 (*b*) 1
   (*c*) n (*d*) 2n

5.3 Encoder

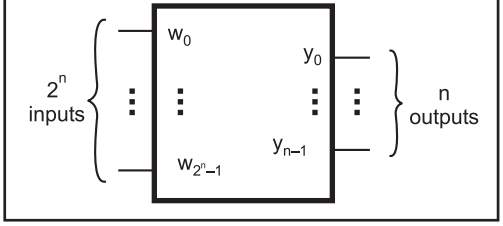
An encoder performs the opposite function of a decoder. It encodes given information into a more compact form.

5.3.1 Binary Encoder

A binary encoder encodes information from 2n inputs into an n-bit code, as indicated in Figure 5,5. Exactly one of the input signals should have a value of 1, and the outputs present the binary number that identifies which input is equal to 1. The truth table for a 4-to-2 encoder is provided in Figure 5.6. Observe that the output y0 is 1 when either input w1 or w3 is 1, and output y1 is 1 when input w2 or w3 is 1. Hence these outputs can be generated by the circuit in Figure 5.6. Note that we assume that the inputs are one-hot encoded. All input patterns that have multiple inputs set to 1 are not shown in the truth table, and they are treated as do-not-care conditions.

Encoders are used to reduce the number of bits needed to represent given information. A practical use of encoders is for transmitting information in a digital system. Encoding the information allows the transmission link to be built using fewer wires. Encoding is also useful if information is to be stored for later use because fewer bits need to be stored.

Figure 5.5: A 2n-to-n Binary Encoder

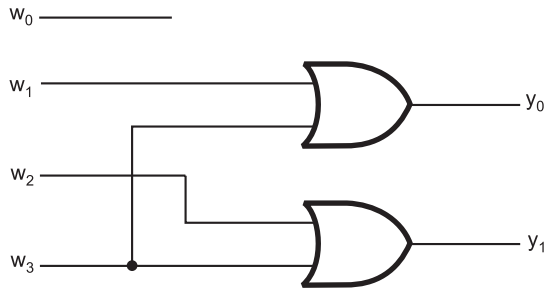


Notes

Figure 5.6: A 4-to-2 Binary Encoder

w_3	w_2	w_1	w_0	y_1	y_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

(*a*) Truth table



(*b*) Circuit

Task Draw a logic diagram for 8-to-2 binary encoder.

5.3.2 Priority Encoder

Another useful class of encoders is based on the priority of input signals. In a priority encoder each input has a priority level associated with it. The encoder outputs indicate the active input that has the highest priority. When an input with a high priority is asserted, the other inputs with lower priority are ignored. The truth table for a 4-to-2 priority encoder is shown in Figure 5.7. It assumes that w0 has the lowest priority and w3 the highest. The outputs y1 and y0 represent the binary number that identifies the highest priority input set to 1. Since it is possible that none of the inputs is equal to 1, an output, z, is provided to indicate this condition. It is set to 1 when at least one of the inputs is equal to 1. It is set to 0 when all inputs are equal to 0.

Figure 5.7: Truth Table for a 4-to-2 Priority Encoder

w_3	w_2	w_1	w_0	y_1	y_0	z
0	0	0	0	d	d	0
0	0	0	1	0	0	1
0	0	1	x	0	1	1
0	1	x	x	1	0	1
1	x	x	x	1	1	1

The outputs $y1$ and $y0$ are not meaningful in this case, and hence the first row of the truth table can be treated as a don't-care condition for $y1$ and $y0$. The behaviour of the priority encoder is most easily understood by first considering the last row in the truth table. It specifies that if input $w3$ is 1, then the outputs are set to $y1y0 = 11$. Because w3 has the highest priority level, the values of inputs $w2$, $w1$, and $w0$ do not matter. To reflect the fact that their values are irrelevant, $w2$, $w1$, and $w0$ are denoted by the symbol x in the truth table. The second-last row in the truth table stipulates that if $w2 = 1$, then the outputs are set to $y1y0 = 10$, but only if $w3 = 0$. Similarly, input $w1$ causes

Notes

the outputs to be set to $y1y0 = 01$ only if both $w3$ and $w2$ are 0. Input $w0$ produces the outputs $y1y0 = 00$ only if $w0$ is the only input that is asserted. Alogic circuit that implements the truth table can be synthesized by using the techniques developed. However, a more convenient way to derive the circuit is to define a set of intermediate signals, i0, . . . , i3, based on the observations above. Each signal, ik , is equal to 1 only if the input with the same index, wk , represents the highest-priority input that is set to 1. The logic expressions for i0, . . . , i3 are

$$i_0 = \overline{w}_3 \overline{w}_2 \overline{w}_1 w_0$$

$$i_1 = \overline{w}_3 \overline{w}_2 w_1$$

$$i_2 = \overline{w}_3 w_2$$

$$i_3 = w_3$$

Using the intermediate signals, the rest of the circuit for the priority encoder has the same structure as the binary encoder in Figure 5.6, namely

$$y_0 = i_1 + i_3$$

$$y_1 = i_2 + i_3$$

The output z is given by

$$z = i_0 + i_1 + i_2 + i_3$$

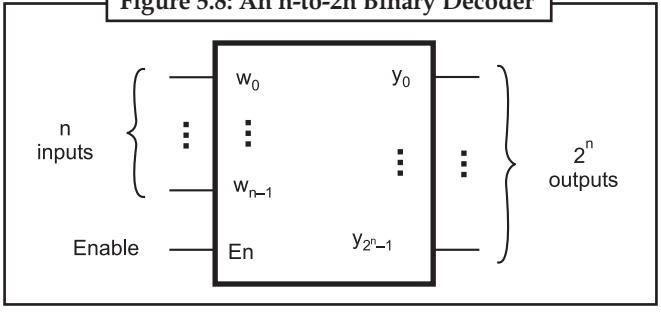
A simple CPU with 8 registers may use 3-to-8 logic decoders inside the instruction decoder to select two source registers of the register file to feed into the ALU as well as the destination register to accept the output of the ALU.

Prepare a truth table of 16-to-4 binary decoder.

5.4 Decoder

Decoder circuits are used to decode encoded information. A binary decoder, depicted in Figure 5.8, is a logic circuit with n inputs and 2^n outputs. Only one output is asserted at a time, and each output corresponds to one valuation of the inputs. The decoder also has an enable input, En, that is used to disable the outputs; if En = 0, then none of the decoder outputs is asserted. If En = 1, the valuation of $w_{n-1} \cdots w_1 w_0$ determines which of the outputs is asserted. An n-bit binary code in which exactly one of the bits is set to 1 at a time is referred to as one-hot encoded, meaning that the single bit that is set to 1 is deemed to be "hot."

Figure 5.8: An n-to-2n Binary Decoder



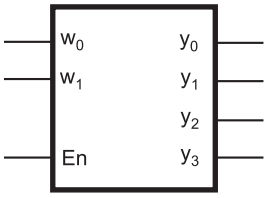
The outputs of a binary decoder are one-hot encoded. A 2-to-4 decoder is given in Figure 5.9. The two data inputs are w1 and w0. They represent a two-bit number that causes the decoder to

Notes

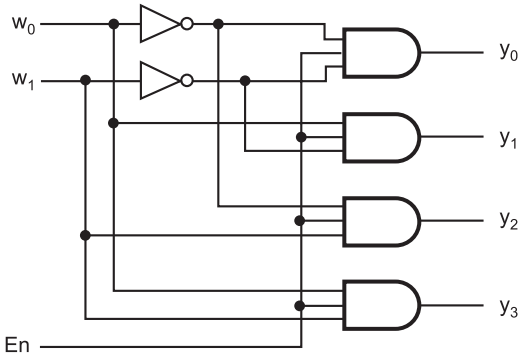
assert one of the outputs y0, ... , y3. Although a decoder can be designed to have either active-high or active-low outputs, in Figure 5.9 active-high outputs are assumed setting the inputs w1w0

Figure 5.9: A 2-to-4 Decoder

En	w_1	w_0	y_0	y_1	y_2	y_3
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1
0	x	x	0	0	0	0

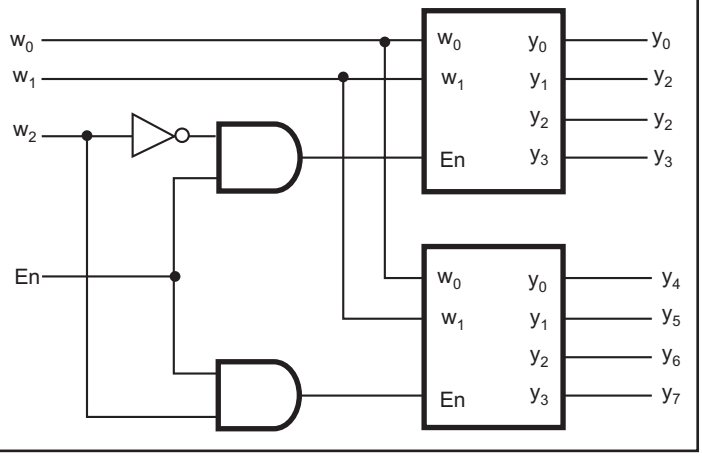


(*a*) Graphical symbol (*b*) Graphical symbol



(*c*) Logic circuit

Figure 5.10: A 3-to-8 Decoder Using two 2-to-4 Decoders



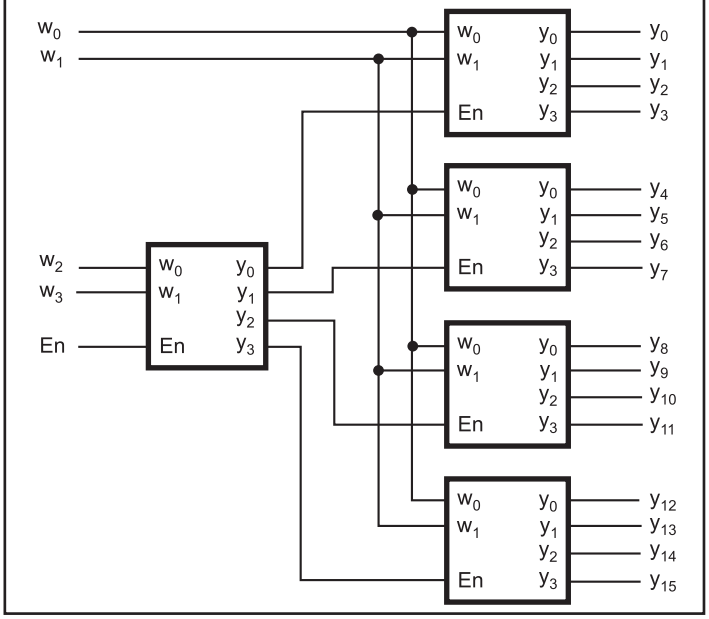
to 00, 01, 10, or 11 causes the output y0, y1, y2, or y3 to be set to 1, respectively. A graphical symbol for the decoder is given in part (*b*) of the figure, and a logic circuit is shown in part (*c*).

Notes

Larger decoders can be built using the sum-of-products structure in Figure 5.9 c, or else they can be constructed from smaller decoders. Figure 5.10 shows how a 3-to-8 decoder is built with two 2-to-4 decoders. The w2 input drives the enable inputs of the two decoders. The top decoder is enabled if w2 = 0, and the bottom decoder is enabled if w2 = 1. This concept can be applied for decoders of any size. Figure 5.11 shows how five 2-to-4 decoders can be used to construct a 4-to-16 decoder. Because of its treelike structure, this type of circuit is often referred to as a decoder tree.

Decoders are useful for many practical purposes. We showed the sum-of-products implementation of the 4-to-1 multiplexer, which requires AND gates to distinguish the four different valuations of the select inputs s1 and s0. Since a decoder evaluates the values on its inputs, it can be used to build a multiplexer as illustrated in Figure 5.11. The enable input of the decoder is not needed in this case, and it is set to 1. The four outputs of the decoder represent the four valuations of the select inputs.

Figure 5.11: A 4-to-16 Decoder Built using a Decoder Tree



Did u know?

Be careful in the manual decoding, it should be followed in all cases and each circuit should be tested as it is completed.

We can convert a 3-to-8 decoder from two 2-to-4 decoders with enabling signals.

Case Study

Combinatorial Game Theory

Combinatorial game theory (CGT) is a branch of applied mathematics and theoretical computer science that studies sequential games with perfect information, that is, two-player games which have a position in which the players take turns changing in defined ways or moves to achieve a defined winning condition. CGT does not study games with imperfect or incomplete information (sometimes called games of chance, like poker).

Contd...

Notes

It restricts itself to games whose position is public to both players, and in which the set of available moves is also public (perfect information). Combinatorial games include well-known games like chess, checkers, Go, Arimaa, Hex, and Connect6. They also include one-player combinatorial puzzles, and even no-player automata, like Conway's Game of Life.[2] In CGT, the moves in these games are represented as a game tree.

Game theory in general includes games of chance, games of imperfect knowledge, and games in which players can move simultaneously, and they tend to represent real-life decision-making situations. CGT has a different emphasis than "traditional" or "economic" game theory, which was initially developed to study games with simple combinatorial structure, but with elements of chance (although it also considers sequential moves, see extensive-form game). Essentially, CGT contributed new methods for analyzing game trees, for example using surreal numbers, which are a subclass of all two-player perfect-information games. The type of games studied by CGT is also of interest in artificial intelligence, particularly for automated planning and scheduling. In CGT there has been less emphasis on refining practical search algorithms (like the alpha-beta pruning heuristic, included in most artificial intelligence textbooks today), but more emphasis on descriptive theoretical results, like measures of game complexity or proofs of optimal solution existence without necessarily specifying an algorithm (see strategy-stealing argument, for instance).

An important notion in CGT is that of solved game (which has several flavours), meaning for example that one can prove that the game of tic-tac-toe results in a draw if both players play optimally. While this is a trivial result, deriving similar results for games with rich combinatorial structures is difficult. For instance, in 2007 it was announced that checkers has been (weakly, but not strongly) solved—optimal play by both sides also leads to a draw—but this result was a computer-assisted proof. Other real-world games are mostly too complicated to allow complete analysis today (although the theory has had some recent successes in analyzing Go endgames). Applying CGT to a position attempts to determine the optimum sequence of moves for both players until the game ends, and by doing so discover the optimum move in any position. In practice, this process is tortuously difficult unless the game is very simple.

History

CGT arose in relation to the theory of impartial games, in which any play available to one player must be available to the other as well. One very important such game is nim, which can be solved completely. Nim is an impartial game for two players, and subject to the normal play condition, which means that a player who cannot move loses. In the 1930s, the Sprague-Grundy theorem showed that all impartial games are equivalent to heaps in nim, thus showing that major unifications are possible in games considered at a combinatorial level (in which detailed strategies matter, not just pay-offs).

In the 1960s, Elwyn R. Berlekamp, John H. Conway and Richard K. Guy jointly introduced the theory of a partisan game, in which the requirement that a play available to one player be available to both is relaxed. Their results were published in their book **Winning Ways for Your Mathematical Plays** in 1982. However, the first book published on the subject was Conway's **On Numbers and Games**, also known as ONAG, which introduced the concept of surreal numbers and the generalization to games. **On Numbers and Games** was also a fruit of the collaboration between Berlekamp, Conway, and Guy.

Combinatorial games are generally, by convention, put into a form where one player wins when the other has no moves remaining. It is easy to convert any finite game with only two possible results into an equivalent one where this convention applies. One of the most

Contd...

Notes

important concepts in the theory of combinatorial games is that of the sum of two games, which is a game where each player may choose to move either in one game or the other at any point in the game, and a player wins when his opponent has no move in either game. This way of combining games leads to a rich and powerful mathematical structure.

John Conway states in ONAG that the inspiration for the theory of partisan games was based on his observation of the play in go endgames, which can often be decomposed into sums of simpler endgames isolated from each other in different parts of the board.

Questions:

1. Explain the concept of combinational game theory.
2. Who was the founder of game theory?

Self Assessment

Choose the correct answer:

6. Priority encoder: Each input has a priority level associated with it. The encoder outputs indicate the active input that has the highest priority.

   (*a*) True (*b*) False

7. Decoder is a logic circuit with n inputs and ..... outputs.

   (*a*) n^2 (*b*) 2^n

   (*c*) 1 (*d*) $n^2 + 1$

8. Which digital system translates coded characters into a more useful form?

   (*a*) Encoder (*b*) Display

   (*c*) Counter (*d*) Decoder

9. How many inputs are required for a 1-of-10 BCD decoder?

   (*a*) 4 (*b*) 8

   (*c*) 10 (*d*) 1

5.5 Summary

- The full adder is usually a component in a cascade of adders, which add 8, 16, 32, etc. binary numbers.
- Decoders are useful for many practical purposes. We showed the sum-of products implementation of the 4-to-1 multiplexer, which requires AND gates to distinguish the four different valuations of the select inputs s1 and s0.
- A full subtractor has the borrow-in capability and so allows cascading which results in the possibility of multi-bit subtraction.
- The purpose of the decoder and encoder circuits is to convert from one type of input encoding to a different output encoding.
- The half-subtractor is a combinational circuit which is used to perform subtraction of two bits.

5.6 Keywords

***Adder*:** This is a digital circuit that performs addition of binary numbers. In modern computers adders reside in the arithmetic logic unit (ALU).

Notes

***Binary encoder*:** It encodes information from 2n inputs into an n-bit code. Exactly one of the input signals should have a value of 1, and the outputs present the binary number that identifies which input is equal to 1.

***Decoder*:** This circuit is used to decode encoded information. A binary decoder is a logic circuit with n inputs and 2n outputs.

***Priority encoder*:** Here each input has a priority level associated with it the encoder outputs indicate the active input that has the highest priority.

***Segment decoder*:** It converts one binary-coded decimal (BCD) digit into information suitable for driving a digit-oriented display.

***Subtractor*:** This circuit takes two binary numbers as input and subtracts one binary number input from the other binary number input. Similar to adders, it gives out two outputs, difference and borrows (carry–in the case of Adders).

Lab Exercise

1. Write logic equations of half adder and draw circuit diagram.
2. Draw a circuit diagram for 3-to-8 decoder using two 2-to-4 decoders.

5.7 Review Questions

1. Describe the adder in brief. And draw the circuit diagram.
2. What do you understand by comparators?
3. Define the BCD-to-7-segment decoder.
4. What is subtractor? Define the types of subtractor.
5. What do you understand by encoder? Draw the circuit diagram.
6. What do you understand by decoder? Write the truth table.
7. Differentiate between half adder and full adder.
8. Write the truth table of full adder.
9. Differentiate between half subtractor and full subtractor.
10. Explain the Priority Encoders.

Answers to Self Assessment

1. (*c*) 2. (*b*) 3. (*a*) 4. (*b*) 5. (*d*)
6. (*a*) 7. (*b*) 8. (*d*) 9. (*a*)

5.8 Further Reading

Books

Digital Fundamentals: Ninth Edition, by Floyd Thomas L.

Online link

http://www.ariese.articlealley.com/combinational-circuit-design-165328.html

Notes

Unit 6: Implementation of Combinational Logic Circuit

CONTENTS

Objectives

After studying this unit, you will be able to:

- Discuss the multiplexer
- Describe the demultiplexer
- Explain the code converters
- Describe the comparators

Introduction

Combinational logic (sometimes also referred to as combinatorial logic) is a type of digital logic which is implemented by Boolean circuits, where the output is a pure function of the present input only. This is in contrast to sequential logic, in which the output depends not only on the present input but also on the history of the input. In other words, sequential logic has memory while combinational logic does not. Combinational logic is used for building circuits where certain outputs are desired, given certain inputs. The construction of combinational logic is generally done using one of two methods: a sum-of-products, or a product-of-sums.

This unit introduces several combinational circuits that are frequently used by digital designers, including a data selector (also called a multiplexor or just "mux"), a binary decoder, a seven-segment decoder, an encoder, and a shifter. Each of these circuits can be used by themselves in the solution of some simpler logic problems, but they are more often used as building blocks in the creation of larger, more complex circuits. In this module, these circuits will be developed from first principles following a general design procedure that will serve as a model for all later designs. In later modules, these circuits will be used as modular (or "macro") building blocks in larger designs.

Notes

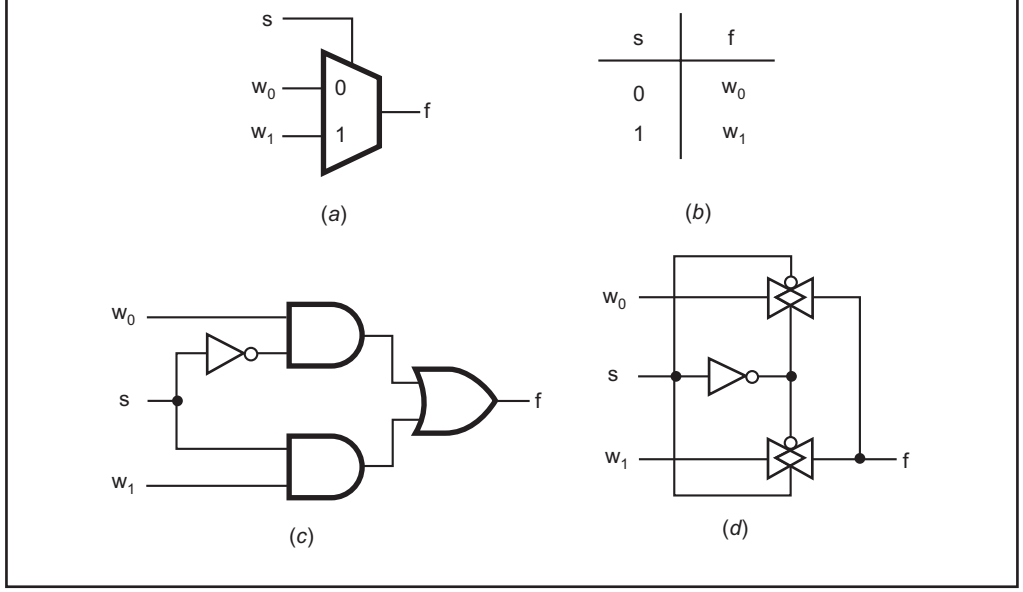
This general design procedure has five main steps. First, you must gain a clear understanding of the design intent of each circuit before any design activities start. When you are doing original design work, this understanding comes from many sources, including other persons, previous or competing designs, research papers, or your own insightful thinking. For now, the discussion that leads the presentation of each new circuit is intended to impart that clear understanding to you. Second, a block diagram that shows all circuit inputs and outputs will be developed. A block diagram is an indispensable part of any design, especially when dealing with complex circuits. In conceiving and capturing a block diagram, you are committing to a set of input and output signals, and those signals define the context and boundaries of your design. Third, the design requirements will be captured in an engineering formalism like a truth table or logic equation. This formalism removes all ambiguity from the design, and establishes a solid specification for the circuit. Fourth, the formally stated requirements will be used to find minimal circuits that meet the specifications. And finally, those minimal circuits will be created and implemented using the ISE/WebPack tool and a Digilent board, and verified in hardware to ensure they meet their behavioural requirements. In practice, a few types of logic circuits are often used as building blocks in larger designs.

6.1 Multiplexers

A multiplexer circuit has a number of data inputs, one or more select inputs, and one output. It passes the signal value on one of the data inputs to the output. The data input is selected by the values of the select inputs. Figure 6.1 shows a 2-to-1 multiplexer. Part (*a*) gives the symbol commonly used. The select input, s, chooses as the output of the multiplexer either input w0 or w1. The multiplexer's functionality can be described in the form of a truth table as shown in part (*b*) of the figure. Part (*c*) gives a sum-of-products implementation of the 2-to-1 multiplexer, and part (*d*) illustrates how it can be constructed with transmission gates.

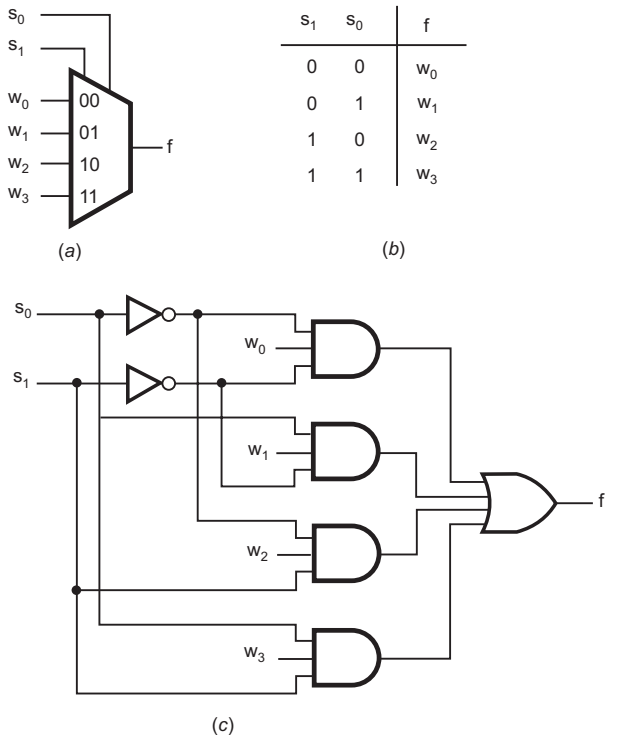
Figure 6.2a depicts a larger multiplexer with four data inputs, w_0,……..,w_3, and two select inputs, s_1 and s_0. As shown in the truth table in part (*b*) of the figure, the two-bit number represented by s1s0 selects one of the data inputs as the output of the multiplexer.

Figure 6.1: A 2-to-1 Multiplexer (*a*) Graphical Symbol (*b*) Truth Table (*c*) Sum-of-Products Circuit (*d*) Circuit with Transmission Gates



Notes

Figure 6.2: A 4-to-1 Multiplexer (*a*) Graphical Symbol (*b*) Truth Table (*c*) A 4-to-1 Multiplexer

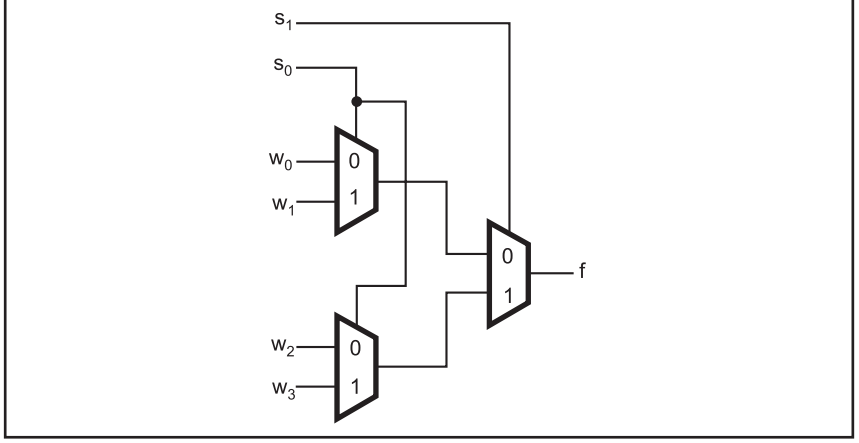


A sum-of-products implementation of the 4-to-1 multiplexer appears in Figure 6.3c. It realizes the multiplexer function

$$f = \acute{s}_1\acute{s}_0 w_0 + \acute{s}_1 s_0 w_1 + s_1\acute{s}_0 w_2 + s_1 s_0 w_3$$

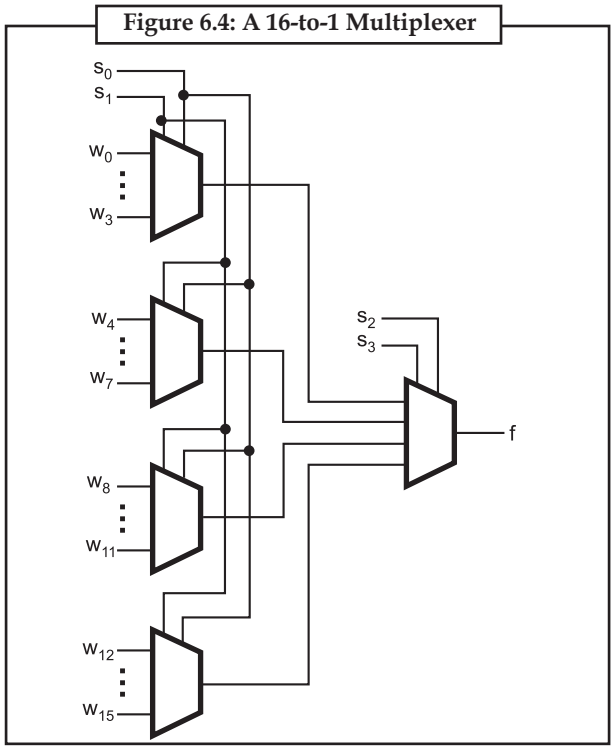
It is possible to build larger multiplexers using the same approach. Usually, the number of data inputs, n, is an integer power of two. A multiplexer that has n data inputs, w_0, ..., w_{n-1}, requires $[\log_2 n]$ select inputs. Larger multiplexers can also be constructed from smaller multiplexers. For example, the 4-to-1 multiplexer can be built using three 2-to-1 multiplexers as illustrated in Figure 6.3. If the 4-to-1 multiplexer is implemented using transmission gates, then the structure in this figure is always used. Figure 6.4 shows how a 16-to-1 multiplexer is constructed with five 4-to-1 multiplexers.

Figure 6.3: Using 2-to-1 Multiplexers to Build a 4-to-1 Multiplexer



Notes

Figure 6.4: A 16-to-1 Multiplexer

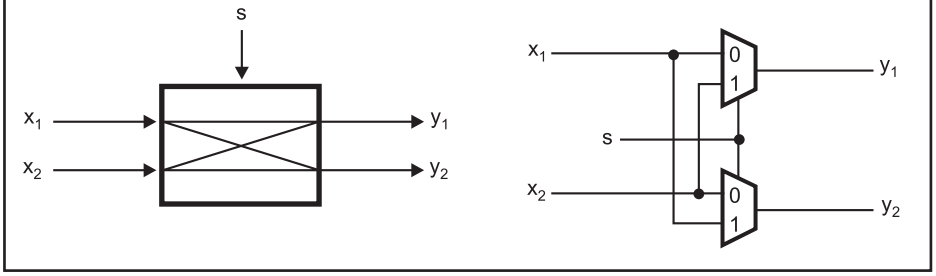


Task Prepare a truth table of 4-to-1 Multiplexer.

Example: Figure 6.5 shows a circuit that has two inputs, x_1 and x_2, and two outputs, y_1 and y_2. As indicated by the blue lines, the function of the circuit is to allow either of its inputs to be connected to either of its outputs, under the control of another input, s. A circuit that has n inputs and k outputs, whose sole function is to provide a capability to connect any input to any output, is usually referred to as an n × k crossbar switch. Crossbars of various sizes can be created, with different numbers of inputs and outputs. When there are two inputs and two outputs, it is called a 2 × 2 crossbar.

Figure 6.6b shows how the 2 × 2 crossbar can be implemented using 2-to-1 multiplexers. The multiplexer select inputs are controlled by the signal s. If s = 0, the crossbar connects x_1 to y_1 and x_2 to y_2, while if s = 1, the crossbar connects x_1 to y_2 and x_2 to y_1. Crossbar switches are useful in many practical applications in which it is necessary to be able to connect one set of wires to another set of wires, where the connection pattern changes from time to time.

Figure 6.5: A Practical Application of Multiplexers
(*a*) A 2 x 2 Crossbar Switch (*b*) Implementation using Multiplexers



Notes

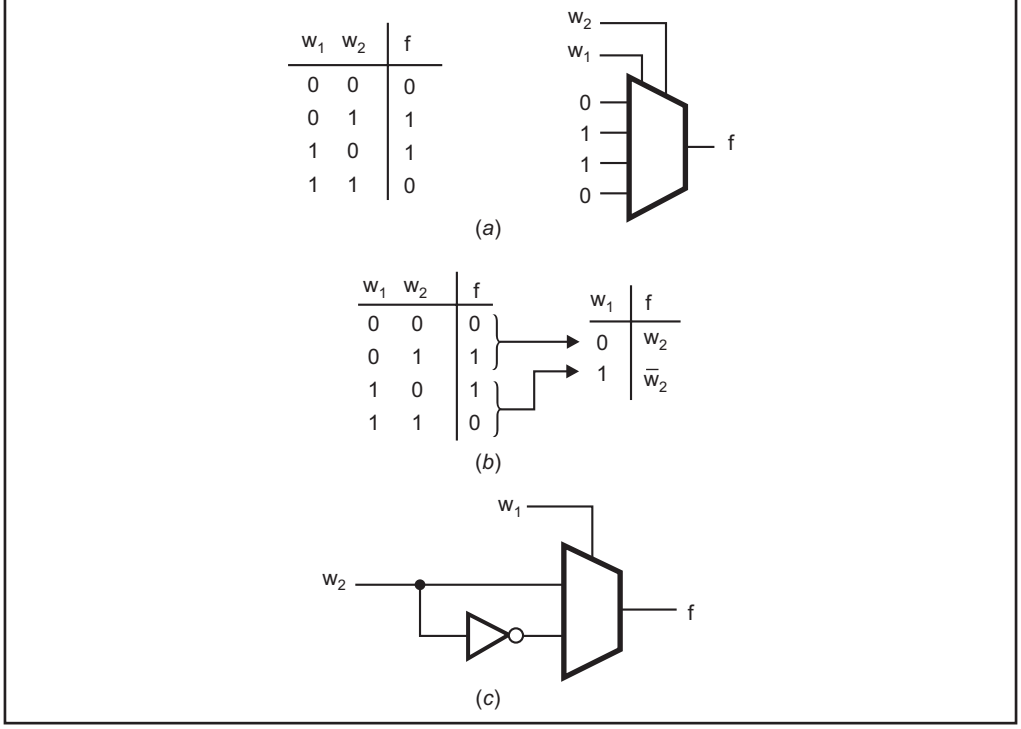
Did u know? Multiplex systems were originally implemented to use the spare capacity available on a line (circuit) for concurrent character or message transmission.

6.1.1 Synthesis of Logic Functions Using Multiplexers

Multiplexers are useful in many practical applications, such as those described above. They can also be used in a more general way to synthesize logic functions. Consider the example in Figure 6.6a. The truth table defines the function $f = w_1 \oplus w_2$. This function can be implemented by a 4-to-1 multiplexer in which the values of f in each row of the truth table are connected as constants to the multiplexer data inputs. The multiplexer select inputs are driven by w_1 and w_2. Thus for each valuation of w_1w_2, the output f is equal to the function value in the corresponding row of the truth table.

The above implementation is straightforward, but it is not very efficient. A better implementation can be derived by manipulating the truth table as indicated in Figure 6.6b, which allows f to be implemented by a single 2-to-1 multiplexer. One of the input signals, w_1 in this example, is chosen as the select input of the 2-to-1 multiplexer. The truth table is redrawn to indicate the value of f for each value of w_1. When $w_1 = 0$, f has the same value as input w_2, and when $w_1 = 1$, f has the value of w_2. The circuit that implements this truth table is given in Figure 6.6c. This procedure can be applied to synthesize a circuit that implements any logic function.

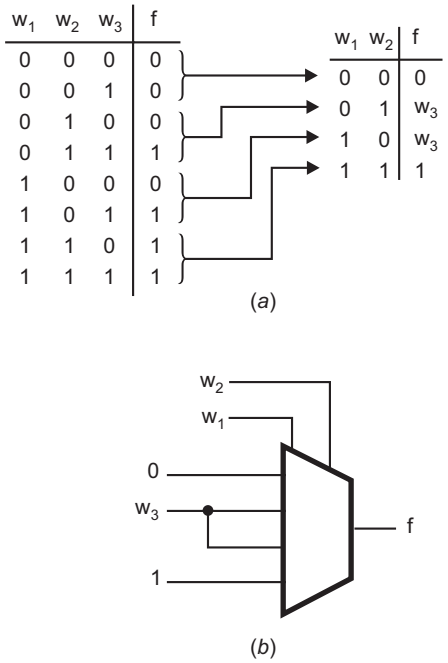
Figure 6.6: Synthesis of a Logic Function using Multiplexers
(*a*) Implementation Using a 4-to-1 Multiplexer (*b*) Modified Truth Table (*c*) Circuit



Example: Figure 6.7a gives the truth table for the three-input majority function, and it shows how the truth table can be modified to implement the function using a 4-to-1 multiplexer. Any two of the three inputs may be chosen as the multiplexer select inputs. We have chosen w1 and w2 for this purpose, resulting in the circuit in Figure 6.7b.

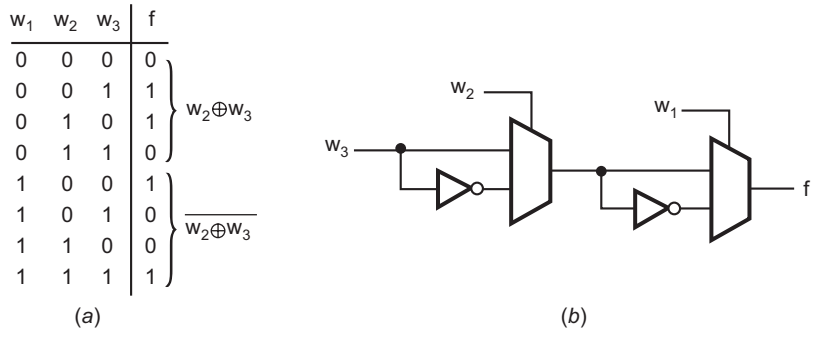
Notes

Figure 6.7: Implementation of the Three-input Majority Function using a 4-to-1 Multiplexer (a) Modified Truth Table (b) Circuit



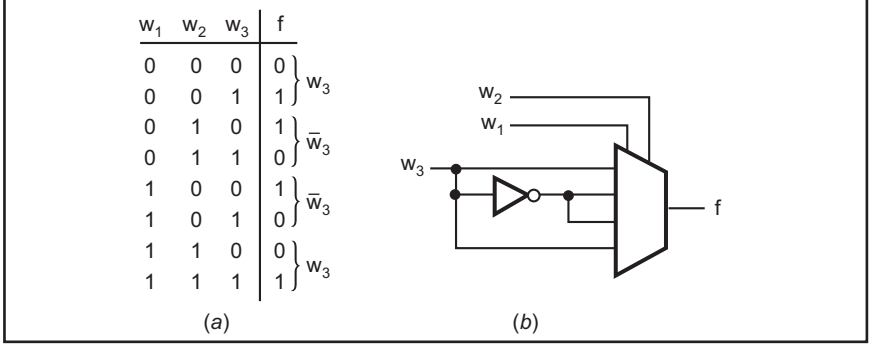
Example: Figure 6.8a indicates how the function $f = w_1 \oplus w_2 \oplus w_3$ can be implemented using 2-to-1 multiplexers. When $w_1 = 0$, f is equal to the XOR of w_2 and w_3, and when $w_1 = 1$, f is the XNOR of w_2 and w_3. The left multiplexer in the circuit produces $w_2 \oplus w_3$, using the result from Figure 6.7, and the right multiplexer uses the value of w_1 to select either $w_2 \oplus w_3$ or its complement. Note that we could have derived this circuit directly by writing the function as $f = (w_2 \oplus w_3) \oplus w_1$. Figure 6.9 gives an implementation of the three-input XOR function using a 4-to-1 multiplexer. Choosing w_1 and w_2 for the select inputs results in the circuit shown.

Figure 6.8: Three-input XOR Implemented with 2-to-1 Multiplexers (a) Truth Table (b) Circuit



Notes

Figure 6.9: Three-input XOR Implemented with a 4-to-1 Multiplexer (*a*) Truth Table (*b*) Circuit



6.1.2 Multiplexer Synthesis Using Shannon's Expansion

In each case the inputs to the multiplexers are the constants 0 and 1, or some variable or its complement. Besides using such simple inputs, it is possible to connect more complex circuits as inputs to a multiplexer, allowing function to be synthesized using a combination of multiplexers and other logic gates. Suppose that we want to implement the three-input majority function in Figure 6.7 using a 2-to-1 multiplexer in this way. Figure 6.10 shows an intuitive way of realizing this function. The truth table can be modified as shown on the right. If $w_1 = 0$, then $f = w_2w_3$, and if $w_1 = 1$, then $f = w_2 + w_3$. Using w_1 as the select input for a 2-to-1 multiplexer leads to the circuit in Figure 6.10b. This implementation can be derived using algebraic manipulation as follows: The function in Figure 6.10a is expressed in sum-of-products form as

$$f = \hat{w}_1w_2w_3 + w_1\hat{w}_2w_3 + w_1w_2\hat{w}_3 + w_1w_2w_3$$

It can be manipulated into

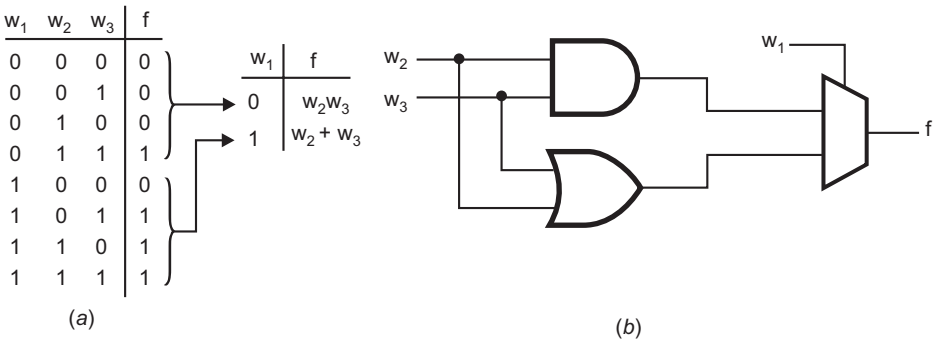
$$f = \hat{w}_1(w_2w_3) + w_1(\hat{w}_2w_3 + w_2\hat{w}_3 + w_2w_3)$$

$$= \hat{w}_1(w_2w_3) + w_1(w_2 + w_3)$$

which corresponds to the circuit in Figure 6.10b.

Multiplexer implementations of logic functions require that a given function be decomposed in terms of the variables that are used as the select inputs. This can be accomplished by means of a theorem proposed by Claude Shannon.

Figure 6.10: The Three-input Majority Function Implemented using a 2-to-1 Multiplexer (*a*) Truth table (*b*) Circuit



Notes

Self Assessment

Choose the correct answer:

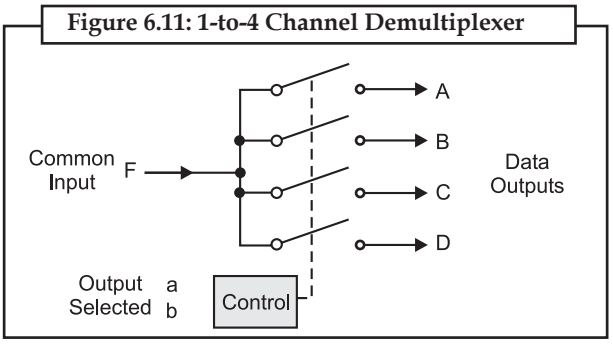
1. A device which converts BCD to Seven Segment is called:
   (*a*) Encoder (*b*) Decoder
   (*c*) Multiplexer (*d*) Demultiplexer
2. The following switching functions are to be implemented using a Decoder:
   $f_1 = \Sigma m(1, 2, 4, 8, 10, 14)$, $f_2 = \Sigma m(2, 5, 9, 11)$, $f_3 = \Sigma m(2, 4, 5, 6, 7)$
   The minimum configuration of the decoder should be
   (*a*) 2 – to – 4 line (*b*) 3 – to – 8 line.
   (*c*) 4 – to – 16 line (*d*) 5 – to – 32 line.
3. Combinational circuit consists of input variables, logic gates and output variables.
   (*a*) True (*b*) False
4. One multiplexer can take the place of:
   (*a*) several SSI logic gates
   (*b*) combinational logic circuits
   (*c*) several Ex-NOR gates
   (*d*) several SSI logic gates or combinational logic circuits

6.2 Demultiplexer

The demultiplexer is the inverse of the multiplexer, in that it takes a single data input and n address inputs. It has 2^n outputs. The address input determine which data output is going to have the same value as the data input. The other data outputs will have the value 0.

The data distributor, known more commonly as a Demultiplexer or "Demux", is the exact opposite of the Multiplexer. The demultiplexer takes one single input data line and then switches it to any one of a number of individual output lines one at a time. The demultiplexer converts a serial data signal at the input to a parallel data at its output lines as shown below:

Figure 6.11: 1-to-4 Channel Demultiplexer



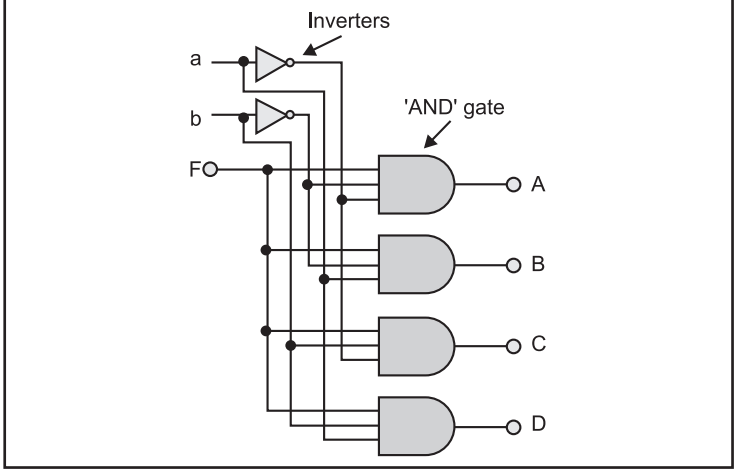
The function of the Demultiplexer is to switch one common data input line to any one of the 4 output data lines A to D in our example above. As with the multiplexer the individual solid state switches are selected by the binary input address code on the output select pins "a" and "b" and by adding more address line inputs it is possible to switch more outputs giving a 1-to-2^n data line outputs. Some standard demultiplexer IC´s also have an "enable output" input pin which disables or prevents the input from being passed to the selected output. Also some have latches built into their outputs to maintain the output logic level after the address inputs have been

Notes

changed. However, in standard decoder type circuits the address input will determine which single data output will have the same value as the data input with all other data outputs having the value of logic "0".

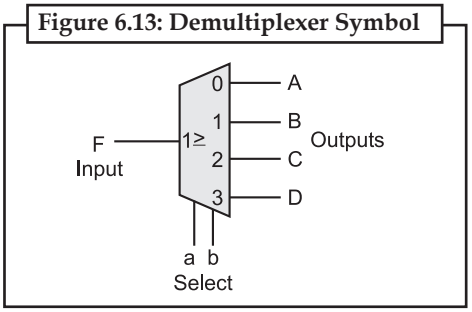
The implementation of the Boolean expression above using individual logic gates would require the use of six individual gates consisting of AND and NOT gates as shown:

Figure 6.12: 4 Channel Demultiplexer using Logic Gates



The symbol used in logic diagrams to identify a demultiplexer is as follows:

Figure 6.13: Demultiplexer Symbol



Standard Demultiplexer IC packages available are the TTL 74LS138 1 to 8-output demultiplexer, the TTL 74LS139 Dual 1-to-4 output demultiplexer or the CMOS CD4514 1-to-16 output demultiplexer. Another type of demultiplexer is the 24-pin, 74LS154 which is a 4-bit to 16-line demultiplexer/ decoder. Here the individual output positions are selected using a 4-bit binary coded input. Like multiplexers, demultiplexers can also be cascaded together to form higher order demultiplexers.

Unlike multiplexers which convert data from a single data line to multiple lines and demultiplexers which convert multiple lines to a single data line, there are devices available which convert data to and from multiple lines and in the next tutorial about combinational logic devices, we will look at Encoders which convert multiple input lines into multiple output lines, converting the data from one form to another.

Task Draw the circuit diagram and truth table of 2-to-1 multiplexer.

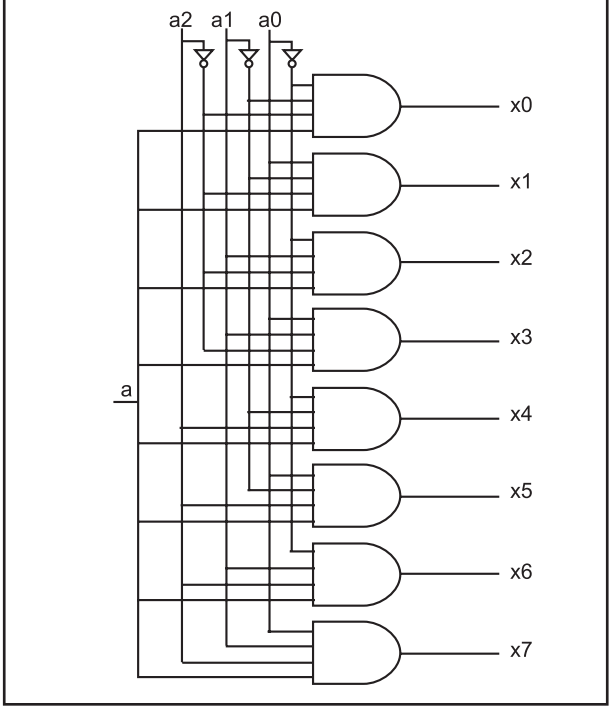
Notes

Example: Here is an abbreviated truth table for the demultiplexer. We could have given the full table since it has only 16 rows, but we will use the same convention as for the multiplexer where we abbreviated the values of the data inputs.

a2	a1	a0	d	x7	x6	x5	x4	x3	x2	x1	x0
0	0	0	c	0	0	0	0	0	0	0	c
0	0	1	c	0	0	0	0	0	0	c	0
0	1	0	c	0	0	0	0	0	c	0	0
0	1	1	c	0	0	0	0	c	0	0	0
1	0	0	c	0	0	0	c	0	0	0	0
1	0	1	c	0	0	c	0	0	0	0	0
1	1	0	c	0	c	0	0	0	0	0	0
1	1	1	c	c	0	0	0	0	0	0	0

Here is one possible circuit diagram for the demultiplexer:

Figure 6.14: Circuit Diagram of Demultiplexer



Caution

You must complete internal configruation of multiplexing before making the multiplexing process in any application, otherwise it may be failed.

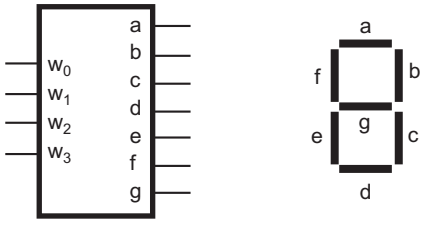
6.3 Code Converters

The purpose of the decoder and encoder circuits is to convert from one type of input encoding to a different output encoding. For example, a 3-to-8 binary decoder converts from a binary number on the input to a one-hot encoding at the output. An 8-to-3 binary encoder performs the opposite conversion. There are many other possible types of code converters. One common example is a BCD-to-7-segment decoder, which converts one binary-coded decimal (BCD) digit into information

Notes

suitable for driving a digit-oriented display. As illustrated in Figure 6.15a, the circuit converts the BCD digit into seven signals that are used to drive the segments in the display. Each segment is a small light-emitting diode (LED), which glows when driven by an electrical signal. The segments are labelled from a to g in the figure. The truth table for the BCD-to-7-segment decoder is given in Figure 6.15c. For each valuation of the inputs w_3,.....,w_0, the seven outputs are set to

Figure 6.15: A BCD-to-7-Segment Display Code Converter (*a*) Code Converter (*b*) 7-segment Display (*c*) Truth Table



(*a*)

(*b*)

w_3	w_2	w_1	w_0	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1

(*c*)

display the appropriate BCD digit. Note that the last 6 rows of a complete 16-row truth table are not shown. They represent don't-care conditions because they are not legal BCD codes and will never occur in a circuit that deals with BCD data. Acircuit that implements the truth table can be derived using the synthesis techniques. Finally, we should note that although the word **decoder** is traditionally used for this circuit, a more appropriate term is code converter. The term **decoder** is more appropriate for circuits that produce one-hot encoded outputs.

Did u know?

A seven-segment display (SSD), or seven-segment indicator, is a form of electronic display device for displaying decimal numerals that is an alternative to the more complex dot-matrix displays.

6.4 Comparators

A comparator is a device that compares two voltages or currents and switches its output to indicate which is larger.

Another useful type of arithmetic circuit compares the relative sizes of two binary numbers. Such a circuit is called a comparator. This considers the design of a comparator that has two n-bit inputs, A and B, which represent unsigned binary numbers. The comparator produces three outputs, called AeqB, AgtB, and AltB. The AeqB output is set to 1 if A and B are equal. The AgtB output is 1 if A is greater than B, and the AltB output is 1 if A is less than B. The desired comparator can be designed by creating a truth table that specifies the three outputs as functions of A and B. However, even for moderate values of n, the truth table is large. A better approach is to derive the comparator circuit by considering the bits of A and B in pairs. We can illustrate this by a small

Notes

example, where n = 4. Let A = $a_3a_2a_1a_0$ and B = $b_3b_2b_1b_0$. Define a set of intermediate signals called i_3, i_2, i_1, and i_0. Each signal, i_k, is 1 if the bits of A and B with the same index are equal.

That is, $i_k = \overline{a_k \oplus b_k}$. The comparator's AeqB output is then given by

$$AeqB = i_3 i_2 i_1 i_0$$

An expression for the AgtB output can be derived by considering the bits of A and B in the order from the most-significant bit to the least-significant bit. The first bit-position, k, at which a_k and b_k differ determines whether A is less than or greater than B. If $a_k = 0$ and bk = 1, then A < B. But if ak = 1 and bk = 0, then A > B. The AgtB output is defined by

$$AgtB = a_3\bar{b}_3 + i_3a_2\bar{b}_2 + i_3i_2a_1\bar{b}_1 + i_3i_2i_1a_0\bar{b}_0$$

The ik signals ensure that only the first digits, considered from the left to the right, of A and B that differ determine the value of AgtB.

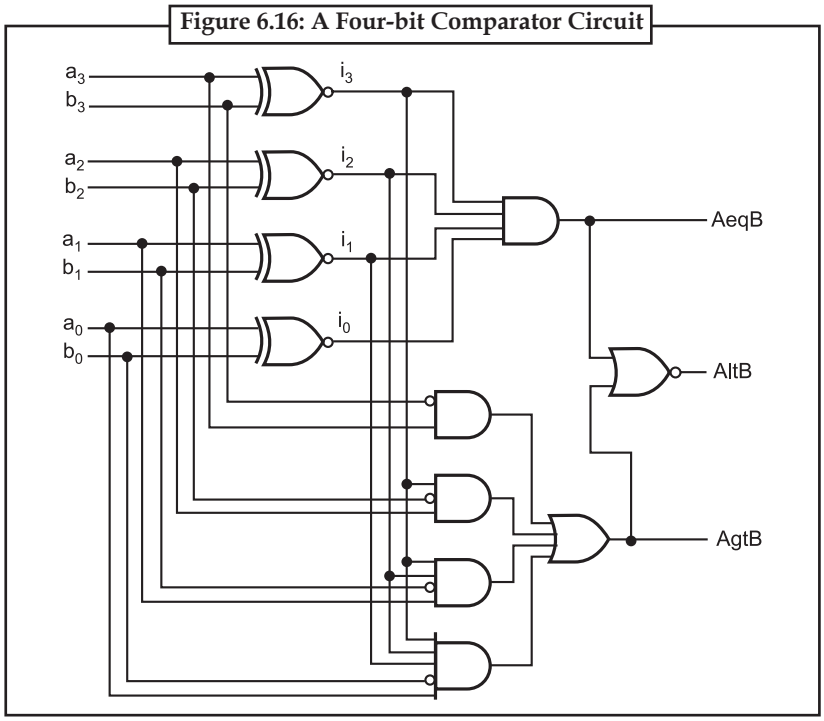
The AltB output can be derived by using the other two outputs as

$$AltB = \overline{AeqB + AgtB}$$

A logic circuit that implements the four-bit comparator circuit is shown in Figure 6.16. This approach can be used to design a comparator for any value of n. Comparator circuits, like most logic circuits, can be designed in different ways.

VHDLprovides several types of statements that can be used to assign logic values to signals. In the examples of VHDL code given so far, only simple assignment statements have been used, either for logic or arithmetic expressions. This introduces other types of assignment statements,

Figure 6.16: A Four-bit Comparator Circuit



which are called selected signal assignments, conditional signal assignments, generate statements, if-then-else statements, and case statements.

Notes

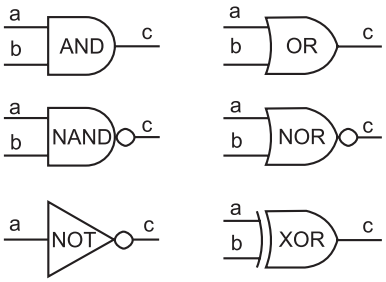
Boolean Logic Operation

As we all know' the microprocessor is the 'brain' of the modern digital computer, but have you ever wondered how a piece of silicon manages to process information so accurately and at such a high speed rate? The answer lies behind the basic building blocks of the processor, the logic gates, and a special type of information processing technique invented in the middle of the 1800s by George Boole, called Boolean logic.

Basically, Boolean logic operates only in the binary system by following a couple of simple rules. There are about seven simple logic gates that need to be studies in order to understand the full picture of how Boolean logic and computer microprocessors work by combining logical gates in a single electronic circuit that may contain several million transistors.

We will start with the simplest of all – the NOT gate, or the logical inverter (see bottom image, lower left corner). The NOT gate has one entrance and one exit and has the role of inverting logic bits. This basically means that when '1' logic bit is applied on the 'a' terminal, for example, the 'c' terminal must produce a'0 logic bit. The situation is reversed when '0' logic is applied on the 'a' terminal by forcing the 'c' terminal to output a '1' logic bit. Inversion is one of the basic operations in Boolean logic.

Another logic gate of critical importance is the AND gate (upper left corner), which practically designates an operation similar to multiplying. As you can see, the AND gate has two input terminals (a,b) and an output terminal (c). In fact the number of input terminals is unlimited. Boolean logic puts it very simple. If one of the input values is '0' logic, then the value of the other bits is irrelevant and the output will always produce a '0' logic. A '1' logic bit is present at the output terminal, only when all input bits are equal to '1' logic.



The OR gate, operates according to the logic that if one of the input terminals bears a '1' logic bit, then the 'c' terminal will always produce a '1'. The situation reverses when all of the input bits are '0' logic. The NAND (NOT AND) and NOR (NOT OR) gates can be imagined as an AND respectively an OR gate to whose output terminals a NOT gate (see image for symbols) is connected. Their logic is basically identical to that of AND and OR gates, but the output value is always reversed.

The last two significant logic gates are the XOR and XNOR gates, both of which can be constructed by using the basic logic gates (similar to the case of NAND and OR gates). XOR outputs a '1' logic only when a single input value equals '1' logic. In other cases the output produces a '0' bit. XNOR gates produce XOR reverse values.

By combining logic gates in logic ways a whole range of electronic devices can be constructed, starting with flip-flops, simple memories, counters, Random Access Memories all the way up to highly complex logical circuitry such as computer processors. You would be amazed as to how simple some of these devices are.

Contd...

Notes

Questions:

1. Explain the XOR and XNOR gates.
2. Explain the operations in Boolean logic gate

Self Assessment

Multiple choice questions:

5. How many outputs are on a BCD decoder?

   (*a*) 4 (*b*) 16

   (*c*) 8 (*d*) 10

6. Use the weighting factors to convert the following BCD numbers to binary.

   0101 0011 0010 0110 1000

   (*a*) 01010011 001001101000 (*b*) 11010100 100001100000

   (*c*) 110101 100001100 (*d*) 101011 001100001

7. What control signals may be necessary to operate a 1-line-to-16 line decoder?

   (*a*) flasher circuit control signal (*b*) a LOW on all gate enable inputs

   (*c*) input from a hexadecimal counter (*d*) a HIGH on all gate enable circuits

6.5 Summary

- Combinational circuit consists of input variables, logic gates and output variables. The logic gates accept signals from the inputs and generate the outputs which are different logic function combinations of the inputs.
- The logic gates accept signals from the inputs and generate the outputs which are different logic function combinations of the inputs.
- Multiplexers can also be used in a more general way to synthesize logic functions.
- In the demultiplexer the address inputs determine which data output is going to have the same value as the data input, the other data outputs will have the value 0.
- The demultiplexer is the inverse of the multiplexer, in that it takes a single data input and *n* address inputs.

6.6 Keywords

***Combinational logic*:** It is a type of digital logic which is implemented by Boolean circuits, where the output is a pure function of the present input only.

***Demultiplexer*:** This is the inverse of the multiplexer, in that it takes a single data input and *n* address inputs. It has 2^n outputs:

***Multiplexer*:** This circuit has a number of data inputs, one or more select inputs, and one output. It passes the signal value on one of the data inputs to the output.

***Shannon's expansion*:** Shannon's expansion or the Shannon decomposition is a method by which a Boolean function can be represented by the sum of two sub-functions of the original.

Notes

Lab Exercise

1. Draw the circuit diagram and truth table of Code Converters.
2. Prepare a truth table of four-bit comparator circuit.

6.7 Review Questions

1. Describe the multiplexer. And write 2 to 1 truth table.
2. Describe the demultiplexer.
3. Implement the function by using a 4-to-1 multiplexer and as few other gates as possible. Assume that only the uncomplemented inputs w1, w2, w3, and w4 are available.

$$f(w_1, w_2, w_3, w_4) = \bar{w}_1\bar{w}_2\bar{w}_4\bar{w}_5 + w_1w_2 + w_1w_3 + w_1w_4 + w_3w_4w_5$$

4. Describe the Shannon's expansion for multiplexer.
5. Explain and draw the circuit diagram a 4-to-1 multiplexer.
6. What do you understand by A 2 × 2 crossbar switch?
7. Define Claude Shannon truth table.
8. Explain the Boolean expression of 4 Channel Demultiplexer using Logic Gates.
9. Discuss the BCD-to-7-segment decoder.
10. Write the truth table of four-bit comparator circuit.

Answers to Self Assessment

1. (*c*) 2. (*d*) 3. (*a*) 4. (*d*) 5. (*c*)
6. (*d*) 7. (*b*)

6.8 Further Readings

Books

Digital Fundamentals – Prentice-Hall, Ninth Edition, by Floyd Thomas L.

Digital Electronics – Principles and Applications Tata McGraw Hill, 1999, by Tokheim R.L.

Online link

http://www.ariese.articlealley.com/combinational-circuit-design-165328.html

Unit 7: Standard Integrated Circuits (ICs)

Notes

CONTENTS

Objectives

After studying this unit, you will be able to:

- Discuss the resistor-transistor logic (RTL)
- Explain about transistor-transistor logic (TTL)
- Understand the diode-transistor logic (DTL)
- Understand about emitter-coupled logic (ECL)
- Explain about integrated circuit (IC)
- Discuss about high threshed logic (HTL)
- Explain about NMOS and CMOS

Notes

Introduction

An integrated circuit, commonly referred to as an IC, is a microscopic array of electronic circuits and components that has been diffused or implanted onto the surface of a single crystal, or chip, of semiconducting material such as silicon. It is called an integrated circuit because the components, circuits, and base material are all made together, or integrated, out of a single piece of silicon, as opposed to a discrete circuit in which the components are made separately from different materials and assembled later. ICs range in complexity from simple logic modules and amplifiers to complete microcomputers containing millions of elements.

The impact of integrated circuits on our lives has been enormous. ICs have become the principal components of almost all electronic devices. These miniature circuits have demonstrated low cost, high reliability, low power requirements, and high processing speeds compared to the vacuum tubes and transistors which preceded them. Integrated circuit microcomputers are now used as controllers in equipment such as machine tools, vehicle operating systems, and other applications where hydraulic, pneumatic, or mechanical controls were previously used. Because IC microcomputers are smaller and more versatile than previous control mechanisms, they allow the equipment to respond to a wider range of input and produce a wider range of output. They can also be reprogrammed without having to redesign the control circuitry. Integrated circuit microcomputers are so inexpensive that they are even found in children's electronic toys.

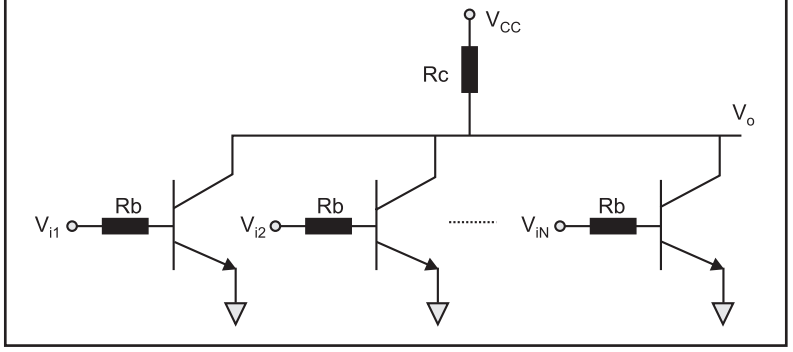
The first integrated circuits were created in the late 1950s in response to a demand from the military for miniaturized electronics to be used in missile control systems. At the time, transistors and printed circuit boards were the state-of-the-art electronic technology. Although transistors made many new electronic applications possible, engineers were still unable to make a small enough package for the large number of components and circuits required in complex devices like sophisticated control systems and hand-held programmable calculators. Several companies were in competition to produce a breakthrough in miniaturized electronics, and their development efforts were so close that there is some question as to which company actually produced the first IC. In fact, when the integrated circuit was finally patented in 1959, the patent was awarded jointly to two individuals working separately at two different companies.

7.1 Resistor-Transistor Logic (RTL)

Resistor-Transistor Logic, or RTL, refers to the obsolete technology for designing and fabricating digital circuits that employ logic gates consisting of nothing but transistors and resistors. RTL gates are now seldom used, if at all, in modern digital electronics design because it has several drawbacks, such as bulkiness, low speed, limited fan-out, and poor noise margin. A basic understanding of what RTL is, however, would be helpful to any engineer who wishes to get familiarized with TTL, which for the past many years has become widely used in digital devices such as logic gates, latches, gates, counters, and the like.

Figure 7.1 shows an example of an N-input RTL NOR gate. It consists of N transistors, whose collectors are all tied up to Vcc through a common resistor, and whose emitters are all grounded. Their bases individually act as inputs for input voltages Vi (i = 1, 2,..., N), which represent input logic levels. The output Vo is taken across the collector-resistor node and ground. Vo is only 'high' if the inputs to the bases of all the transistors are 'low'.

Figure 7.1: A Simple N-input RTL NOR Gate



Notes

One of the earliest gates used in integrated circuits is a special type of RTL gate known as the direct-coupled transistor logic (DCTL) gate. A DCTL gate is one wherein the bases of the transistors are connected directly to inputs without any base resistors. Thus, the RTL NOR gate shown in Figure 7.1 becomes a DCTL NOR gate if all the base resistors (Rb's) are eliminated. Without the base resistors, DCTL gates are more economical and simpler to fabricate onto integrated circuits than RTL gates with base resistors.

The main drawback of DCTL gates is that they suffer from a phenomenon known as current hogging. Ideally, several transistors that are connected in parallel will share the load current equally among themselves when they are all brought into saturation. In the real world, however, the saturation points of different transistors are attained with different levels of input voltages to the base (Vbe). As such, transistors that are in parallel and share the same input voltage (which are commonly encountered in DCTL circuits) do not share the load current evenly among themselves.

In fact, once the transistor with the lowest Vbesat saturates, the other transistors are prevented from saturating themselves. This causes the saturated transistor to 'hog' the load current, i.e. it carries the bulk of the load current whereas those transistors that were prevented from saturating carries a minimal portion of it. Current hogging, which prevented DCTL from becoming widely used, is largely avoided in RTL circuits simply by retaining the base resistors.

RTL gates also exhibit limited 'fan-outs'. The fan-out of a gate is the ability of its output to drive several other gates. The more gates it can drive, the higher is its fan-out. The fan-out of a gate is limited by the current that its output can supply to the gate inputs connected to it when the output is at logic '1', since at this state it must be able to drive the connected input transistors into saturation.

Another weakness of an RTL gate is its poor noise margin. The noise margin of a logic gate for logic level '0', $\Delta 0$, is defined as the difference between the maximum input voltage that it will recognize as a '0' (Vil) and the maximum voltage that may be applied to it as a '0' (Vol of the driving gate connected to it). For logic level '1', the noise margin $\Delta 1$ is the difference between the minimum input voltage that may be applied to it as a '1' (Voh of the driving gate connected to it) and the minimum input voltage that it will recognize as a '1' (Vih). Mathematically, $\Delta 0$ = Vil-Vol and $\Delta 1$ = Voh-Vih. Any noise that causes a noise margin to be overcome will result in a '0' being erroneously read as a '1' or vice versa. In other words, noise margin is a measure of the immunity of a gate from reading an input logic level incorrectly.

In an RTL circuit, the collector output of the driving transistor is directly connected to the base resistor of the driven transistor. Circuit analysis would easily show that in such an arrangement, the differences between Vil and Vol, and between Voh and Vih, are not that large. This is why RTL gates are known to have poor noise margins in comparison to DTL and TTL gates.

Did u know?

The first microcomputer chip, often called a microprocessor, was developed by Intel Corporation in 1969. It went into commercial production in 1971 as the Intel 4004.

Task Draw schematic diagram of N-input RTL NAND gate.

Notes

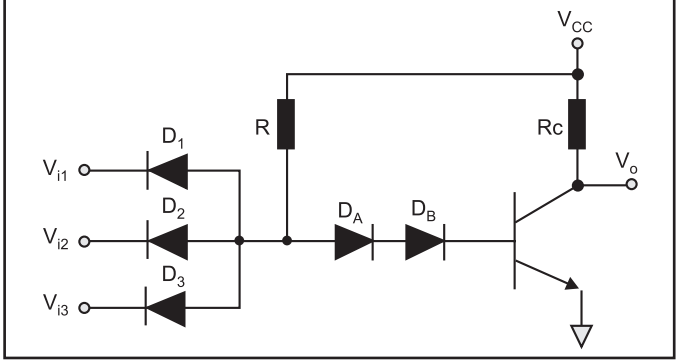
7.2 Diode-Transistor Logic (DTL)

Diode-Transistor Logic, or DTL, refers to the technology for designing and fabricating digital circuits wherein logic gates employ both diodes and transistors. DTL offers better noise margins and greater fan-outs than RTL, but suffers from low speed, especially in comparison to TTL.

The RTL allows the construction of NOR gates easily, but NAND gates are relatively more difficult to get from RTL. The DTL, however, allows the construction of simple NAND gates from a single transistor, with the help of several diodes and resistors.

It consists of a single transistor Q configured as an inverter, which is driven by a current that depends on the inputs to the three input diodes D1-D3.

Figure 7.2: A Simple 3-input DTL NAND Gate



In the NAND gate in Figure 7.2, the current through diodes DA and DB will only be large enough to drive the transistor into saturation and bring the output voltage Vo to logic '0' if all the input diodes D1-D3 are 'off', which is true when the inputs to all of them are logic '1'. This is because when D1-D3 are not conducting, all the current from Vcc through R will go through DA and DB and into the base of the transistor, turning it on and pulling Vo to near ground.

However, if any of the diodes D1-D3 gets an input voltage of logic '0', it gets forward-biased and starts conducting. This conducting diode 'shunts' almost all the current away from the reverse-biased DA and DB, limiting the transistor base current. This forces the transistor to turn off, bringing up the output voltage Vo to logic '1'.

One advantage of DTL over RTL is its better noise margin. The noise margin of a logic gate for logic level '0', Δ0, is defined as the difference between the maximum input voltage that it will recognize as a '0' (Vil) and the maximum voltage that may be applied to it as a '0' (Vol of the driving gate connected to it). For logic level '1', the noise margin Δ1 is the difference between the minimum input voltage that may be applied to it as a '1' (Voh of the driving gate connected to it) and the minimum input voltage that it will recognize as a '1' (Vih). Mathematically, Δ0 = Vil-Vol and Δ1 = Voh –Vih. Any noise that causes a noise margin to be overcome will result in a '0' being erroneously read as a '1' or vice versa. In other words, noise margin is a measure of the immunity of a gate from reading an input logic level incorrectly.

In a DTL circuit, the collector output of the driving transistor is separated from the base resistor of the driven transistor by several diodes. Circuit analysis would easily show that in such an arrangement, the differences between Vil and Vol, and between Voh and Vih, are much larger than those exhibited by RTL gates, wherein the collector of the driving transistor is directly connected to the base resistor of the driven transistor. This is why DTL gates are known to have better noise margins than RTL gates.

Notes

One problem that DTL does not solve is its low speed, especially when the transistor is being turned off. Turning off a saturated transistor in a DTL gate requires it to first pass through the active region before going into cut-off. Cut-off, however, will not be reached until the stored charge in its base has been removed. The dissipation of the base charge takes time if there is no available path from the base to ground. This is why some DTL circuits have a base resistor that is tied to ground, but even this requires some trade-offs. Another problem with turning off the DTL output transistor is the fact that the effective capacitance of the output needs to charge up through Rc before the output voltage rises to the final logic '1' level, which also consumes a relatively large amount of time. TTL, however, solves the speed problem of DTL elegantly.

Task Draw the circuit diagram of 3-input DTL using NOR gate.

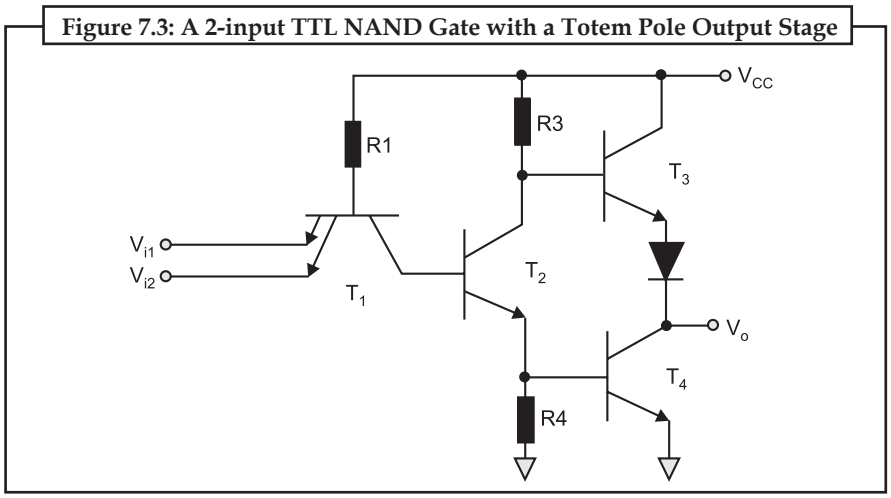
7.3 Transistor-Transistor Logic (TTL)

Transistor-Transistor Logic, or TTL, refers to the technology for designing and fabricating digital integrated circuits that employ logic gates consisting primarily of bipolar transistors. It overcomes the main problem associated with DTL, i.e. lack of speed.

The input to a TTL circuit is always through the emitter(s) of the input transistor, which exhibits a low input resistance. The base of the input transistor, on the other hand, is connected to the Vcc line, which causes the input transistor to pass a current of about 1.6 mA when the input voltage to the emitter(s) is logic '0', i.e. near ground. Letting a TTL input 'float' (left unconnected) will usually make it go to logic '1', but such a state is vulnerable to stray signals, which is why it is good practice to connect TTL inputs to Vcc using 1 kohm pull-up resistors.

The most basic TTL circuit has a single output transistor configured as an inverter with its emitter grounded and its collector tied to Vcc with a pull-up resistor, and with the output taken from its collector. Most TTL circuits, however, use a totem pole output circuit, which replaces the pull-up resistor with a Vcc-side transistor sitting on top of the GND-side output transistor. The emitter of the Vcc-side transistor (whose collector is tied to Vcc) is connected to the collector of the GND-side transistor (whose emitter is grounded) by a diode. The output is taken from the collector of the GND-side transistor. Figure 7.3 shows a basic 2-input TTL NAND gate with a totem-pole output.

Figure 7.3: A 2-input TTL NAND Gate with a Totem Pole Output Stage



Notes

In the TTL NAND gate of Figure 7.3, applying a logic '1' input voltage to both emitter inputs of T1 reverse-biases both base-emitter junctions, causing current to flow through R1 into the base of T2, which is driven into saturation. When T2 starts conducting, the stored base charge of T3 dissipates through the T2 collector, driving T3 into cut-off. On the other hand, current flows into the base of T4, causing it to saturate and pull down the output voltage Vo to logic '0', or near ground. Also, since T3 is in cut-off, no current will flow from Vcc to the output, keeping it at logic '0'. Note that T2 always provides complementary inputs to the bases of T3 and T4, such that T3 and T4 always operate in opposite regions, except during momentary transition between regions.

On the other hand, applying a logic '0' input voltage to at least one emitter input of T1 will forward-bias the corresponding base-emitter junction, causing current to flow out of that emitter. This causes the stored base charge of T2 to discharge through T1, driving T2 into cut-off. Now that T2 is in cut-off, current from Vcc will be diverted to the base of T3 through R3, causing T3 to saturate. On the other hand, the base of T4 will be deprived of current, causing T to go into cut-off. With T4 in cut-off and T3 in saturation, the output Vo is pulled up to logic '1', or closer to Vcc.

Outputs of different TTL gates that employ the totem-pole configuration must not be connected together since differences in their output logic will cause large currents to flow from the logic '1' output to the logic '0' output, destroying both output stages. The output of a typical TTL gate under normal operation can sink currents of up to 16 mA.

The noise margin of a logic gate for logic level '0', $\Delta 0$, is defined as the difference between the maximum input voltage that it will recognize as a '0' (Vil) and the maximum voltage that may be applied to it as a '0' (Vol of the gate driving it). For logic level '1', the noise margin $\Delta 1$ is the difference between the minimum input voltage that may be applied to it as a '1' (Voh of the gate driving it) and the minimum input voltage that it will recognize as a '1' (Vih). Mathematically, $\Delta 0$ = Vil-Vol and $\Delta 1$ = Voh-Vih. Any noise that causes a noise margin to be overcome will result in a '0' being erroneously read as a '1' or vice versa. In other words, noise margin is a measure of the immunity of a gate from reading an input logic level incorrectly. For TTL, Vil = 0.8V and Vol = 0.4V, so $\Delta 0$ = 0.4V, and Voh = 2.4V and Vih = 2.0 V, so $\Delta 1$ = 0.4V. These noise margins are not as good as the noise margins exhibited by DTL.

As mentioned earlier, TTL has a much higher speed than DTL. This is due to the fact that when the output transistor (T4 in Figure 7.3) is turned off, there is a path for the stored charge in its base to dissipate through, allowing it to reach cut-off faster than a DTL output transistor. At the same time, the equivalent capacitance of the output is charged from Vcc through T3 and the output diode, allowing the output voltage to rise more quickly to logic '1' than in a DTL output wherein the output capacitance is charged through a resistor.

The commercial names of digital IC's that employ TTL start with '74', e.g. 7400, 74244, etc. Most TTL devices nowadays, however, are named '74LSXXX', with the 'LS' standing for 'low power Schottky'. Low power Schottky TTL devices employ a Schottky diode, which is used to limit the voltage between the collector and the base of a transistor, making it possible to design TTL gates that use significantly less power to operate while allowing higher switching speeds.

Notes

7.3.1 Test Parameters for TTL Devices

Table 7.1: Test Parameter

Test Parameter	Unit	Typical Description
Logic High Input Voltage, Vih(Logic "1")	V	This is the minimum voltage that an input of a TTL digital IC is guaranteed to recognize as a Logic "1". Standard Spec: 2.0 V min.
Logic Low Input Voltage, Vil (Logic "0")	V	This is the maximum voltage that an input of a TTL digital IC is guaranteed to recognize as a Logic "0". Standard Spec: 0.8 V max.
Logic High Output Voltage, Voh (Logic "1")	V	This is the minimum voltage that an output of a TTL digital IC is guaranteed to deliver as a Logic "1". Standard Spec: 2.4 V min.
Logic Low Output Voltage, Vol (Logic "0")	V	This is the maximum voltage that an output of a TTL digital IC is guaranteed to deliver as a Logic "0". Standard Spec: 0.4 V max.
Logic High Input Current, Iih	µA	This is the minimum amount of current needed by an input of a TTL digital IC to stay at Logic "1". Example of an Actual Spec: 50 µA min. when Vcc=Max; Vin = 2.7V
Logic Low Input Current, Iil	mA	This is the maximum amount of current that the input of a TTL digital IC can sink to stay at Logic "0". Example of an Actual Spec: -1.6 mA max. when Vcc=max; Vin = 0.5V
Output Short Circuit Current, Ios	mA	This is the amount of current that a short-circuited TTL digital output exhibits or can handle. Example of an Actual Spec: -40 mA min., -65 mA typ., -100 mA max. when Vcc=max; Vout = 0V
Supply Current, Icc	mA	This is the Vcc supply current exhibited by a TTL IC. Example of an Actual Spec: 30 mA min., 50 mA max. when Vcc=max

Notes

7.4 Emitter-Coupled Logic (ECL)

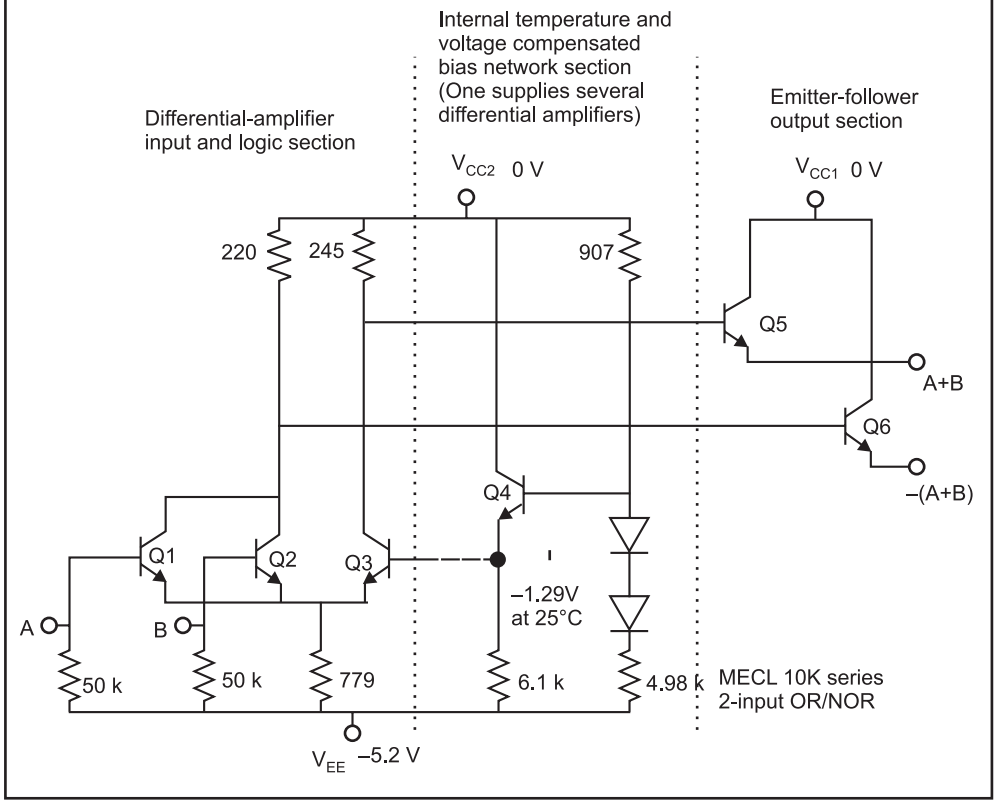
In electronics, emitter-coupled logic (ECL) is a logic family that achieves high speed by using an overdriven BJT differential amplifier with single-ended input whose emitter current is limited to avoid the slow saturation region of transistor operation. As the current is steered between two legs of an emitter-coupled pair, ECL is sometimes called current-steering logic (CSL), current-mode logic (CML) or current-switch emitter-follower (CSEF) logic.

In ECL, the transistors are never in saturation, the input/output voltages have a small swing (0.8 V), the input impedance is high and the output resistance is low; as a result, the transistors change states quickly, gate delays are low, and the fan-out capability is high. In addition, the essentially-constant current draw of the differential amplifiers minimizes delays and glitches due to supply-line inductance and capacitance, and the complementary outputs decrease the propagation time of the whole circuit by saving additional inverters.

ECL's major disadvantage is that each gate continuously draws current, which means it requires (and dissipates) significantly more power than those of other logic families, especially when quiescent.

The equivalent of emitter-coupled logic made out of FETs is called source-coupled FET logic (SCFL).

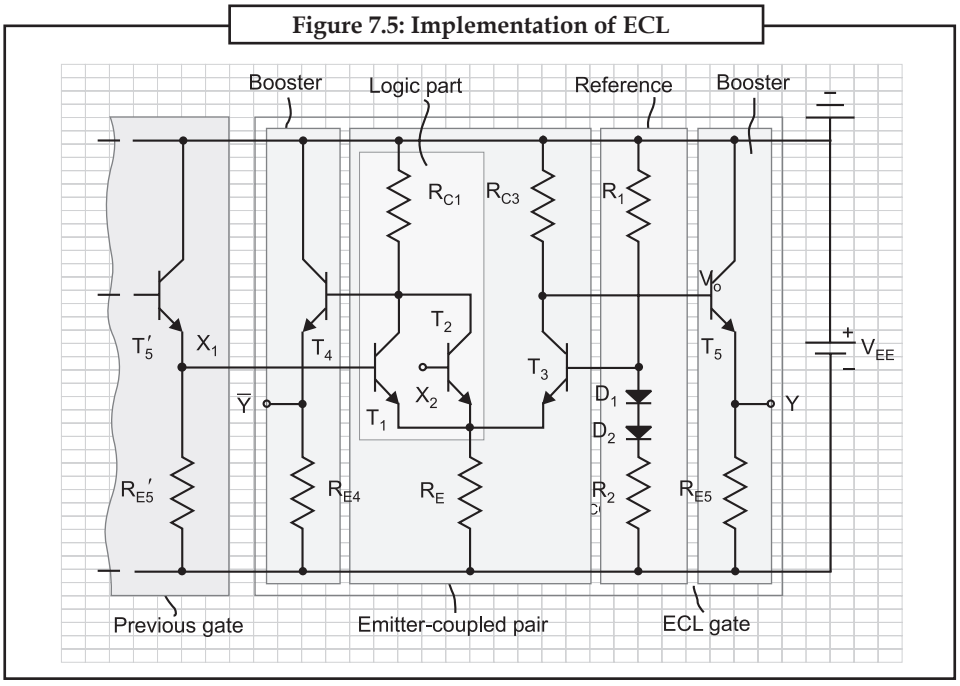
Figure 7.4: Emitter-coupled Logic



Notes

7.4.1 Implementation

Figure 7.5: Implementation of ECL



ECL is based on an emitter-coupled (long-tailed) pair, shaded red in the figure on the right. The left half of the pair (shaded yellow) consists of two parallel-connected input transistors T1 and T2 (an exemplary two-input gate is considered) implementing NOR logic. The base voltage of the right transistor T3 is held fixed by a reference voltage source, shaded light green: the voltage divider with a diode thermal compensation (R1, R2, D1 and D2) and sometimes a buffering emitter follower (not shown on the picture); thus the emitter voltages are kept relatively steady. As a result, the common emitter resistor R_E acts nearly as a current source. The output voltages at the collector load resistors R_{C1} and R_{C3} are shifted and buffered to the inverting and non-inverting outputs by the emitter followers T4 and T5 (shaded blue). The output emitter resistors R_{E4} and R_{E5} do not exist in all versions of ECL. In some cases 50 Ω line termination resistors connected between the bases of the input transistors and −2V act as emitter resistors.

7.4.2 Operation

The ECL circuit operation is considered below with assumption that the input voltage is applied to T1 base, while T2 input is unused or a logical "0" is applied.

During the Transition

The core of the circuit – the emitter-coupled pair (T1 and T3) – acts as a differential amplifier with single-ended input. The "long-tail" current source (R_E) sets the total current flowing through the two legs of the pair. The input voltage controls the current flowing through the transistors by sharing it between the two legs, steering it all to one side when not near the switching point. The gain is higher than at the end states (see below) and the circuit switches quickly.

***Low Input Voltage* (*logical "0"*) *or At High Input Voltage* (*logical "1"*)**

The differential amplifier is overdriven. The one transistor (T1 or T3) is cut-off and the other (T3 or T1) is in active linear region acting as a common-emitter stage with emitter degeneration that takes all the current, starving the other cut-off transistor.

The active transistor is loaded with the relatively high emitter resistance R_E that introduces a significant negative feedback (emitter degeneration). To prevent saturation of the active transistor

Notes

so that the diffusion time that slows the recovery from saturation will not be involved in the logic delay,[2] the emitter and collector resistances are chosen such that at maximum input voltage some voltage is left across the transistor. The residual gain is low ($K=R_C/R_E<1$). The circuit is insensitive to the input voltage variations and the transistor stays firmly in active linear region. The input resistance is high because of the series negative feedback.

The cut-off transistor breaks the connection between its input and output. As a result, its input voltage does not affect the output voltage. The input resistance is high again since the base-emitter junction is cut-off.

Positive emitter-coupled logic (PECL) is a further development of ECL using a positive 5V supply instead of a negative 5V supply. Low-voltage positive emitter-coupled logic (LVPECL) is a power optimized version of PECL, using a positive 3.3V instead of 5V supply. PECL and LVPECL are differential signalling systems, and are mainly used in high speed and clock distribution circuits.


Caution

Use of the IC in excess of absolute maximum rating such as the applied voltage or operating temperature range may result in IC damage.

7.5 Integrated Circuit (IC)

In an integrated circuit, electronic components such as resistors, capacitors, diodes, and transistors are formed directly onto the surface of a silicon crystal. The process of manufacturing an integrated circuit will make more sense if one first understands some of the basics of how these components are formed.

7.5.1 Design

Some integrated circuits can be considered standard, off-the-shelf items. Once designed, there is no further design work required. Examples of standard ICs would include voltage regulators, amplifiers, analog switches, and analog-to-digital or digital-to-analog converters. These ICs are usually sold to other companies who incorporate them into printed circuit boards for various electronic products.

Other integrated circuits are unique and require extensive design work. An example would be a new microprocessor for computers. This design work may require research and development of new materials and new manufacturing techniques to achieve the final design.

7.5.2 The Manufacturing Process

Hundreds of integrated circuits are made at the same time on a single, thin slice of silicon and are then cut apart into individual IC chips. The manufacturing process takes place in a tightly controlled environment known as a clean room where the air is filtered to remove foreign particles. The few equipment operators in the room wear lint-free garments, gloves, and coverings for their heads and feet. Since some IC components are sensitive to certain frequencies of light, even the light sources are filtered. Although manufacturing processes may vary depending on the integrated circuit being made, the following process is typical.

Preparing the Silicon Wafer

- A cylindrical ingot of silicon about 1.5 to 4.0 inches (3.8 to 10.2 cm) in diameter is held vertically inside a vacuum chamber with a high-temperature heating coil encircling it. Starting at the top of the cylinder, the silicon is heated to its melting point of about 2550°F (1400°C). To avoid contamination, the heated region is contained only by the surface tension of the molten silicon. As the region melts, any impurities in the silicon become mobile. The heating coil is slowly moved down the length of the cylinder, and the impurities are carried along with the melted region. When the heating coil reaches the bottom, almost all of the

Notes

impurities have been swept along and are concentrated there. The bottom is then sliced off, leaving a cylindrical ingot of purified silicon.

- A thin, round wafer of silicon is cut off the ingot using a precise cutting machine called a wafer slicer. Each slice is about 0.01 to 0.025 inches (0.004 to 0.01 cm) thick. The surface on which the integrated circuits are to be formed is polished.
- The surfaces of the wafer are coated with a layer of silicon dioxide to form an insulating base and to prevent any oxidation of the silicon which would cause impurities. The silicon dioxide is formed by subjecting the wafer to superheated steam at about 1830°F (1000°C) under several atmospheres of pressure to allow the oxygen in the water vapour to react with the silicon. Controlling the temperature and length of exposure controls the thickness of the silicon dioxide layer.

Masking

- The complex and interconnected design of the circuits and components is prepared in a process similar to that used to make printed circuit boards. For ICs, however, the dimensions are much smaller and there are many layers superimposed on top of each other. The design of each layer is prepared on a computer-aided drafting machine, and the image is made into a mask which will be optically reduced and transferred to the surface of the wafer. The mask is opaque in certain areas and clear in others. It has the images for all of the several hundred integrated circuits to be formed on the wafer.
- A drop of photo resist material is placed in the centre of the silicon wafer, and the wafer is spun rapidly to distribute the photo resist over the entire surface. The photo resist is then baked to remove the solvent.
- The coated wafer is then placed under the first layer mask and irradiated with light. Because the spaces between circuits and components are so small, ultraviolet light with a very short wavelength is used to squeeze through the tiny clear areas on the mask. Beams of electrons or x-rays are also sometimes used to irradiate the photo resist.
- The mask is removed and portions of the photo resist are dissolved. If a positive photo resist was used, then the areas that were irradiated will be dissolved. If a negative photo resist was used, then the areas that were irradiated will remain. The uncovered areas are then either chemically etched to open up a layer or is subjected to chemical doping to create a layer of P or N regions.

Doping—Atomic Diffusion

- One method of adding dopants to create a layer of P or N regions is atomic diffusion. In this method a batch of wafers is placed in an oven made of a quartz tube surrounded by a heating element. The wafers are heated to an operating temperature of about 1500-2200°F (816-1205°C), and the dopant chemical is carried in on an inert gas. As the dopant and gas pass over the wafers, the dopant is deposited on the hot surfaces left exposed by the masking process. This method is good for doping relatively large areas, but is not accurate for smaller areas. There are also some problems with the repeated use of high temperatures as successive layers are added.

Doping—Ion Implantation

- The second method to add dopants is ion implantation. In this method a dopant gas, like phosphine or boron trichloride, is ionized to provide a beam of high-energy dopant ions which are fired at specific regions of the wafer. The ions penetrate the wafer and remain implanted. The depth of penetration can be controlled by altering the beam energy, and the amount of dopant can be controlled by altering the beam current and time of exposure. Schematically, the whole process resembles firing a beam in a bent cathode-ray tube. This

Notes

method is so precise, it does not require masking—it just points and shoots the dopant where it is needed. However, it is much slower than the atomic diffusion process.

Making Successive Layers

- The process of masking and etching or doping is repeated for each successive layer depending on the doping process used until all of the integrated circuit chips are complete. Sometimes a layer of silicon dioxide is laid down to provide an insulator between layers or components. This is done through a process known as chemical vapour deposition, in which the wafer's surface is heated to about 752°F (400°C), and a reaction between the gases silane and oxygen deposits a layer of silicon dioxide. A final silicon dioxide layer seals the surface, a final etching opens up contact points, and a layer of aluminum is deposited to make the contact pads. At this point, the individual ICs are tested for electrical function.

Making Individual ICs

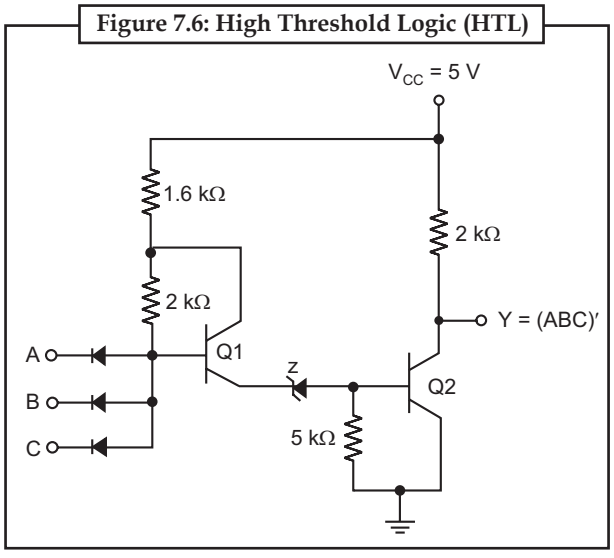
- The thin wafer is like a piece of glass. The hundreds of individual chips are separated by scoring a crosshatch of lines with a fine diamond cutter and then putting the wafer under stress to cause each chip to separate. Those ICs that failed the electrical test are discarded. Inspection under a microscope reveals other ICs that were damaged by the separation process, and these are also discarded.
- The good ICs are individually bonded into their mounting package and the thin wire leads are connected by either ultrasonic bonding or thermo compression. The mounting package is marked with identifying part numbers and other information.
- The completed integrated circuits are sealed in anti-static plastic bags to be stored or shipped to the end-user.

7.6 High Threshold Logic (HTL)

The HTL is a variant of DTL which is used in such environments where noise is very high.

The threshold values at the input to a logic gate determine whether a particular input is interpreted as a logic 0 or a logic 1 (e.g. anything less than 1 V is a logic 0 and anything above 3 V is a logic 1. In this example, the threshold values are 1V and 3V). HTL incorporates Zener diodes to create a large offset between logic 1 and logic 0 voltage levels. These devices usually ran off a 15 V power supply and were found in industrial control, where the high differential was intended to minimize the effect of noise.

Figure 7.6: High Threshold Logic (HTL)



Notes

7.6.1 Advantages

- Increased Noise Margin
- Spike Control
- High Noise Threshold Value

7.6.2 Disadvantages

Slow speed due to increased supply voltage resulting in use of high value resistors.


Caution

The CMOS settings directly affect the operation of the motherboard. If they are incorrectly set, the machine may not function very well or may fail to boot altogether.

Self Assessment

True or False:

1. The DCTL gate is suffering from a phenomenon known as current hogging.

   (*a*) True (*b*) False

2. The first integrated circuits were created in the late 1948s.

   (*a*) True (*b*) False

3. The high input currents required by a CMOS circuit results in high power consumption.

   (*a*) True (*b*) False

4. CMOS circuits were invented by Frank Wanlass of Fairchild Semiconductor in 1963.

   (*a*) True (*b*) False

5. CMOS chips work with a broader range of power supply voltages.

   (*a*) True (*b*) False

7.7 NMOS and CMOS Logic Gates

The term '*Complementary Metal-Oxide-Semiconductor*', or simply 'CMOS', refers to the device technology for designing and fabricating integrated circuits that employ logic using both n- and p-channel MOSFET's. CMOS is the other major technology utilized in manufacturing digital IC's aside from TTL, and is now widely used in microprocessors, memories, and digital ASIC's.

The input to a CMOS circuit is always to the gate of the input MOS transistor, which exhibits a very high resistance. This high gate resistance is due to the fact that the gate of a MOS transistor is isolated from its channel by an oxide layer, which is a dielectric. As such, the current flowing through a CMOS input is virtually zero, and the device is operated mainly by the voltage applied to the gate, which controls the conductivity of the device channel.

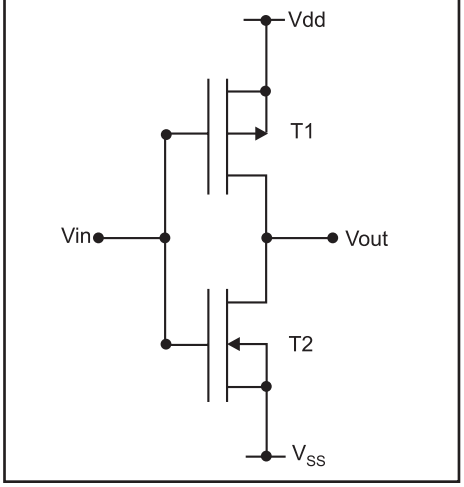
The low input currents required by a CMOS circuit results in lower power consumption, which is the major advantage of CMOS over TTL. In fact, power consumption in a CMOS circuit occurs only when it is switching between logic levels. This power dissipation during a switching action is known as 'dynamic power'. In a typical CMOS IC, output switching may take about a hundred picoseconds, and may occur every 10 nanoseconds (or 100 million times per second). Switching an output from one logic level to another requires the charging and discharging of various load capacitances, which dissipates power that is proportional to these capacitances and the frequency of switching.

Figure 7.7 shows an example of a CMOS circuit – an inverter that employs a p-channel and an n-channel MOS transistor. A logic '1' Vin voltage at the input would make T1 (p-channel) turn

Notes

off and T2 (n-channel) turn on, pulling Vout to near Vss, or logic '0'. A logic '0' Vin voltage, on the other hand, will make T1 turn on and T2 turn off, pulling Vout to near Vdd, or logic '1'. Note that the p- and n-channel MOS transistors in the circuit are complementary, so they are always in opposite states, i.e. for any given Vin level, one of them is 'on' while the other is 'off'.

Figure 7.7: A CMOS Inverter



CMOS circuits were invented by Frank Wanlass of Fairchild Semiconductor in 1963, although the first CMOS IC's were not produced until 1968, this time at RCA. The original CMOS devices consumed less power than TTL but ran slower too, so early applications centred on circuits where battery consumption was more important than speed of operation. Steadily CMOS technology has improved, subsequently becoming the technology of choice for digital circuits. Aside from low power consumption, CMOS circuits are also easy and cheap to fabricate, allowing denser circuit integration than their bipolar counterparts.

CMOS circuits are quite vulnerable to electrostatic discharge (ESD) damage, mainly by gate oxide punch through from high ESD voltages. Because of this issue, modern CMOS IC's are now equipped with on-chip ESD protection circuits, which reduce (but not totally eliminate) risks of ESD damage. Proper handling and processing of CMOS IC's to prevent ESD damage are also a 'must'.

In the 70's and 80's, CMOS IC's were run using digital voltages that were compatible with TTL so both could be inter-operated with each other. By the 1990's, however, the need for much lower power consumption for mobile devices had resulted in the deployment of more and more CMOS devices that run on much lower power supply voltages. The lower operating voltages also allowed the use of thinner, higher-performance gate dielectrics in CMOS IC's.

7.7.1 Overview of CMOS

- CMOS is also sometimes explained as complementary-symmetry metal–oxide–semiconductor. The words "*complementary-symmetry*" refer to the fact that the typical digital design style with CMOS uses complementary and symmetrical pairs of p-type and n-type MOSFETs for logic functions.
- Semiconductors are made of silicon and germanium, materials which "sort of" conduct electricity, but not enthusiastically. Areas of these materials that are "doped" by adding impurities become full-scale conductors of either extra electrons with a negative charge (N-type transistors) or of positive charge carriers (P-type transistors).
- In CMOS technology, both kinds of transistors are used in a complementary way to form a current gate that forms an effective means of electrical control.
- CMOS transistors use almost no power when not needed. As the current direction changes more rapidly, however, the transistors become hot. This characteristic tends to limit the speed at which microprocessors can operate.

Notes

7.7.2 Important Features of CMOS

- The main advantage of CMOS over NMOS and bipolar technology is the much smaller power dissipation. Unlike NMOS or bipolar circuits, a CMOS circuit has almost no static power dissipation. Power is only dissipated in case the circuit actually switches.
- When the gate switches states, current is drawn from the power supply to charge the stray capacitance at the output of the gate. This means that current draw of CMOS devices increases with clock speed.
- Because the logic thresholds of CMOS were proportional to the power supply voltage, CMOS devices were well-adapted to battery-operated systems with simple power supplies.
- CMOS gates can also tolerate much wider voltage ranges than TTL gates because the logic thresholds are (approximately) proportional to power supply voltage, and not the fixed levels required by bipolar circuits.
- CMOS chips work with a broader range of power supply voltages. While TTL ICs all require a power supply voltage of 5V (+/- 0.5V), CMOS works with a wider range of power supply voltage – usually anywhere from 3 to 15V.

7.7.3 Test Parameters of CMOS

Table 7.2: Test Parameters of CMOS

Test Parameter	Unit	Typical Description
Logic High Input Voltage, Vih (Logic "1")	V	This is the minimum voltage that an input of a CMOS digital IC is guaranteed to recognize as a Logic "1". Typical Spec: 3.5 V min. for Vss = 5V
Logic Low Input Voltage, Vil (Logic "0")	V	This is the maximum voltage that an input of a CMOS digital IC is guaranteed to recognize as a Logic "0". Typical Spec: 1.5 V max. for Vss = 5V
Logic High Output Voltage, Voh(Logic "1")	V	This is the minimum voltage that an output of a CMOS digital IC is guaranteed to deliver as a Logic "1". Typical Spec: 4.99 V min. for Vss = 5V
Logic Low Output Voltage, Vol (Logic "0")	V	This is the maximum voltage that an output of a CMOS digital IC is guaranteed to deliver as a Logic "0". Typical Spec: 0.01 V max. for Vss = 5V
Logic High Input Current, Iih	pA	This is the minimum amount of current needed by an input of a CMOS digital IC to stay at Logic "1". Example of an Actual Spec: +10 pA min. when Vss = 5V; Vin = 3.5V
Logic Low Input Current, Iil	pA	This is the maximum amount of current that the input of a CMOS digital IC can sink to stay at Logic "0". Example of an Actual Spec: –10 pA max. when Vss = 5V; Vin = 1.5V

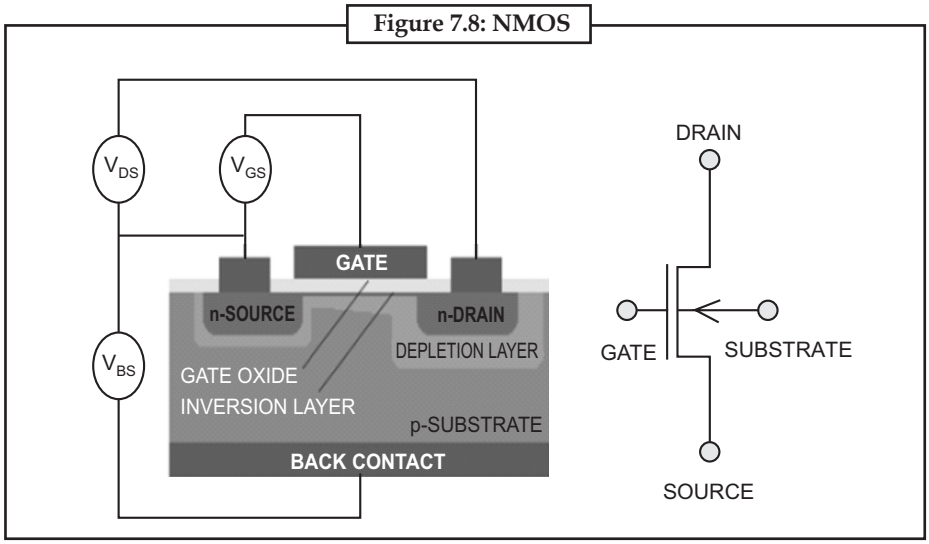
Notes

Logic High Output Current, Ioh	mA	This is the maximum amount of current that an output of a CMOS digital IC can source while at Logic "1". Example of an Actual Spec: –0.5 mA max. when Vss = 5V; Vin = 3.5V
Logic Low Output Current, Iol	mA	This is the maximum amount of current that an output of a CMOS digital IC can sink while at Logic "0". Example of an Actual Spec: +0.4 mA max. when Vss = 5V; Vin = 1.5V

The n-type Metal-Oxide-Semiconductor Field-Effect-Transistor (MOSFET) consists of a source and a drain, two highly conducting n-type semiconductor regions which are isolated from the p-type substrate by reversed-biased p-n diodes. A metal (or poly-crystalline) gate covers the region between source and drain, but is separated from the semiconductor by the gate oxide. The basic structure of an n-type MOSFET and the corresponding circuit symbol are shown in the Figure 7.8.

As can be seen on the figure, the source and drain regions are identical. It is the applied voltages which determine which n-type region provides the electrons and becomes the source, while the other n-type region collects the electrons and becomes the drain. The voltages applied to the drain and gate electrode as well as to the substrate by means of a back contact are referred to the source potential, as also indicated on the Figure 7.8.

Figure 7.8: NMOS

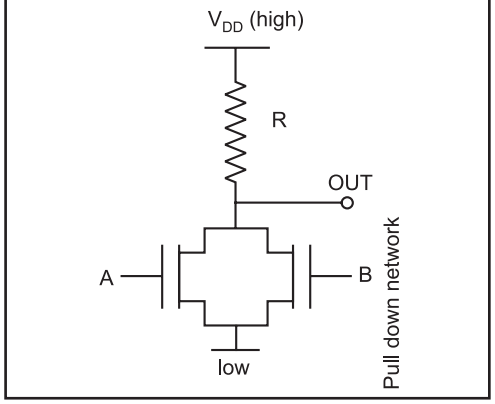


NMOS transistors have four modes of operation: cut-off (or sub-threshold), triode, saturation (sometimes called active), and velocity saturation.

The n-type MOSFETs are arranged in a so-called "pull-down network" (PDN) between the logic gate output and negative supply voltage, while a resistor is placed between the logic gate output and the positive supply voltage. The circuit is designed such that if the desired output is low, then the PDN will be active, creating a current path between the negative supply and the output.

Notes

Figure 7.9: Pull-down Network (PDN)



As an example, here is a NOR gate in NMOS logic. If either input A or input B is high (logic 1, = True), the respective MOS transistor acts as a very low resistance between the output and the negative supply, forcing the output to be low (logic 0, = False). When both A and B are high, both transistors are conductive, creating an even lower resistance path to ground. The only case where the output is high is when both transistors are off, which occurs only when both A and B are low, thus satisfying the truth table of a NOR gate:

Table 7.3: Truth Table of NOR Gate

A	B	A NOR B
0	0	1
0	1	0
1	0	0
1	1	0

A MOSFET can be made to operate as a resistor, so the whole circuit can be made with n-channel MOSFETs only. For many years, this made NMOS circuits much faster than comparable PMOS and CMOS circuits, which had to use much slower p-channel transistors. It was also easier to manufacture NMOS than CMOS, as the latter has to implement p-channel transistors in special n-wells on the p-substrate. The major problem with NMOS (and most other logic families) is that a DC current must flow through a logic gate even when the output is in a steady state (low in the case of NMOS). This means static power dissipation, i.e. power drain even when the circuit is not switching. This is a similar situation to the modern high speed, high density CMOS circuits (microprocessors etc.) which also has significant static current draw, although this is due to leakage, not bias. However, older and/or slower static CMOS circuits used for ASICs, SRAM etc., typically have very low static power consumption.

Also, NMOS circuits are slow to transition from low to high. When transitioning from high to low, the transistors provide low resistance, and the capacitative charge at the output drains away very quickly (similar to discharging a capacitor through a very low resistor). But the resistance between the output and the positive supply rail is much greater, so the low to high transition takes longer (similar to charging a capacitor through a high value resistor). Using a resistor of lower value will speed up the process but also increases static power dissipation. However, a better (and the most common) way to make the gates faster is to use depletion-mode transistors instead of enhancement-mode transistors as loads. This is called depletion-load NMOS logic.

7.7.4 Important Features of NMOS

- N-MOS is 1/3 of the size of the P-MOS.
- N-MOS is much faster than P-MOS.

Notes

- The present invention facilitates semiconductor fabrication by employing high-k dielectric layers in NMOS regions but not in PMOS regions of semiconductor transistor devices.
- NMOS devices within an I/O region having higher operation voltage requirements employ a high-k dielectric layer and a first oxide layer as dielectric layers. NMOS devices within a core region of the device employ only the high-k dielectric layer as a dielectric layer. As a result, high-k dielectric materials are employed for the NMOS devices without negatively impacting operation of the PMOS devices.

7.7.5 CMOS versus TTL

CMOS

The term 'Complementary Metal-Oxide-Semiconductor', or simply 'CMOS', refers to the device technology for designing and fabricating integrated circuits that employ logic using both n- and p-channel MOSFET's. CMOS is the other major technology utilized in manufacturing digital IC's aside from TTL, and is now widely used in microprocessors, memories, and digital ASIC's.

TTL

Transistor-Transistor Logic, or TTL, refers to the technology for designing and fabricating digital integrated circuits that employ logic gates consisting primarily of bipolar transistors. It overcomes the main problem associated with DTL, i.e. lack of speed.

Comparisons between CMOS and TTL are given below:

- The input to a CMOS circuit is always to the gate of the input MOS transistor, which exhibits a very high resistance. This high gate resistance is due to the fact that the gate of a MOS transistor is isolated from its channel by an oxide layer, which is a dielectric. As such, the current flowing through a CMOS input is virtually zero, and the device is operated mainly by the voltage applied to the gate, which controls the conductivity of the device channel.
- The input to a TTL circuit is always through the emitter(s) of the input transistor, which exhibits a low input resistance. The base of the input transistor, on the other hand, is connected to the Vcc line.
- The low input currents required by a CMOS circuit results in lower power consumption, which is the major advantage of CMOS over TTL. In fact, power consumption in a CMOS circuit occurs only when it is switching between logic levels. This power dissipation during a switching action is known as 'dynamic power'.
- In a typical CMOS IC, output switching may take about a hundred picoseconds, and may occur every 10 nanoseconds (or 100 million times per second). Switching an output from one logic level to another requires the charging and discharging of various load capacitances, which dissipates power that is proportional to these capacitances and the frequency of switching.
- TTL devices can drive more power into a load than CMOS. TTL is less sensitive to static-discharge failure and less expensive. It is also faster.
- Most TTL circuits use a totem pole output circuit, which replaces the pull-up resistor with a Vcc-side transistor sitting on top of the GND side output transistor. The emitter of the Vcc-side transistor (whose collector is tied to Vcc) is connected to the collector of the GND-side transistor (whose emitter is grounded) by a diode. The output is taken from the collector of the GND-side transistor.
- TTL uses a different type of transistor (bipolar) whereas CMOS uses MOSFETs. FETs do not have current flowing into the "gate" when static, whereas bipolar do (into the "base"), so generally this allows them to operate with lower power especially in a stand-by type of state.

Contd....

Notes

- CMOS logic has advantage of having smaller dimensions with new coming (usually smaller) technologies. Thus, the RC constant (RC defines transition time, where R = resistance is coming from R-ON resistance and C = capacitance is coming from gate capacitance) is smaller. Smaller RC constant means shorter transition time. Logic is faster and can do more in same time.
- CMOS devices do not produce as much waste heat as transistor-transistor logic (TTL). CMOS also allows a high density of logic functions on a chip. It has the characteristics of high noise immunity.
- One of the benefits of using TTL circuits over CMOS circuits is the fact that they are not easily damaged by static unlike CMOS devices.
- Due to the output structure of TTL devices, the output impedance is asymmetrical between the high and low state, making them unsuitable for driving transmission lines. This is usually solved by buffering the outputs with special line driver devices where signals need to be sent through cables.


Did u know?

Conventional CMOS devices work over a range of −55°C to +125°C. There were theoretical indications as early as August 2008 that silicon CMOS will work down to −233°C (40K).


Case Study

MEMSIC

It is hard enough to design mixed-signal processing onto the same chip as a Micro-Electro-Mechanical Systems (MEMS) device, but MEMSIC has managed to integrate these technologies on the same silicon and sell hundreds of thousands of accelerometers in a variety of industries.

The company has also overcome two other hurdles: keeping production costs low by sticking to a standard CMOS IC process, and standardizing development on a single, lean set of EDA tools.

Detecting Acceleration and Motion

Most accelerometers depend on moving mass to determine motion, but MEMSIC differentiates itself from its competitors through its use of a thermo-mechanical sensor in silicon.

In the centre of the 1mm-square sensor is a heater operating at 100°C above ambient temperature. Around the heater are symmetrically placed thermopiles for reporting temperature in different locations. (A thermopile is a series of thermocouples, or temperature-sensing elements, connected in a series to boost voltage.) The entire sensor is hermetically sealed in an air/gas cavity, outside of which is analog circuitry for amplification, control, analog-to-digital conversion and, in the 3-axis models, digital compensation/calibration circuitry.

In the absence of motion, the thermal profile is balanced among the thermopiles, but any motion or acceleration modifies the convection pattern around the heater, such that the thermopiles in the direction of the acceleration become hotter than the others. The analog circuitry interprets the resulting signal changes from the thermopiles as motion and acceleration.

With no moving parts, MEMSIC's accelerometers are longer-lasting, more reliable, and as much as 25 times more shock-resistant (up to 100,000 g) than their mechanical counterparts for measuring tilt, inclination, shock, or vibration. The chips appear in such products as car

Contd....

Notes

alarms, mobile electronics, global positioning systems, elevator controls, patient monitoring devices and head-mounted displays for gaming.

MEMS Designs, CMOS Fabrication

To take advantage of lower fabrication costs, MEMSIC designs its sensors almost exclusively with standard CMOS layers: for example, the heater is gate polysilicon and the first layer of the thermopile is metal and polysilicon.

"We have a tremendous advantage over our competitors," continues Yongyao. "Our process is almost independent of the fabrication foundry because our design is 95-99% CMOS. We can easily change process and foundry to take advantage of better production pricing. Our competitors, on the other hand, use proprietary MEMS processes, fabricating either by themselves or through a specialized foundry, and that is always more expensive than working with a traditional CMOS foundry."

MEMSIC also enjoys an advantage when changing geometry. Most of its competitors are still producing at 1-2 micron, and a change to .25 micron in MEMS would result in a completely different process and a costly conversion. MEMSIC has produced in .6 and .25 micron—with .18 micron on the roadmap—and its standard CMOS IC process allows it to ramp up volume and production quickly after a change in geometry.

92,000 Accelerometers in Beijing

The marquee application of MEMSIC's technology was in the electronic "Waving Torch" distributed to all attendees of the opening ceremonies for the 2008 Olympics in Beijing.

The torch resembles a 20-30cm wand, with a linear array of LEDs. Shaken from side to side, the torch tricks the human eye into seeing iconic Olympic images—symbols for major sports, the Olympic logo, Chinese greetings, and the five Olympic mascots—displayed in mid-air as the LEDs switch on and off. The core technology in the torch includes a MEMSIC algorithm and accelerometer (designed with Tanner tools) to detect the user's back-and-forth hand movement and to fire each LED as needed for the image.

"We worked on this project for half a year as an Olympic promotional tool," says Yongyao. "The user waves the torch through the air, and the LEDs display the pattern according to the motion. It is a good example of how much information an accelerometer can provide on position, orientation and speed."

About MEMSIC

MEMSIC, Inc. designs, manufactures and markets CMOS Micro-Electro-Mechanical Systems (MEMS) IC products that have on-chip mixed signal processing. MEMSIC is the first and the only company that integrates a MEMS inertial sensor with mixed signal processing circuitry onto a single chip using a standard CMOS IC process. This combination of technology has successfully yielded products at substantially lower cost and higher system performance and functionality than competitive products in the market for sophisticated accelerometers. In addition, this technological approach allows the Company to easily integrate additional functions, or create new sensors, using a standard CMOS IC process to expand into other MEMS application areas beyond accelerometers.

The Company's accelerometers, sometimes called inertial sensors, are used to measure tilt or inclination, shock or vibration, or inertial acceleration. Any application that requires the control or measurement of motion is a potential application for accelerometers.

Contd....

Notes

About MEMS Pro

MEMS Pro is a tool suite from SoftMEMS LLC for designing and analyzing MEMS. Its integration with Tanner and other tool suites shortens development time and provides designers reliable analysis for manufacture.

The MEMS Pro suite offers mixed MEMS/IC schematic capture and simulation, full custom mask layout capability and verification, 3D model generation and visualization, behavioural model creation, and links to 3D analysis packages.

SoftMEMS was founded in 2004 by Dr. Mary Ann Maher. The company's products are based on the MEMS Pro software developed by Dr. Maher's team at Tanner Research in 1997 and the MEMS Xplorer software developed by Dr. Jean Michel Karam's teams at TIMA and MEMSCAP.

Questions:

1. What is the process of detecting acceleration and motion?

2. Explain the inertial sensors.

Self Assessment

True or False:

6. NMOS circuits are slow to transition from low to high.

   (*a*) True (*b*) False

7. Current draw of CMOS devices increases with clock speed.

   (*a*) True (*b*) False

Multiple choice questions:

8. Positive emitter-coupled logic (PECL) is a further development of ECL using a positive 5V supply.

   (*a*) 4V (*b*) 10V

   (*c*) 11V (*d*) 5V

9. In NMOS the DC current must flow through a logic gate even when the output is in a steady state.

   (*a*) steady state (*b*) dynamic state

   (*c*) both (*a*) and (*b*) (*d*) None of these

10. The number of input combinations for a 3-input gate is:

    (*a*) 4 (*b*) 6

    (*c*) 5 (*d*) 8

7.8 Summary

- ICs have become the principal components of almost all electronic devices. These miniature circuits have demonstrated low cost, high reliability, low power requirements, and high processing speeds compared to the vacuum tubes and transistors.
- The fan-out of a gate is limited by the current that its output can supply to the gate inputs connected to it when the output is at logic '1', since at this state it must be able to drive the connected input transistors into saturation.
- CMOS is the other major technology utilized in manufacturing digital IC's aside from TTL, and is now widely used in microprocessors, memories, and digital ASIC's.

Notes

- Transistor-Transistor Logic refers to the technology for designing and fabricating digital integrated circuits that employ logic gates consisting primarily of bipolar transistors.
- The major problem with NMOS is that a DC current must flow through a logic gate even when the output is in a steady state.

7.9 Keywords

***Direct-Coupled Transistor Logic (DCTL) Gate*:** A DCTL gate is one wherein the bases of the transistors are connected directly to inputs without any base resistors.

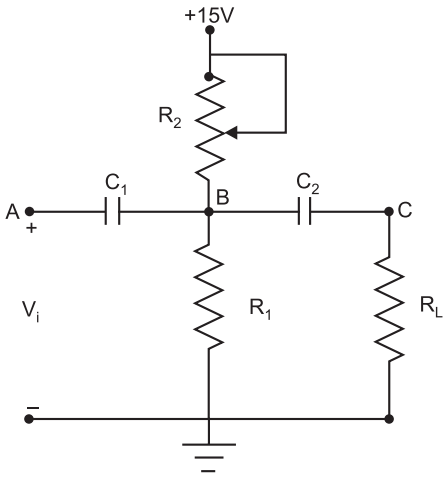
***High Threshold Logic* (*HTL*):** HTL is a variant of DTL which is used in such environments where noise is very high. It incorporates Zener diodes to create a large offset between logic 1 and logic 0 voltage levels

***Metal-Oxide-Semiconductor Field-Effect-Transistor* (*MOSFET*):** A MOSFET can be made to operate as a resistor, so the whole circuit can be made with n-channel MOSFETs only.

***Positive Emitter-Coupled Logic* (*PECL*):** PECL is a further development of ECL using a positive 5V supply instead of a negative 5V supply.

***Transistor-Transistor Logic* (*TTL*):** TTL refers to the technology for designing and fabricating digital integrated circuits that employ logic gates consisting primarily of bipolar transistors.

Lab Exercise



1. Set up the circuit with R1 = 5:1 k, RL = 100 k, and C1 = C2 = 0:47 _F. Use a 10 k potentiometer as R2. Adjust the potentiometer until you read a voltage of 7.5 V between points B and ground. Then, set the function generator to produce a sinusoidal wave with amplitude of 2 V and a frequency of 5 kHz and connect it to the input. Using the scope, sketch the signals at points A, B, and C and explain your observations. Make sure that the scope traces are set to DC.
2. Change the value of potentiometer and report the change in signals at points B and C in above figure.

7.10 Review Questions

1. What is the integrated circuit? Explain in detail.
2. What is RTL? And also explain its working.

Notes

3. Briefly explain about the DTL.
4. What is TTL? And also explain its parameters.
5. What is ECL? Describe its operation and draw the truth table and circuit diagram.
6. Define the manufacturing process of integrated circuits.
7. Explain about HTL with advantages and disadvantages.
8. What is CMOS? And also explain its features and parameters.
9. Explain in detail about NMOS.
10. Explain comparison between CMOS and TTL.

Answers to Self Assessment

1. (*a*) 2. (*a*) 3. (*b*) 4. (*a*) 5. (*a*)
6. (*a*) 7. (*a*) 8. (*d*) 9. (*a*) 10. (*d*)

7.11 Further Reading

Books *Absolute Beginner's Guide to Computer Basics,* by Michael Miller.

Online link www.top-windows-tutorials.com/computer-basics.html

Notes

Unit 8: Memory

CONTENTS

Objectives

After studying this unit, you will be able to:

- Explain the random access memory
- Explain the read only memory
- Describe the programmable read only memory
- Describe the erasable programable read only memory
- Discuss the electrically erasable programable read-only memory
- Explain the programable logic array
- Describe the programmable array logic
- Discuss the use of PAL and PLA in combinational circuit

Introduction

The CPU contains necessary circuitry for data processing controlling other components of the computer. However, one thing it does not have built into it is the place to store programs and data needed during data processing. CPU contains several registers for storing data and instructions but they can store only a few bytes at a time. They are just sufficient to hold only one or two

Notes

instructions with corresponding data. If the instructions and data of a program being executed by a CPU were to reside in secondary storage like a disk, and fetched and loaded one by one into CPU registers as the program execution proceeded, this would lead to the CPU being idle most of the time. This is because there is a large speed mismatch between the rate at which CPU can process data and the rate at which data can be transferred from disk to CPU registers. For example, a CPU can process data at a rate of about 5 nanosecond/byte and a disk reader can read data at a speed of about 5 microseconds/byte. Hence, within the time in which a disk can supply one byte of data a CPU can process 1000 bytes. This would lead to a very slow overall performance even if a computer used a very fast CPU. To overcome this problem, there is a need to have a reasonably large storage space that can hold the instructions and data of the program(s) on which CPU is currently working time to fetch and load data from this storage space into CPU registers must also be very small as compared to that space from disk storage to reduce the speed mismatch problem with CPU speed. Every computer has such a storage space known as primary storage, main memory, or simply memory. It is a temporary storage area built into the computer hardware. Instructions and data of a program reside mainly in this area when CPU is executing the program. Physically, this memory consists of some integrated circuit (IC) chips either on the motherboard or on a small circuit board attached to the motherboard of a computer system. This built-in memory allows CPU to store and retrieve data very quickly. The rate of fetching data from this memory is of the order of 50 nanoseconds/byte. Hence, the rate of data fetching from main memory is about 100 times faster than that from a high speed secondary storage like disk.

8.1 Random Access Memory (RAM)

While memory can refer to any medium of data storage, it usually refers to RAM, or random access memory. When your computer boots up, it loads the operating system into its memory, or RAM. This allows your computer to access system functions, such as handling mouse clicks and keystrokes, since the event handlers are all loaded into RAM. Whenever you open a program, the interface and functions used by that program are also loaded into RAM.

RAM is a very high-speed type of memory, which makes it ideal for storing active programs and system processes. It is different than hard disk space in that RAM is made up of physical memory chips, while hard disks are magnetic disks that spin inside a hard drive. Accessing RAM is much faster than accessing the hard disk because RAM access is based on electric charges, while the hard drive needs to seek to the correct part of the disk before accessing data. However, all the information stored in RAM is erased when the computer's power is turned off. The hard disk, on the other hand, stores data magnetically without requiring any electrical power.

To summarize, memory is a vital part of the way computers and many electronic devices function. While memory and RAM can often be used synonymously, it is good to know about other types of memory as well. Hopefully you will be able to store the information you have learned in your own memory.

The RAM (random access memory) is the place in a computer where the operating system, application programs and data in current use are kept so that they can be quickly reached by the computer's processor. RAM is much faster to read from and write to than the other kinds of storage in a computer, the hard disk, floppy disk and CD-ROM. However, the data in RAM stays there only as long as your computer is running. When you turn the computer off, RAM loses its data. When you turn your computer on again, your operating system and other files are once again loaded into RAM, usually from your hard disk.

The RAM can be compared to a person's short-term memory and the hard disk to the long-term memory. The short-term memory focuses on work at hand, but can only keep so many facts in view at one time. If short-term memory fills up your brain sometimes is able to refresh it from facts stored in long-term memory. A computer also works this way. If RAM fills up, the processor

Notes

needs to continually go to the hard disk to overlay old data in RAM with new slowing down the computer's operation. Unlike the hard disk which can become completely full of data so that it would not accept any more, RAM ever runs out of memory. It keeps operating, but much more slowly than you may want it to.

The RAM is small, both in physical size (it is stored in microchips) and in the amount of data it can hold. It is much smaller than your hard disk. A typical computer may come with 256 million bytes of RAM and a hard disk that can hold 40 billion bytes. RAM comes in the form of "discrete" (meaning separate) microchips and also in the form of modules that plug into holes in the computer's motherboard. These holes connect through a bus or set of electrical paths to the processor. The hard drive, on the other hand, stores data on a magnetized surface that looks like a phonograph record.

Most personal computers are designed, to allow you, to add additional RAM modules up to a certain limit. Having more RAM in your computer reduces the number of times that the computer processor has to read data in from your hard disk, an operation that takes much longer than reading data from RAM. (RAM access time is in nanoseconds; hard disk access time is in milliseconds.)

The RAM is called "random access" because any storage location can be accessed directly. Originally, the term distinguished regular core memory from offline memory, usually on magnetic tape in which an item of data could only be accessed by starting from the beginning of the tape and finding an address sequentially. Perhaps it should have been called "non sequential memory" because RAM access is hardly random. RAM is organized and controlled in a way that enables data to be stored and retrieved directly to specific locations. Note that other forms of storage such as the hard disk and CD-ROM are also accessed directly (or "randomly") but the term random access is not applied to these forms of storage.

8.2 Read Only Memory (ROM)

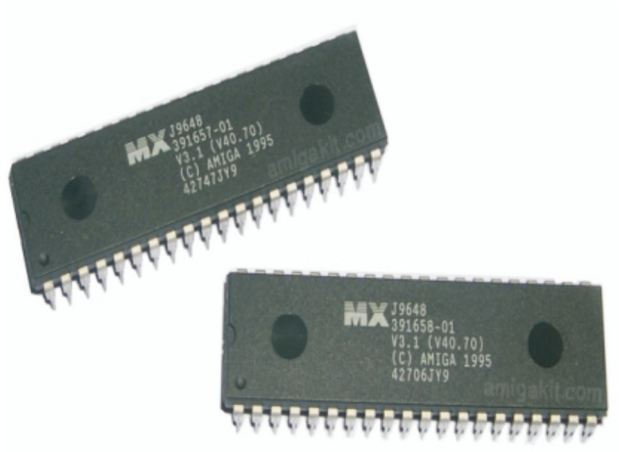
A special type of RAM, called read only memory (ROM), is a non-volatile memory chip in which data is stored permanently and cannot be altered by usual programs. In fact, storing data permanently into this kind of memory is called "burning in the data" because data in such memory is stored by using fuse-links. Once a fuse link is burnt, it is permanent. Data stored in a ROM chip can only be read and used they cannot be changed. This is the reason why it is called read-only memory (ROM). Since ROM chips are non-volatile, data stored in a ROM are not lost when power is switched off or interrupted unlike in the case of a volatile RAM chip. ROM is also known as field stores, permanent stores, or dead stores.

ROMs are mainly used to store programs and data that do not change and are frequently used. Most basic computer operations are carried out by wired electronic circuits. However, several higher-level and frequently used operations require very complicated electronic circuits for their implementation. Hence, instead of building electronic circuits for these operations, special programs are written to perform them. These programs are called microprograms because they deal with low-level machine functions and are essentially substitutes for additional hardware. ROMs are used by computer manufacturers to store these microprograms so that they cannot be modified by the users.

A good example of a microprogram is the set of instructions needed to make a computer system ready to use when it is powered on. This microprogram, called "system boot program", contains a set of start-up instructions to check if the system hardware like memory, I/O devices, etc. are functioning properly. It looks for an operating system and loads its core part in the volatile RAM of the system to produce the initial display-screen prompt. Note that this microprogram is used every time the computer is switched on and needs to be retained when the computer is switched off. Hence, ROM is an ideal storage for storing it.

Notes

Figure 8.1 Read Only Memory (ROM)



Did u know? ROM is useful for binary storage of cryptographic data, as it makes them difficult to replace, which may be desirable in order to enhance information security.

Task Draw the block diagram of CPU.

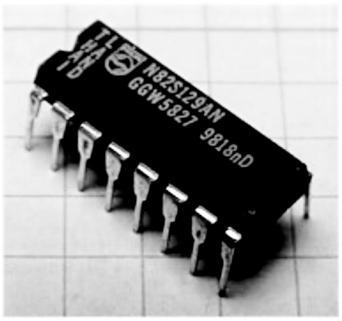
8.3 Programmable Read Only Memory (PROM)

A programmable read only memory (PROM) or field programmable read-only memory (FPROM) or one-time programmable non-volatile memory (OTP NVM) is a form of digital memory where the setting of each bit is locked by a fuse or antifuse.

There are two types of read only memory (ROM) manufacturer-programmed and user-programmed. A manufacturer-programmed ROM is one in which data is burnt in by the manufacturer of the electronic equipment in which it is used. For example, a personal computer manufacturer may store the system boot program permanently in a ROM chip located on the motherboard of each PC manufactured by it. Similarly, a printer manufacturer may store the printer controller software in a ROM chip located on the circuit board of each manufactured by it. Manufacturer-programmed ROMs are used mainly in those cases where the demand for such programmed ROMs is large. Note that manufacturer-programmed ROM chips are supplied by the manufacturers of electronic equipment and it is not possible for a user to modify the programs or data stored in a ROM chip. On the other hand, a user-programmed ROM is one in which a user can load and store "read-only" programs and data. That is, it is possible for a user to "customize" a system by converting his/her programs to microprograms and storing them in a user-programmed ROM chip. Such a ROM is commonly known as Programmable Read-Only Memory(PROM) because a user can program it. Once the user programs are stored in a PROM chip they can be executed usually in a fraction of the time previously required. PROMs are programmed to record information using a special device known as PROM-programmer. However, once the chip has been programmed, the PROM becomes a ROM. That is, the information recorded in it can only be read (it cannot be changed). PROM is also non-volatile storage, i.e., the stored information remains intact even if power is switched off or interrupted.

Notes

Figure 8.2: Programmable Read Only Memory (PROM)



8.4 Erasable Programmable Read Only Memory (EPROM)

It is an array of floating-gate transistors individually programmed by an electronic device that supplies higher voltages than those normally used in digital circuits.

Once information is stored in a ROM or PROM chip it cannot be altered. Erasable Programmable Read Only Memory (EPROM) overcomes this problem. As the name implies, it is possible to erase information stored in an EPROM chip and the chip can be reprogrammed to store new information. EPROMs are often used by R&D personnel (experimenters) who frequently change the microprograms to test the efficiency of a computer system with new programs. EPROMs are also useful for those applications in which one may like to store a program in a ROM that would normally not change but under some unforeseen conditions, one may like to alter it. When an EPROM is in use, information stored in it can only be "read" and the information remains in the chip until it is erased.

EPROM chips are of two types – one in which the stored information is erased by exposing the chip for some time of ultraviolet light and the other one in which the stored information is erased by using high voltage electric pulses is known as Ultra Violet EPROM ((UVEPROAM) and the latter is known as Electrically EPROM (EEPROM). It is easier to alter information stored in an EEPROM chip as compared to an UVEPROM chip. EPROM is also known as flash memory because of the ease with which programs stored in it can be altered. Flash memory is used in many new 1/O and storage devices like USB (Universal Serial Bus) pen drive and MP3 music player.

Figure 8.3: Erasable Programmable Read Only Memory (EPROM)

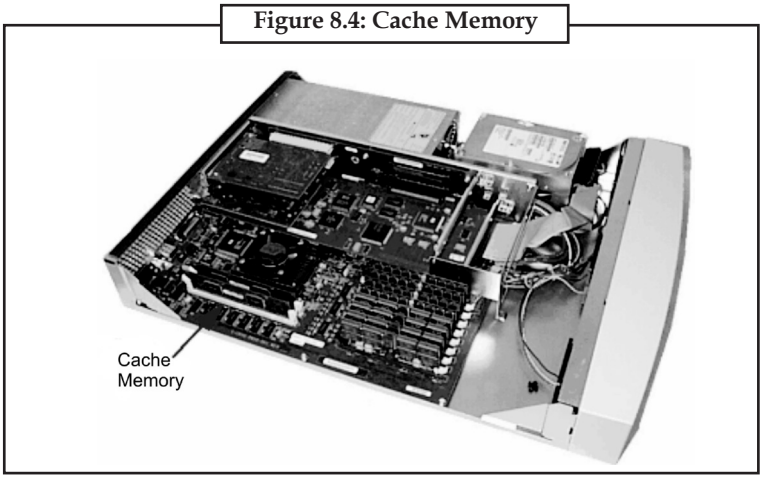
8.4.1 Cache Memory

Use of main memory helps in minimizing disk-processor speed mismatch to a large extent because the rate of data fetching by a computer's CPU from its main memory is about 100 times faster than that from a high-speed secondary storage like disk. However, even with the use of main memory, memory-processor speed mismatch becomes a bottleneck in the speed with which the CPU can process instructions because there is a 1 to 10 speed mismatch between the processor

Notes

and memory. That is, the rate at which data can be fetched from memory is about 10 times slower than the rate at which CPU can process data. Obviously, the overall performance of a processor can be improved greatly by minimizing the memory-processor speed mismatch. Cache memory (pronounced "cash" memory) is commonly used for this purpose. It is an extremely fast, small memory between CPU and main memory whose access time is closer to the processing speed of CPU. It acts as a high-speed buffer between CPU and main memory and is used to temporarily store very active data and instructions during processing. Since cache memory is faster than main memory, processing speed is improved by making the data and instructions needed for current processing available in the cache. Cache memory is an extremely fast and small memory between CPU and main memory. Its access time is closer to the processing speed of CPU. It acts as a high-speed buffer between CPU and main memory and is used to temporarily store very active data and instructions during processing.

Figure 8.4: Cache Memory



Did u know? The EPROM was invented by Dov Frohman of Intel in 1971, who was awarded US patent 3660819 in 1972.

8.5 Electrically Erasable Programmable Read Only Memory (EEPROM)

The EEPROM stands for electrically erasable programmable read only Memory. An EEPROM is like an EPROM chip since, it can be written in or programmed more than once. Unlike the EPROM chip, however, an EEPROM chip need not be taken out of the computer or electronic device of which it is part when a new program or data needs to be written on it.

Selective programing can be done to an EEPROM chip. The user can alter the value of certain cells without needing to erase the programming on other cells. Thus, sections of data can be erased and replaced without needing to alter the rest of the chip's programming.

Data stored in an EEPROM chip is permanent, at least until the user decides to erase and replace the information it contains. Furthermore, the data stored in an EEPROM chip is not lost even when power is turned off.

8.5.1 EEPROM Structure

The EEPROM chip is physically similar to the EPROM chip. It is also composed of cells with two transistors. The floating gate is separated from the control gate by a thin oxide layer. Unlike the EPROM chip, however, the EEPROM chip's oxide layer is much thinner. In EEPROM chips, the insulating layer is only around 1 nanometre thick whereas in EPROM chips, the oxide layer is around 3 nanometres thick. The thinner oxide layer means lower voltage requirements for initiating changes in cell value.

Notes

Tunneling the electrons of the floating gate towards the oxide layer separating the floating gate and the control gate is still the method of changing a bit's value from 1 to 0. To erase EEPROM programming, the electron barrier still has to be overcome by the application of enough programming voltage.

8.5.2 EEPROM Limitations

While the EEPROM can be reprogrammed, the number of times it can be altered, is limited. This is the main reason why EEPROM chips are popular for storing only configuration data such as the computer's BIOS code which does not require frequent reprogramming. The oxide insulating layer can be damaged by frequent rewrite. Modern-day EEPROMs can be rewritten up to a million times.

Task Prepare a process structure of EEPROM.

Self Assessment

Multiple choice questions:

1. RAM chips plugged into special sockets on the motherboard are known as .....
   (*a*) SIMMs (*b*) SINNs
   (*c*) SIMNs (*d*) None of these
2. ROM with 12 input variables would require ..... AND gates.
   (*a*) 212 (*b*) 210
   (*c*) 222 (*d*) 22
3. ROM has a large number of fusible links .....
   (*a*) m × 2 (*b*) m × 2n
   (*c*) m2 × 2n (*d*) m2 × 2

True or False:

4. EPROM chips are of four types.
   (*a*) True (*b*) False
5. RAM is much faster to read from and write to than the other kinds of storage.
   (*a*) True (*b*) False
6. Read only memory (ROM) is a volatile memory.
   (*a*) True (*b*) False

8.6 Programmable Logic Array (PLA)

One way to design a combinational logic circuit is to get gates and connect them with wires. One disadvantage with this way of designing circuits is its lack of portability. You can now get chips called PLA (programmable logic arrays) and "program" them to implement Boolean functions.

Fortunately, a PLA is quite simple to learn, and produces nice neat circuits too.

A programmable logic array (PLA) is a large scale integrated programmable logic device which is used for synthesizing combinational logic functions. It consists of a programmable AND gate

Notes

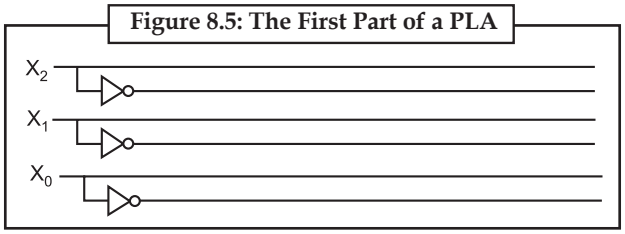
array followed by a programmable OR gate array. To synthesize the output logic functions, first of all these are needed to be reduced to their minimum sum of product expressions. In PLA implementation, a designer should try to deduce these expressions in such a way that maximum number of common product terms exist between them. All these product terms are then generated in the AND gate array. From there, these product terms are fed in to the OR gate array where they are added according to the deduced logic expressions in order to get the output functions.

The size of a programmable logic array is defined in terms of, number of inputs, number of product terms number of outputs etc. A, 4 input, 8, product terms and 4, output PLA will be defined as a (4, 8, 4) PLA. As the logic functions are synthesized almost in terms of their minimum sum of product expressions, the digital hardware required in a PLA is much less than that of a PROM. Also the costly decoder circuitry present in a PROM is replaced in the PLA by a more economical AND gate array. Use of less hardware also provides advantages in terms of speed, power consumptions, etc.

The biggest disadvantage of a Programmable Logic Array is the two steps, programming involved. A designer has to first of all program the AND gate array and then program the OR gate array according to the programming of the AND gate array. Also PLA implementation requires the designer to try and deduce logic expressions with as many common terms between different expressions as possible. As this is an intuitive skill rather than a set procedure, that makes it very hard to automate or computerize the design process.

To counter this problem, another line of programmable logic devices, known as PAL's have been devised. These involve only one step programming while also it having similar hardware advantages of a PLA. In fact, PAL's (programmable array logic) have become the most popular LSI programmable logic devices today, far surpassing PROM's and PLA's.

Figure 8.5: The First Part of a PLA

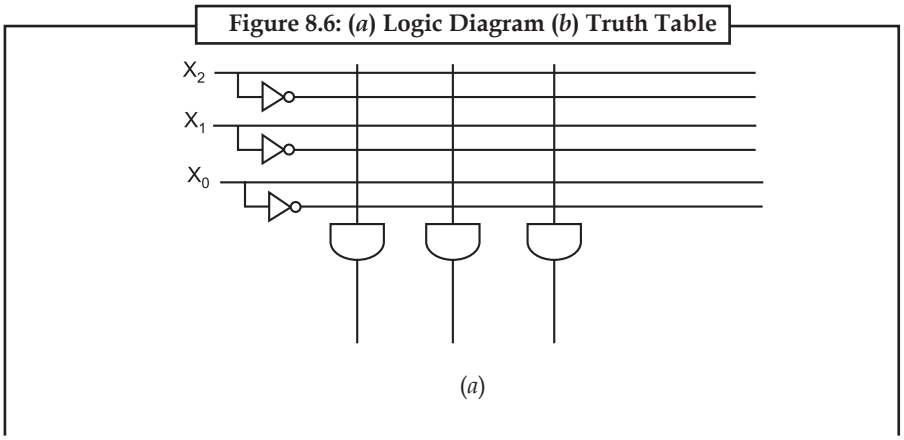


Each variable is hooked to a wire, and to a wire with a NOT gate. So, the top wire is x_2 and the one just below is its negation, $\backslash x_2$.

Then there is x_1 and just below is, its negation, $\backslash x_1$.

The next part is to draw a vertical wire with an AND gate. We have drawn 3 of them.

Figure 8.6: (*a*) Logic Diagram (*b*) Truth Table



(*a*)

Notes

Let us try to implement a truth table with a PLA.

x_2	x_1	x_0	z_1	z_0
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	1	0
1	0	0	1	1
1	0	1	0	0
1	1	0	0	0
1	1	1	0	1

(*b*)

Each of the vertical lines with an AND gate corresponds to a minterm. For example, the first AND gate (on the left) is the minterm: $\backslash x_2 \backslash x_1 x_0$.

The second AND gate (from the left) is the minterm: $\backslash x_2 x_1 x_0$.

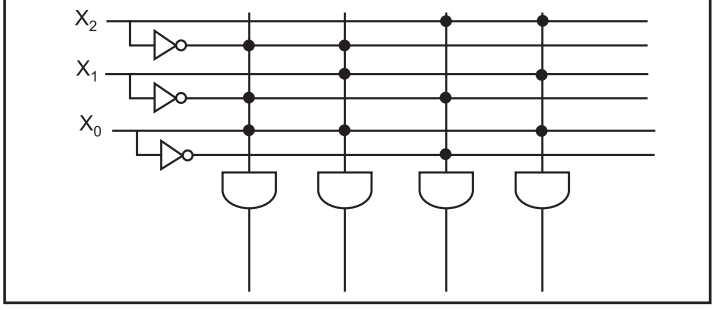
The third AND gate (from the left) is the minterm: $x_2 \backslash x_1 \backslash x_0$.

We have added a fourth AND gate which is the minterm: $x_2 x_1 x_0$.

The first three minterms are used to implement z_1. The third and fourth minterms are used to implement z_0.

This is how the PLA looks after we have all four minterms.

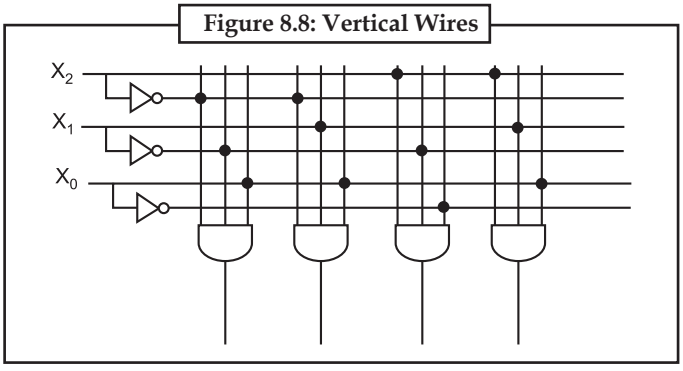
Figure 8.7: PLA After Loooks Four Minterms



Now, you might complain. How is it possible to have a one input AND gate? How can three inputs be hooked to the same wire to an AND gate? Is not that invalid for combinational logic circuits?

That is true, it is invalid. However, the diagram is merely a simplification. We have drawn the each of AND gate with three input wires, which is what it is in reality (there is as many input wires as variables). For each connection (shown with a black dot), there's really a separate wire. We draw one wire just to make it look neat.

Figure 8.8: Vertical Wires

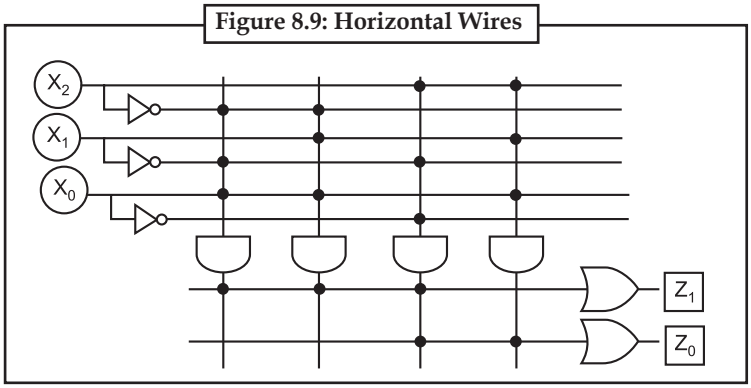


Notes

The vertical wires are called the AND plane. We often leave out the AND gates to make it even easier to draw.

We then add OR gates using horizontal wires, to connect the minterms together.

Figure 8.9: Horizontal Wires

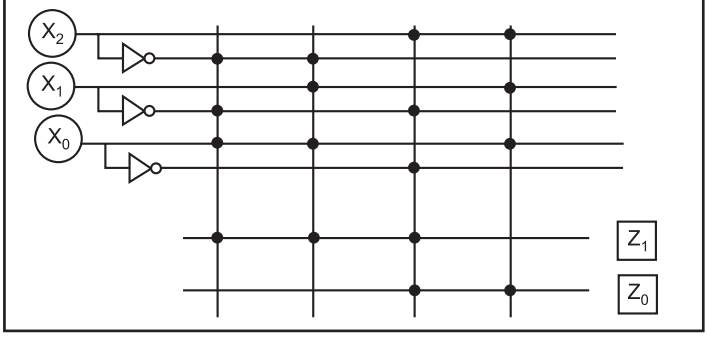


Again, a single wire into the OR gate is really 4 wires. We use the same simplification to make it easier to read.

The horizontal wires make up the OR plane.

This is how the PLA looks when we leave out the AND gates and the OR gates. It is not that the AND gates and OR gates are not there–they are, but they have been left out to make the PLA even easier to draw.

Figure 8.10: PLA without AND and OR Gates



8.6.1 Minterms

Given n variables, it would seem necessary to have 2^n vertical wires (for the AND gates), one for each possible minterm. However, 2^n grows very quickly. So, sometimes there are not 2^n vertical wires.

You can generally get around the problem by not connecting the wire to each of the three variables. For example, you could just have a product term $x_2 \backslash x_0$ or even simply $\backslash x_1$.

For the purpose of implementing truth tables, we will usually tell you not to simplify, and to let each vertical line be a minterm.

8.6.2 Programming

What does it mean to program a PLA? See the black dots. Those connections are made between wires. In effect "programming" the wires means to make the connections within the PLA. Configurable might be a better word than programmable, but that is the name that stuck.

Notes

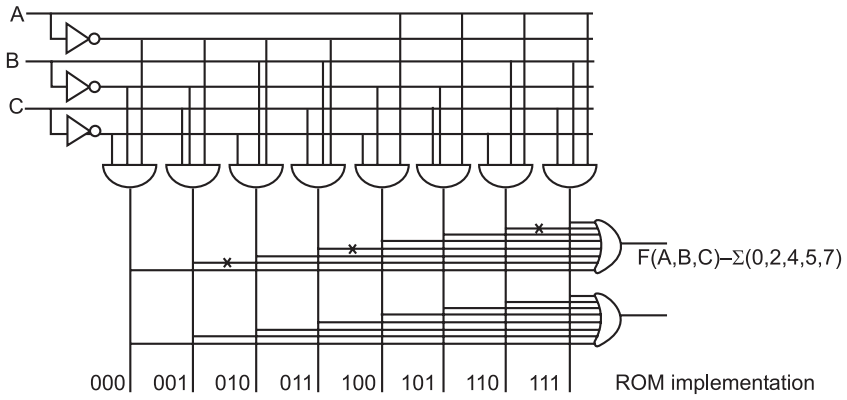
Example: Implement the following function by ROM and PLA:

$$f(A, B, C) = \Sigma(0, 2, 4, 5, 7)$$

This function is implemented by ROM as the OR of five minterms:

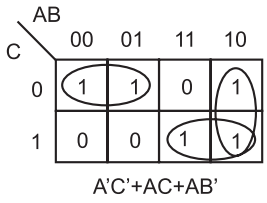
$$f(A, B, C) = A'B'C' + A'BC' + AB'C' + AB'C + ABC$$

Figure 8.11: PLA implementation (*a*) Rom Implementation (*b*) K-Map for Simplification (*c*) Logic Implementation of PLA (*d*) Circuit Diagram



(*a*)

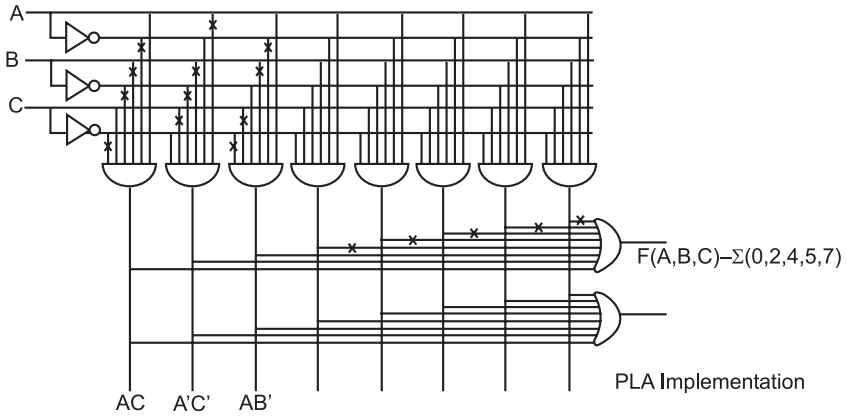
But after simplification by Karnaugh map:



(*b*)

The function can be implemented by a PLA as the OR of three terms:

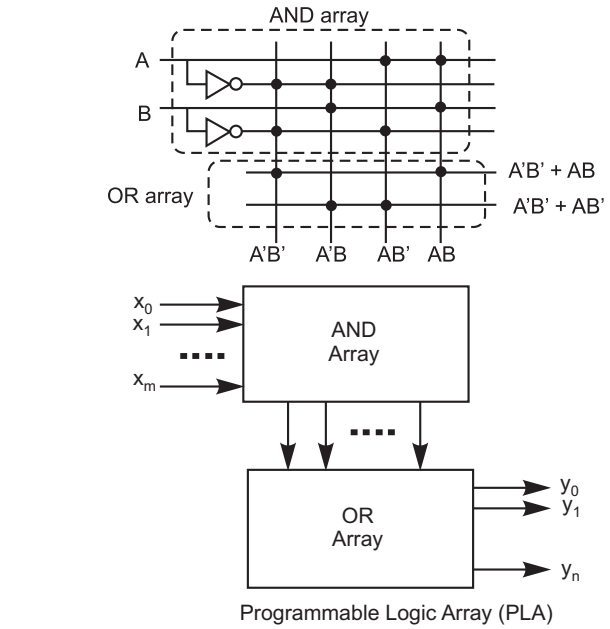
$$f(A, B, C) = AC + A'C' + AB'$$



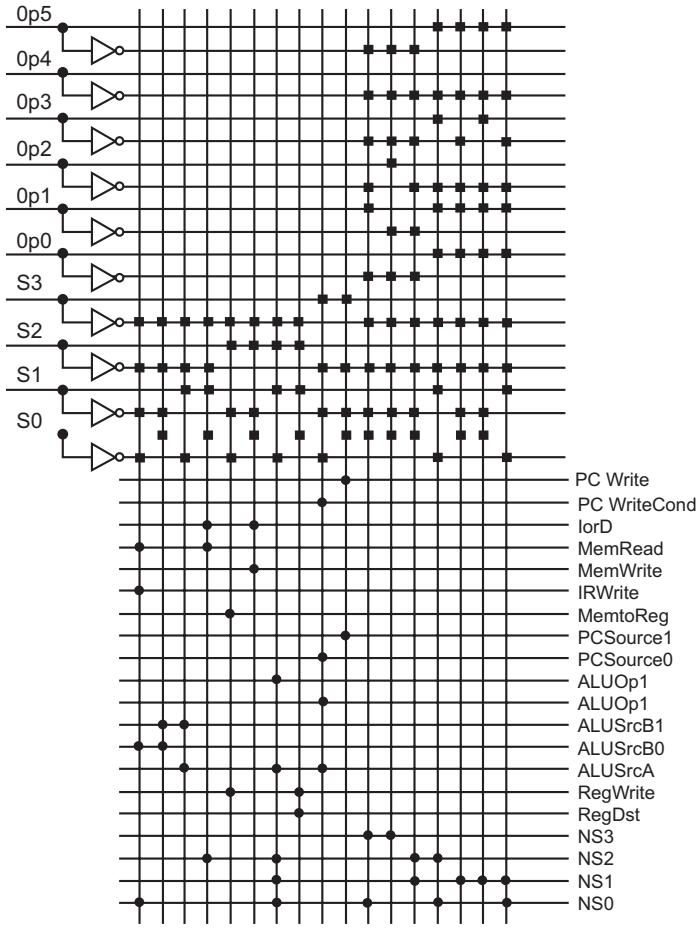
(*c*)

Note that in a PLA, only those terms that are needed are generated by the AND array, while in a ROM, all minterms have to be generated by the AND array.

Notes



Programmable Logic Array (PLA)



(*d*)

Caution The PROM chip can be easily damaged by mishandling. Static electricity can zap the chip's memory, and the little prongs that plugged into the circuit board can be easily bent or broken.

Notes

8.7 Programmable Array Logic (PAL)

A ROM has a large number of fusible links ($m \times 2^n$) because of the large number (2^n) of AND gates. Programming of links is performed only on the outputs from the AND gates. In a PLA, the number of links is drastically reduced by reducing the number of AND gates. The latter is done by changing the expression representing the switching function from a canonic sum-of-products form to a sum of products with fewer terms. The price paid is the need to program not only the outputs from the AND gates, but also the inputs to the AND gates. What other possibility for programming is there beyond the two cases of (*a*) programming the outputs of the AND gates and (*b*) programming both the inputs and the outputs? We are sure you answered, "Programming only the inputs." This is a possibility, but is it worthwhile?

In the case of the ROM, there is no need to program the inputs because, for any function of n variables, there will be the same (large) number of AND gates. In the same way, if the number of OR gates at the output could be fixed, then programming the outputs of the AND gates could be avoided.

In many circuits with multiple outputs, even though the outputs are functions of a large number of input variables, the number of product terms in each output is small. Hence, the number of AND gates that drive each OR gate is small. In such cases, permanently fixing the number of OR gates and leaving only the programming of the AND gate inputs for individual design might make economic sense. The resulting circuit is called programmed array logic (PAL). The number of fusible links in a PAL is only 2np. Standard PALs for a number of low values of p exist. For example, the PAL16L8 has a maximum of 16 inputs and 8 outputs.

A programming table for a PAL is similar to the one for a PLA. A case with six outputs is illustrated in Figure 8.12. A ROM with 12 input variables would require $2^{12} = 4096$ AND gates. However, let us assume that for some possible cases, the canonic sum-of-products expression can be reduced to 16 applicants, only one of which is shown in Figure 8.12. The entries in the table would have the same meanings as those for the PLA. However, for the PAL, the output columns would be fixed by the manufacturer on the basis of the number of AND gates already connected to each OR gate.

Figure 8.12: Programming Table for a PAL Example

Product Term		Inputs												Outputs					
Number	Function	1	2	3	4	5	6	7	8	9	10	11	12	1	2	3	4	5	6
1														•	•	•	•	•	1
2														•	•	•	•	•	1
3														•	•	•	•	•	1
4														•	•	•	•	•	1
5														•	•	•	•	1	•
6														•	•	•	•	1	•
7	$x_1x_2'x_5x_7x_{11}'x_{12}$	1	0	–	–	1	–	1	–	–	–	0	1	•	•	•	1	•	•
8														•	•	•	1	•	•
9														•	•	1	•	•	•
10														•	•	1	•	•	•
11														•	1	•	•	•	•
12														•	1	•	•	•	•
13														1	•	•	•	•	•
14														1	•	•	•	•	•
15														1	•	•	•	•	•
16														1	•	•	•	•	•

In the present case, two of the output OR gates are each driven by four AND gates; the remaining four OR gates are each driven by two AND gates. For any given design problem, the first step is to obtain an appropriate sum-of-products expression, just as in the case of a PLA implementation. The input connections are indicated in the table as in the case of the PAL: an entry is an 1 if a variable appears un-complemented in an implicit, a 0 if it appears complemented, and a dash if it does not appear at all. This is illustrated for one row in Figure 8.12. The number of fusible links in this example is $2 \times 12 \times 16 = 384$. This is 20 percent fewer than the number of links of a PLA having

Notes

the same dimensions. Typically, however, PLAs have many more AND gates and so, for a PAL, the number of links would typically be many times more than the number for a comparable PLA.

8.8 Use of PAL and PLA in Combinational Circuit

The elements of Boolean algebra (two-element "switching algebra") and how the operations in Boolean algebra can be represented schematically by means of gates (primitive devices). How switching expressions can be manipulated and represented in different ways was the subject which also presented various ways of implementing such representations in a variety of circuits using primitive gates.

With all of the tools for the purpose now in hand, The design of more complex logic circuits. Circuits in which all outputs at any given time depend only on the inputs at that time are called combinational logic circuits. The design procedures will be illustrated with important classes of circuits that are now universal in digital systems.

The approach taken is to examine the tasks that a combinational logic circuit is instated to perform and then identify one or more circuits that can perform the task. One circuit may have some specific advantages over others, but it may also have certain deficiencies. Often one factor can be improved, but only at the expense of others. Some important factors are speed of operation, complexity or cost of hardware, power dissipation, and availability in prefabricated units. We will take up a number of different operations that are useful in different contexts and show how appropriate circuits can be designed to carry out these operations.


Caution

During the reprogramming procedure, power to the module and reprogramming tool must not be interrupted.

8.8.1 Combinational Circuit Implementation with PLA

When implementing combinational circuit using PLA, we must reduce the number of product terms. Number of literals in each product term is not important because all variables and their complements are available.

In order to reduce the number of product terms, we have to simplify the functions and their complements in order to fine the combination that result in the minimum number of product terms.

The size of the PLA is determined by the number of inputs, Number of product terms and the number of outputs.

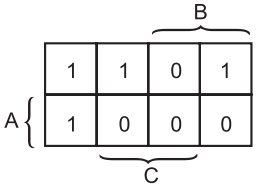

Example:

Implement the following two Boolean functions using a PLA.

$$F_1(A, B, C) = \Sigma(0, 1, 2, 4) \text{ and } F_2(A, B, C) = \Sigma(0, 5, 6, 7)$$

First, we simplify the functions and their complements

$$F_1 = A'B' + A'C' + B'C'$$



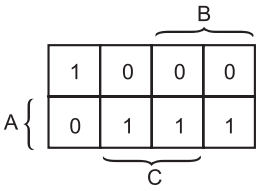
and

$$F_1' = AB + AC + BC$$

$$F_2 = AB + AC + A'B'C'$$

Notes

and $F_2' = A'B + A'C + AB'C'$



The minimum number of product terms is (4) and is obtained if we implement F1 in complement form and F2 in true form.

The following PLA programming table is used.

Table 8.1: PLA Programming Table

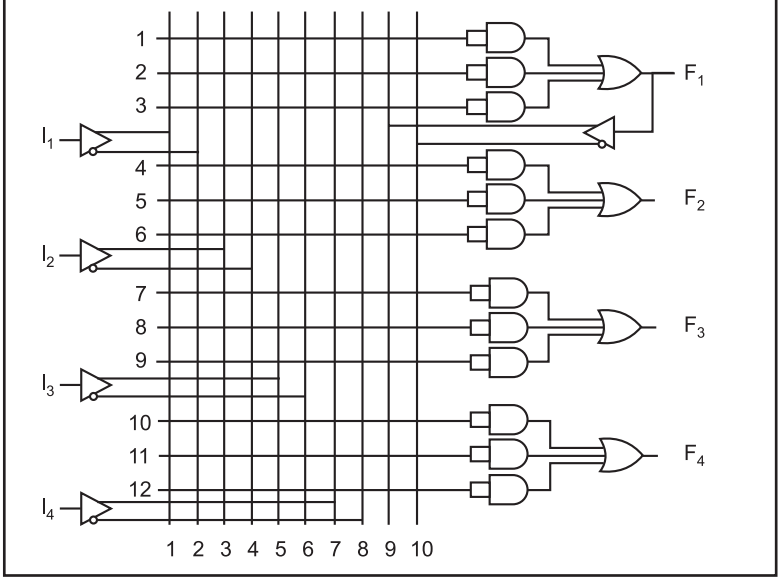
		Inputs			Outputs	
Product terms		A	B	C	F_1	F_2
1	B′C′	–	0	0	1	–
2	AB	1	1	–	–	1
3	A′BC′	0	1	0	1	1
4	AC	1	–	1	1	–
True/Complement →					T	C

8.8.2 Programmable Array Logic (PAL)

PAL has a fixed array of OR gates and programmable array of AND gates. It is easier to program than the PLA but is less flexible. It is suitable to implement a number of Boolean functions which have no or a few overlapping product terms.

The logic diagram of a typical PAL is shown in the following Figure 8.13.

Figure 8.13: PAL Logic Circuit



When we implement Boolean functions with PAL, we must simplify the functions to fit the sections of the PAL. A product term cannot be shared among two or more OR gates. If the function cannot fit in one section, then we may use two sections to implement the function.

Notes

Example:

Implement the following Boolean functions using a PAL that has four sections with three product terms each.

$$F_1(A, B, C, D) = \Sigma(2, 12, 13)$$

$$F_2(A, B, C, D) = \Sigma(7, 8, 9, 10, 11, 12, 13, 14, 15)$$

$$F_3(A, B, C, D) = \Sigma(0, 2, 3, 4, 5, 6, 7, 8, 10, 11, 15)$$

$$F_4(A, B, C, D) = \Sigma(1, 2, 8, 12, 13)$$

Simplifying the functions results in the following functions:

$$F_1(A, B, C, D) = ABC' + A'B'CD'$$

$$F_2(A, B, C, D) = A + BCD$$

$$F_3(A, B, C, D) = A'B + CD + B'D'$$

$$F_4(A, B, C, D) = ABC' + A'B'CD' + AC'D' + A'B'C'D$$

$$= F_1 + AC'D' + A'B'C'D$$

Did u know?

The project to create the PAL device was managed by John Birkner and the actual PAL circuit was designed by H. T. Chua.

Case Study

Fine Soft Studio Memory Management

Fine Soft Studio developed one of the most efficient memory manager in the world. Applications that use our memory manager show striking improvement of runtime performance, especially on multiprocessor platforms.

Business description

Existing hardware is constantly improving. Multiprocessors and CPUs with more than one core becoming usual things. From new applications are expected to use efficiently the possibilities of coming hardware. One of the most important system components is Memory Manager. The part of the system that manages program memory so that the program can run most efficiently. Modern programs have object-oriented, multithreaded architectures which put a lot of pressure on memory manager. For web applications hit processing time is measured in milliseconds, so it is very important to have optimized each component of the system.

About 20% of CPU time is spent for managing memory. The application memory manager must keep it up with minimum of CPU overhead, and the total memory used.

But existing memory managers have not cope with this goals adequately especially when running on multi-CPU ore multi-core modern hardware.

Existing memory managers are not able to get most of computer hardware and substantially limit hardware scalability. When we add new processor, or additional core to CPU programs slow down. Instead of expected linear increase runtime performance decreases because memory managers have memory locking problems.

If two threads request a memory at the same time, the memory manager must enqueue the requests before processing them. One thread must wait for another thread's request to be processed. This serialization step is a large performance barrier, resulting in scaling problems.

Contd...

Notes

The challenge was to build own memory manager which ensures increasing of application runtime performance when new CPU is added and which must be memory space efficient and very fast.

Solution and key benefits

We developed our own memory manager. It uses advanced memory management algorithm. The key benefits of our memory manager are:

it does not use critical sections (uses no-blocking algorithm)

it does not require CPU context switching

Memory Manager 100% effectively consumes all CPUs which are in system

Memory Manager does not have performance degradation when CPU count increases (performance of other memory managers either do not increase or even decrease when CPU count increase)

optimized work with memory blocks of any size

even after long time of work memory is not fragmented.

It is very important for web applications, or for applications which should run for long time without restart - like database servers, operation systems, etc

Memory Manager has built-in logic to capture memory leaks and do memory dumps to see memory disposition/reallocations during application execution

You do not need to make any changes in your existing code to use our Memory Manager

Test application

Application creates a specified number of threads. Each thread executes same actions (emulating memory manager real usage): randomly selects pointer from an array and allocates block of memory if pointer is not assigned or frees memory if pointer is assigned. Size of the memory block is random within defined margins. Application calculates total number of memory block allocations and releases in millions per sec. Resulting value is the maximum value achieved during specified period (higher is better).

Test system configuration: CPU - Inter Core 2 Quad (4 cores), OS - Windows Server 2003

Testing Results

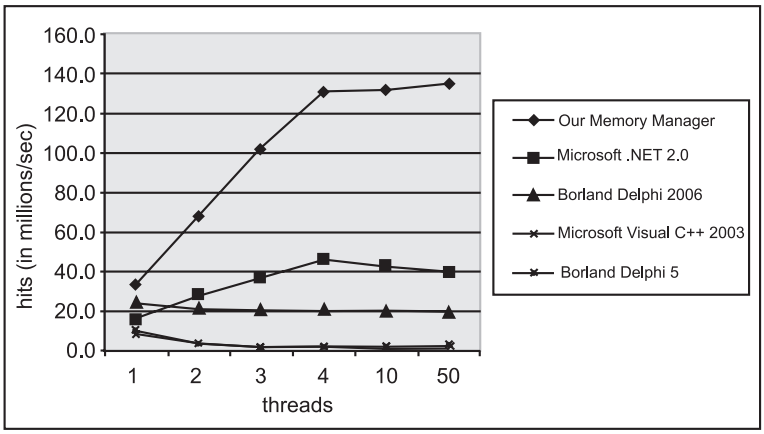
Memory Manager (expected performance growth)	1 thread	2 threads (+100%)	3 threads (+200%)	4 threads (+300%)	10 threads (+300%)	50 threads (+300%)
Our Memory Manager (actual performance growth)	33.9 m/sec	67.8 m/sec (+100%)	101.6 m/sec (+199%)	131.5 m/sec (+287%)	132.0 m/sec (+289%)	134.5 m/sec (+296%)
Microsoft .NET 2.0 (*) (actual performance growth)	15.2 m/sec	27.8 m/sec (+82%)	37.2 m/sec (+144%)	45.7 m/sec (+200%)	42.6 m/sec (+180%)	39.3 m/sec (+158%)
Borland Delphi 2006 (actual performance growth)	23.5 m/sec	21.0 m/sec (–11%)	20.7 m/sec (–12%)	20.5 m/sec (–13%)	19.7 m/sec (–16%)	19.3 m/sec (–18%)

Contd...

Notes

Microsoft Visual C++ 2003 (actual performance growth)	8.1 m/sec	3.0 m/sec (–63%)	1.4 m/sec (–83%)	1.3 m/sec (–84%)	1.2 m/sec (–85%)	1.1 m/sec (–86%)
Borland Delphi 5 (actual performance growth)	9.5 m/sec	3.6 m/sec (–62%)	1.7 m/sec (–82%)	1.7 m/sec (–82%)	1.6 m/sec (–83%)	1.4 m/sec (–85%)

memory manager in .NET by default initializes allocated memory block to zeros. In diagram we showed extrapolated values to eliminate this factor. Real values are about 25% less.



Testing Results Analysis

As you may see from the table above our Memory Manager shows dramatic performance increase even for single threaded mode and perfect scalability for multithreaded mode on multiprocessor/multicore system. The closest competitor MS NET 2.0 shows 2 times less performance even for single threaded mode and much less scalability. Other memory managers shows even performance degradation when running in multithreaded mode.

Testing Results on 16 Cores system

As a Partner in Intel® Software Partner Program we got access to latest hardware platforms and tested our memory manager on 16 cores system (4 CPU Intel Xeon, 4 Cores each). Performance increase of our application running on such system reached unbelievable 15.5 times!

Questions:

1. Discuss the testing result of fine soft studio memory management.
2. Explain the business description of fine soft studio memeory management.

Self Assessment

Multiple choice questions:

7. PAL has a fixed array of ..... gates.
   (*a*) AND (*b*) NOT
   (*c*) OR (*d*) EX-OR
8. The size of the PLA is determined by the number of .....
   (a) functions (*b*) gates
   (*c*) output (*d*) inputs

Notes

9. ..... is the set of instructions.

(*a*) Micro-process (*b*) Micro-processor

(*c*) Multi-program (*d*) Micro-program

8.9 Summary

- CPU contains necessary circuitry for data processing controlling other components of the computer.
- Every computer has such a storage space known as primary storage, main memory, or simply memory.
- A manufacturer programmed ROM is one in which data is burnt in by the manufacturer of the electronic equipment in which it is used.
- Selective programming be done to an EEPROM chip. The user can alter the value of certain cells without needing to erase the programming on other cells.
- Data stored in an EEPROM chip is permanent, at least until the user decides to erase and replace the information it contains.
- While the EEPROM can be reprogrammed, the number of times it can be altered is limited.

8.10 Keywords

***EPROM chips*:** these are of two types - one in which the stored information is erased by exposing the chip for some time of ultraviolet light and the other one in which the stored information is erased by using high voltage electric pulses is known as Ultra Violet EPROM ((UVEPROAM) and the latter is known as Electrically EPROM (EEPROM).

***Erasable programmable read-only memory* (*EPROM*):** It overcomes this problem as the name implies, it is possible to erase information stored in an EPROM chip and the chip can be reprogrammed to store new information.

***Programmable logic array* (*PLA*):** It is a large scale integrated programmable logic device which is used for synthesizing combinational logic functions.

***Read-only memory* (*ROM*):** It is a non-volatile memory chip in which data is stored permanently and cannot be altered by usual programs. In fact, storing data permanently into this kind of memory is called "burning in the data" because data in such memory is stored by using fuse-links.

***Tunneling*:** The electrons of the floating gate towards the oxide layer separating the floating gate and the control gate is still the method of changing a bit's value from 1 to 0.

Lab Exercise

1. Draw the logic circuit and truth table of Programmable Logic Array.
2. Prepare a truth table of vertical wires with three inputs.

8.11 Review Questions

1. Define the RAM and draw the block diagram of RAM.
2. Explain the difference between RAM and ROM.
3. What is a sequential-access storage device?
4. Distinguish between a sequential access, a direct access, and a random access storage device. Write one example of each.

Notes

5. A 4-bit binary number Y = y3y2y1y0 is to be multiplied by a 3-bit binary number X = x2x1x0. Use two 4-bit adders and other gates that you might need to implement this operation, and draw the corresponding diagram.
6. Define programmable logic array (PLA). And draw the circuit diagram.
7. What do you understand by read-only memory (ROM).
8. Write a short note on tunneling.
9. Describe erasable programmable read-only memory (EPROM).
10. What are EPROM chips?

Answers to Self Assessment

1. (*a*) 2. (*a*) 3. (*b*) 4. (*b*) 5. (*a*)
6. (*b*) 7. (*c*) 8. (*d*) 9. (*a*)

8.12 Further Readings


Books

Switching Theory: Insight Through Predicate Logic, by Shimon Peter Vingron.

Absolute Beginner's Guide to Computer Basics, by Michael Miller.


Online link

www.top-windows-tutorials.com/computer-basics.html

Notes

Unit 9: Flip-Flops

CONTENTS

Objectives

After studying this unit, you will be able to:

- Describe the flip-flop circuit
- Explain the types of flip-flop
- Discuss the master slave flip-flop
- Describe the triggering of flip-flops
- Explain the timing signal

Introduction

Flip-flops; not the sandals, but the logic gates, are the fundamental building blocks of sequential logic. There are a variety of different flip-flop types and configurations. In this activity, and this course for that matter, we will only be studying two, the D flip-flop and the J/K flip-flop. After reviewing the basic operation of the 74LS74 D and the 74LS76 J/K flip-flops, this activity will examine two introductory applications of flip-flops.

Note: Where did the flip-flops get their name? The D in the D flip-flop stands for data. No one is absolutely sure where the J/K name originated, but one theory is that it is named after Jack Kilby, the inventor of the integrated circuit.

A timing signal generating device operates in response to angular displacement of a rotating shaft. The shaft has a disc provided with a reference mark and a plurality of timing marks apart from one another in a circumferential direction. A first detector is located adjacent to the disc to detect the reference mark and to reset a shift register. A second detector is also located adjacent

Notes

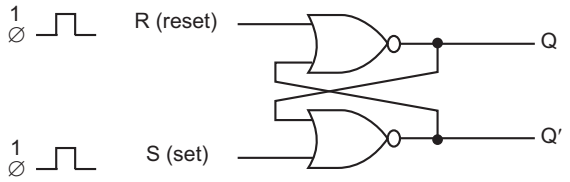
to the disc to generate timing pulses in response to the timing marks and supply the pulses to the shift register. Each of plural logic gates has two input terminals each of which is connected to an output of a predetermined step of the shift register, so that an output signal outputted from each logic gate to the corresponding driving circuit for an operating device constitutes timing signal carrying information including starting and terminating times of operation.

9.1 Basic Flip-Flop Circuit

A flip-flop circuit can be construct from two NAND gates or two NOR gates. These flip-flops are shown in Figure 9.1 and Figure 9.2. Each flip-flop has two outputs, Q and Q′, and two inputs, set and reset. This type of flip-flop is referred to as an SR flip-flop or SR latch. The flip-flop in Figure 9.1 has two useful states. When Q = 1 and Q′ = 0, it is in the set state (or 1-state). When Q = 0 and Q′ = 1, it is in the clear state (or 0-state). The outputs Q and Q′ are complements of each other and are referred to as the normal and complement outputs, respectively. The binary state of the flip-flop is taken to be the value of the normal output.

When an 1 is applied to both the set and reset inputs of the flip-flop in Figure 9.1, both Q and Q′ outputs go to 0. This condition violates the fact that both outputs are complements of each other. In normal operations this condition must be avoided by making sure that 1's are not applied to both inputs simultaneously.

Figure 9.1: Basic Flip-Flop Circuit with NOR Gates (*a*) Logic Diagram (*b*) Truth Table

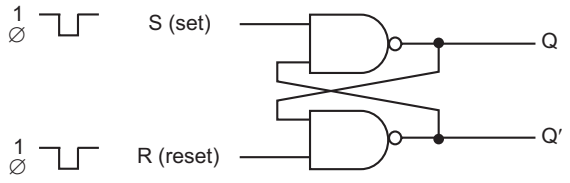


(*a*)

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

(*b*)

Figure 9.2: Basic Flip-Flop Circuit with NAND Gates (*a*) Logic Diagram (*b*) Truth Table



(*a*)

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

(*b*)

Notes

The NAND basic flip-flop circuit in Figure 9.2 (a) operates with inputs normally at 1 unless the state of the flip-flop has to be changed. A 0 applied momentarily to the set input causes Q to go to 1 and Q′ to go to 0, putting the flip-flop in the set state. When both inputs go to 0, both outputs go to 1. This condition should be avoided in normal operation.

Did u know? The first electronic flip-flop was invented in 1918 by William Henry Eccles and F. W. Jordan.

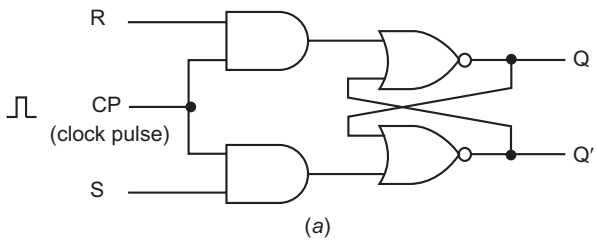
9.2 Types of Flip-Flop

The types of flip-flop circuit that are being used in digital circuits. They can be classified according to the number of inputs they possess and the manner in which they affect the binary state of the flip-flop.

9.2.1 Clocked SR Flip-Flop

The clocked SR flip-flop shown in 3 consists of a basic NOR flip-flop and two AND gates. The outputs of the two AND gates remain at 0 as long as the clock pulse (or CP) is 0, regardless of the S and R input values. When the clock pulse goes to 1, information from the S and R inputs passes through to the basic flip-flop. With both S = 1 and R = 1, the occurrence of a clock pulse causes both outputs to momentarily go to 0. When the pulse is removed, the state of the flip-flop is indeterminate, i.e., either state may result, depending on whether the set or reset input of the flip-flop remains a 1 longer than the transition to 0 at the end of the pulse.

Figure 9.3: Clocked SR Flip-Flop (*a*) Logic Diagram (*b*) Truth Table



(*a*)

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

(*b*)

Task Prepare a truth table for NAND gate.

9.2.2 JK Flip-Flop

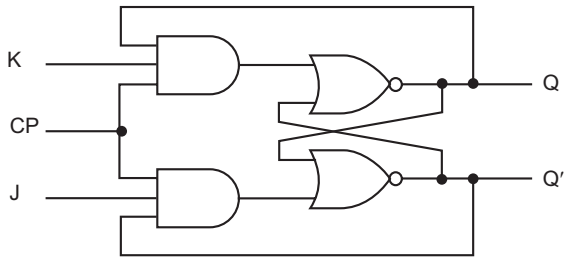
A JK flip-flop is a refinement of the SR flip-flop in that the indeterminate state of the SR type is defined in the JK type. Inputs J and K behave like inputs S and R to set and clear the flip-flop (note that in a JK flip-flop, the letter J is for set and the letter K is for clear). When logic 1 inputs are applied to both J and K simultaneously, the flip-flop switches to its complement state, i e., if Q = 1, it switches to Q = 0 and vice versa.

Notes

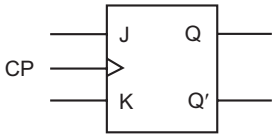
A clocked JK flip-flop is shown in Figure 9.4. Output Q is ANDed with K and CP inputs so that the flip-flop is cleared during a clock pulse only if Q was previously 1. Similarly, ouput Q′ is ANDed with J and CP inputs so that the flip-flop is set with a clock pulse only if Q′ was previously 1.

Note that because of the feedback connection in the JK flip-flop, a CP signal which remains a 1 (while J=K=1) after the outputs have been complemented once will cause repeated and continuous transitions of the outputs. To avoid this, the clock pulses must have time duration less than the propagation delay through the flip-flop. The restriction on the pulse width can be eliminated with a master-slave or edge-triggered construction. The same reasoning also applies to the T flip-flop presented next.

Figure 9.4: Clocked JK Flip-Flop (*a*) Logic Diagram (*b*) Graphical Symbol (*c*) Transition Table



(*a*)



(*b*)

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

(*c*)

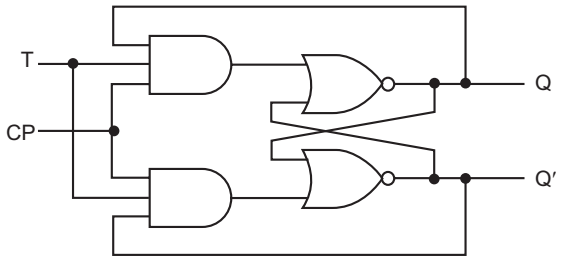
Task Create a structure to make the Q output of JK flip-flop always changes upon the occurrence of the active transition?

9.2.3 T Flip-Flop

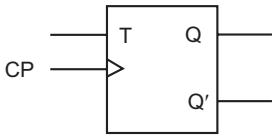
The T flip-flop is a single input version of the JK flip-flop. As shown in Figure 9.5, the T flip-flop is obtained from the JK type if both inputs are tied together. The output of the T flip-flop "toggles" with each clock pulse.

Notes

Figure 9.5: Clocked T Flip-Flop (*a*) Logic Diagram (*b*) Graphical Symbol (*c*) Transition Table



(*a*)



(*b*)

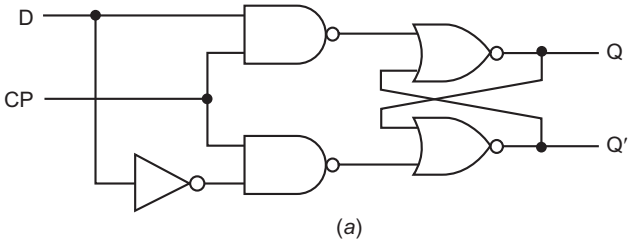
Q	J	Q(t+1)
0	0	0
0	1	1
1	0	1
1	1	0

(*c*)

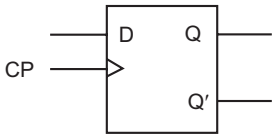
9.2.4 D Flip-Flop

The D flip-flop shown in Figure 9.6 is a modification of the clocked SR flip-flop. The D input goes directly into the S input and the complement of the D input goes to the R input. The D input is sampled during the occurrence of a clock pulse. If it is 1, the flip-flop is switched to the set state (unless it was already set). If it is 0, the flip-flop switches to the clear state.

Figure 9.6: Clocked D Flip-Flop (*a*) Logic Diagram with NAND Gates (*b*) Graphical Symbol (*c*) Transition Table



(*a*)



(*b*)

D	Q	Q′
1	X	1
0	X	0

(*c*)

Notes

Self Assessment

Multiple choice questions:

1. The NAND gate output will be low if the two inputs are

   (*a*) 00 (*b*) 01

   (*c*) 10 (*d*) 11

2. A ring counter consisting of five Flip-Flops will have

   (*a*) 5 states (*b*) 10 states

   (*c*) 32 states (*d*) Infinite states

3. If the input to T Flip-Flop is 100 Hz signal, the final output of the three T Flip-Flops in cascade is:

   (*a*) 1000 Hz (*b*) 500 Hz

   (*c*) 333 Hz (*d*) 12.5 Hz

4. In a JK Flip-Flop, toggle means:

   (*a*) Set Q = 1 and Q = 0

   (*b*) Set Q = 0 and Q = 1

   (*c*) Change the output to the opposite state

   (*d*) No change in output

5. A 4-bit synchronous counter uses flip-flops with propagation delay times of 15 ns each. The maximum possible time required for change of state will be:

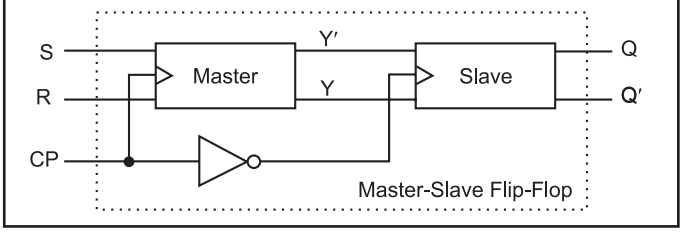
   (*a*) 15 ns (*b*) 30 ns

   (*c*) 45 ns (*d*) 60 ns

9.3 Master-Slave Flip-Flop

A master-slave flip-flop is constructed from two separate flip-flops. One circuit serves as a master and the other as a slave. The logic diagram of an SR flip-flop is shown in Figure 9.7. The master flip-flop is enabled on the positive edge of the clock pulse CP and the slave flip-flop is disabled by the inverter. The information at the external R and S inputs is transmitted to the master flip-flop. When the pulse returns to 0, the master flip-flop is disabled and the slave flip-flop is enabled. The slave flip-flop then goes to the same state as the master flip-flop.

Figure 9.7: Logic Diagram of a Master-Slave Flip-Flop

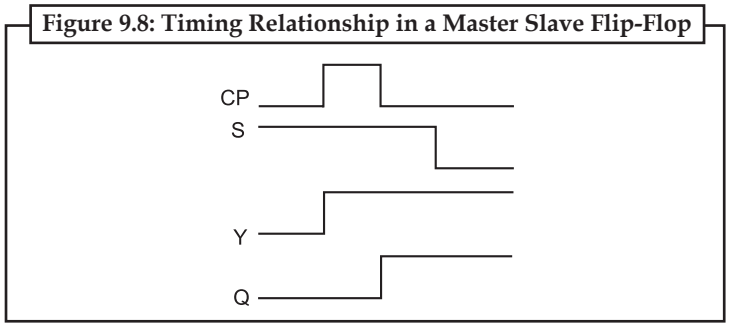


The timing relationship is shown in Figure 9.8 and is assumed that the flip-flop is in the clear state prior to the occurrence of the clock pulse. The output state of the master-slave flip-flop occurs on

Notes

the negative transition of the clock pulse. Some master-slave flip-flops change output state on the positive transition of the clock pulse by having an additional inverter between the CP terminal and the input of the master.

Figure 9.8: Timing Relationship in a Master Slave Flip-Flop



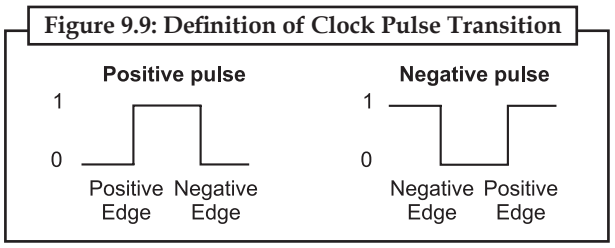
9.4 Triggering of Flip-Flop

The state of a flip-flop is changed by a momentary change in the input signal. This change is called a trigger and the transition it causes is said to trigger the flip-flop. The basic circuits of Figure 9.1 and Figure 9.2 require an input trigger defined by a change in signal level. This level must be returned to its initial level before a second trigger is applied. Clocked flip-flops are triggered by pulses.

The feedback path between the combinational circuit and memory elements in Figure 9.9 can produce instability if the outputs of the memory elements (flip-flops) are changing while the outputs of the combinational circuit that go to the flip-flop inputs are being sampled by the clock pulse. A way to solve the feedback timing problem is to make the flip-flop sensitive to the pulse transition rather than the pulse duration.

The clock pulse goes through two signal transitions: from 0 to 1 and the return from 1 to 0. As shown in Figure 9.9 the positive transition is defined as the positive edge and the negative transition as the negative edge.

Figure 9.9: Definition of Clock Pulse Transition



The clocked flip-flops already introduced are triggered during the positive edge of the pulse, and the state transition starts as soon as the pulse reaches the logic-1 level. If the other inputs change while the clock is still 1, a new output state may occur. If the flip-flop is made to respond to the positive (or negative) edge transition only, instead of the entire pulse duration, then the multiple-transition problem can be eliminated.

9.5 Timing Signal

(Computer science) A pulse generated by the clock of a digital computer to provide synchronization of its activities.

(Electronics) Any signal recorded simultaneously with data on magnetic tape for use in identifying the exact time of each recorded event.

Notes

A timing signal generating circuit including a clock source which generates clock pulses of a predetermined period, a binary counter which divides the frequency of the clock pulses from the clock source by n, a logical array which decodes an output of the binary counter and which is composed of semiconductor elements and flip-flop circuits which are set or reset by outputs of the logical array in response to the clock pulses from the clock source with the outputs of the flip-flop circuits being used as timing signals.

Timing analysis of all signals must be performed. In general, the signals fall into two classes. First, signals such as data, enables, and synchronous inputs must meet set-up (t_{SU}) and hold (t_H) times. Clock signals must meet width (t_W) requirements. These are well-defined and commonly used and will not be discussed further in this application note.

A common mistake in the analysis of digital systems is the timing analysis of asynchronous signals. With respect to flip-flops, these inputs are usually called PRESET and CLEAR. For MSI devices, there are other signals that are similar such as the JAM input to counters or shift registers. Although these signals are labeled "asynchronous" they do have restrictions on their use. Failure to meet the specifications can result in incorrect operation.

The first specification that these signals must meet is minimum pulse width. This is rather straightforward. If this specification is not met, the signal may be ignored or the flip-flop may be driven into a meta stable state.

A more subtle requirement, and the one that is commonly missed, is the removal time which is often referred to as tREM. Note that not all manufacturers use this nomenclature; there may be different terms. Essentially, this is the equivalent of a set up time requirement for synchronous inputs. That is, the asynchronous command must be removed at least tREM prior to the next active edge of the clock. As in the case above, if this specification is not met, the signal may be ignored or the flip-flop may be driven into a meta stable state.

Lastly, for MSI devices or modules that emulate them, data that is "jammed," as well as any related mode or address inputs, must meet set up and hold times with respect to the asynchronous pulse, similar to that of synchronous signals to the clock.

Some timing signals requirements of asynchronous inputs of flip-flops are similar to those of clocks; others to those of synchronous data. Specifically, the timing analysis of asynchronous signals must show:

1. Proper pulse width of the asynchronous pulse.
2. Removal of the asynchronous command pulse tREM prior to the next active edge of the clock. This is the equivalent to a setup time requirement.
3. Proper asynchronous data set-up and hold times with respect to the pulse.

Note that all manufacturers list in their device specification is not the tREM parameters (or its equivalent). However, all flip-flops do have this requirement; unfortunately not all data sheets specify it. The asynchronous inputs may not be removed asynchronously to the clock for reliable operation.

Did u know?

Flip-Flop was the first time an electronic circuit which had stored a "piece of information".

Notes

Case Study

16:1 MUX D Flip-Flop Circuit

100% Fault-Grade Vector Set

The following circuit was developed as a teaching circuit and as such has parameters and labels beyond what would appear in an actual circuit schematic. These parameters have nothing to do with the required Functional, AC Test or Parametric Vector sets.

A parametric gate-tree, used for VIH and VIL measurement is included and its output signal is listed. A simulation format requires that all I/O signals and internal enable nets be listed.

The test sequence for a 16:1 MUX ''was altered to allow clocking to occur between vector steps. The rule of one input per vector changing state is honored in that data and clock do not change in the same vector. The sequence begins after the circuit RESET is executed.

Both the schematic set and a formatted (compacted) output vector set are shown here. The output vectors include input, output and enable signals.

The Marquand Map

The Marquand Map for logical analysis was proposed in a mathematical paper in the late 1800's. It is a convenient mapping method for large functions. The Karnough Map was developed in the 1950's specifically for 4-variable circuits (for coding) and is messier to use in these cases. Figure (a) through Figure (c) show different sizes of Marquand Maps with minterms labeled.

2-Input 1-Output Marquand Map

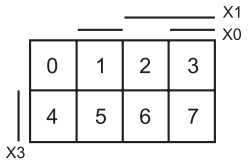


Figure (a): 2-Input 1-Output Marquand Map

3-Input 1-Output Marquand Map

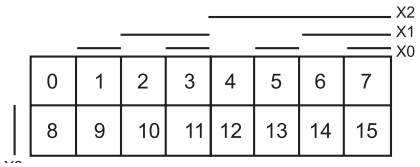


Figure (b): 3-Input 1-Output Marquand Map

4-Input 1-Output Marquand Map

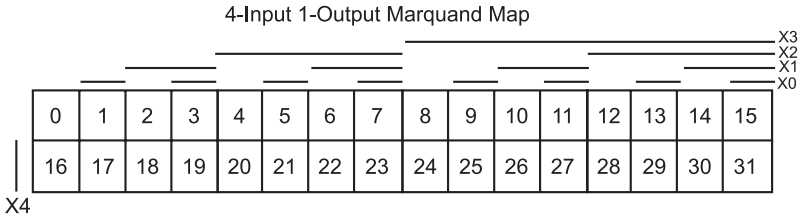


Figure (c): 4-Input 1-Output Marquand Map

Questions:

1. What is the Marquand Map? What is its use?
2. Draw a 5-Input 1-Output Marquand Map.

Notes

Self Assessment

Multiple choice questions:

6. How many Flip-Flops are required to construct a decade counter
   (*a*) 10 (*b*) 3
   (*c*) 4 (*d*) 2
7. How many Flip-Flops are required to construct mod-30 counter
   (*a*) 5 (*b*) 6
   (*c*) 4 (*d*) 8
8. For JK Flip-Flop with J = 1, K = 0, the output after clock pulse will be
   (*a*) 0 (*b*) 1
   (*c*) high impedance (*d*) no change
9. The output of SR Flip-Flop when S = 1, R = 0 is
   (*a*) 1 (*b*) 0
   (*c*) No change (*d*) High impedance

9.6 Summary

- A flip-flop circuit constructed from either two NAND gates or two NOR gates.
- State of a flip-flop is changed by a momentary change in the input signal. This change is called a trigger and the transition it causes is said to trigger the flip-flop.
- Master-slave flip-flop is constructed from two separate flip-flops. One circuit serves as a master and the other as a slave.
- The state of a flip-flop is changed by a momentary change in the input signal.
- A timing signal generating circuit including a clock source which generates clock pulses of a predetermined period, a binary counter which divides the frequency of the clock pulses from the clock source by n, a logical array which decodes an output of the binary counter.
- Timing analysis of all signals must be performed.

9.7 Keywords

***Clocked flip-flops*:** This is introduced triggered during the positive edge of the pulse, and the state transition starts as soon as the pulse reaches the logic-1 level.

***D flip-flop*:** It is a modification of the clocked SR flip-flop the D input goes directly into the S input and the complement of the D input goes to the R input.

***Flip-flops*:** It is not the sandals, but the logic gates, are the fundamental building blocks of sequential logic.

***JK flip-flop*:** It is a refinement of the SR flip-flop in that the indeterminate state of the SR type is defined in the JK type.

***Master-slave flip-flop*:** It occurs on the negative transition of the clock pulse some master-slave flip-flops change output state on the positive transition of the clock pulse by having an additional inverter between the CP terminal and the input of the master.

***SR flip-flop*:** It consists of a basic NOR flip-flop and two AND, gates the outputs of the two AND gates remain at 0 as long as the clock pulse (or CP) is 0, regardless of the S and R input values.

Notes

***T flip-flop*:** This is a single input version of the JK Flip-Flop the T flip-flop is obtained from the JK type if both inputs are tied together.

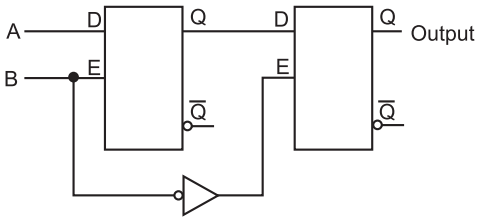
***Timing signal*:** It is generating device operates in response to angular displacement of a rotating shaft. The shaft has a disc provided with a reference mark and a plurality of timing marks apart from one another in a circumferential direction.

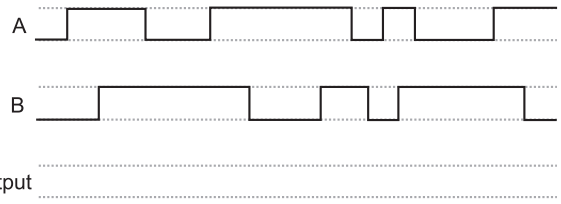
Lab Exercise

1. Prepare a truth table for JK flip-flop.
2. Prepare a truth table for master slave flip flop.

9.8 Review Questions

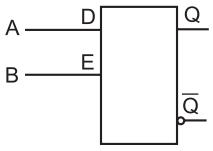
1. What do you understand by Flip-Flops? Explain with circuit diagram.
2. Which one the basic Flip-Flops (of SR, JK, D, T) and why?
3. Define the timing signal.
4. Define the triggering of Flip-Flop. Draw the truth table.
5. Describe the concept of master-slave Flip-Flop.
6. What is the difference between logic gate and Flip-Flop?
7. What is the use of clock in the Flip-Flop? Explain in brief.
8. Determine the final output states over time for the following circuit, built from D-type gated latches:



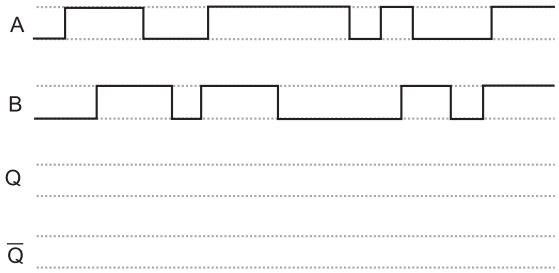


At what specific times in the pulse diagram does the final output assume the input's state? How does this behaviour differ from the normal response of a D-type latch?

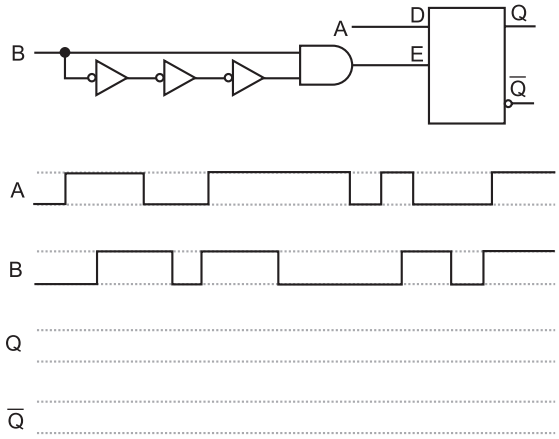
9. Determine the Q and $\overline{Q}$ output states of this D-type gated latch, given the following input conditions:



Notes

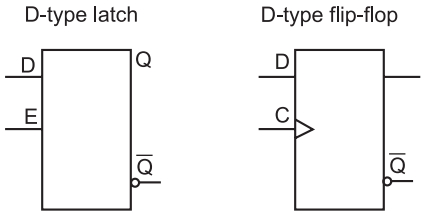


Now, suppose we add a propagation-delay-based one-shot circuit to the enable line of this D-type gated latch. Re-analyze the output of the circuit, given the same input conditions:



Comment on the differences between these two circuits' responses, especially with reference to the enabling input signal (B).

10. Shown here are two digital components: a D-type latch and a D-type flip-flop:



Other than the silly name, what distinguishes a "flip-flop" from a latch? How do the two circuits differ in function?

Answers to Self Assessment

1. (*d*) 2. (*a*) 3. (*d*) 4. (*c*) 5. (*a*)
6. (*c*) 7. (*a*) 8. (*b*) 9. (*a*)

9.9 Further Readings

Books

Digital Fundamentals, Ninth Edition, by Floyd Thomas L.

Digital Electronics – Principles and Applications, by Tokheim R.L.

Online link

http://books.google.co.in/books?

Notes

Unit 10: Clocked Sequential Circuits

CONTENTS

Objectives

After studying this unit, you will be able to:

- Explain the sequential circuits
- Discuss about the analysis of clocked sequential circuits
- Explain the sequential circuits design
- Explain the state minimization
- Explain the state assignment

Introduction

In digital electronics, a clocked sequential system is a system whose output depends only on the current state, whose state changes only when a global clock signal changes, and whose next-state depends only on the current state and the inputs.

Notes

Nearly, all digital electronic devices (microprocessors, digital clocks, mobile phones, cordless telephones, electronic calculators, etc.) are designed as clocked sequential systems. Notable exceptions include digital asynchronous logic systems.

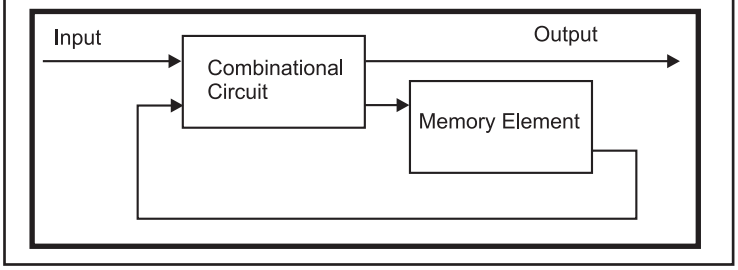
In particular, nearly all computers are designed as clocked sequential systems. Notable exceptions include analogue computers and clock less CPUs.

Typically each bit of the "state" is contained in its own flip-flop. Combinational logic decodes the state into the output signals. More combinational logic encodes the current state and the inputs into the next-state signals. The next-state signals are latched into the flip-flops under the control of the global clock signal (a wire connected to every flip-flop).

10.1 Sequential Circuits

Digital electronics is classified into combinational logic and sequential logic. Combinational logic output depends on the inputs levels, whereas sequential logic output depends on stored levels and also the input levels.

Figure 10.1: Combinational Logic and Sequential Logic



There are two types of input to the combinational logic: External inputs which come from outside the circuit design and are not controlled by the circuit; Internal inputs which a function of a previous output states are.

The internal inputs and outputs are referred to as "secondary" in the course notes. Secondary inputs are state variables produced by the storage elements, where as secondary outputs are excitations for the storage elements.

The memory elements are devices capable of storing binary info. The binary info stored in the memory elements at any given time defines the state of the sequential circuit. The input and the present state of the memory element determine the output. Memory elements next state is also a function of external inputs and present state. A sequential circuit is specified by a time sequence of inputs, outputs, and internal states.

There are two types of sequential circuits. Their classification depends on the timing of their signals:

- Synchronous sequential circuits
- Asynchronous sequential circuits

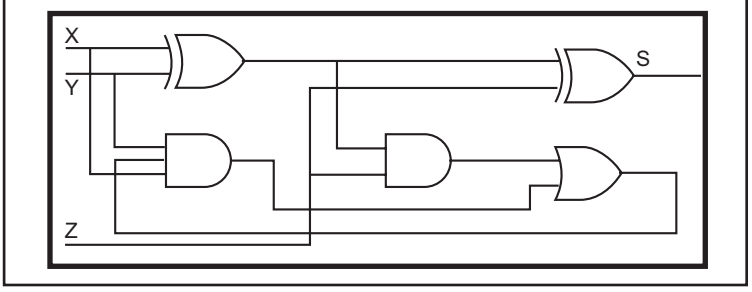
Asynchronous Sequential Circuit

This is a system whose outputs depend upon the order in which its input variables change and can be affected at any instant of time.

Gate-type asynchronous systems are basically combinational circuits with feedback paths. Because of the feedback among logic gates, the system may, at times, become unstable. Consequently they are not often used.

Notes

Figure 10.2: Asynchronous Sequential Circuits

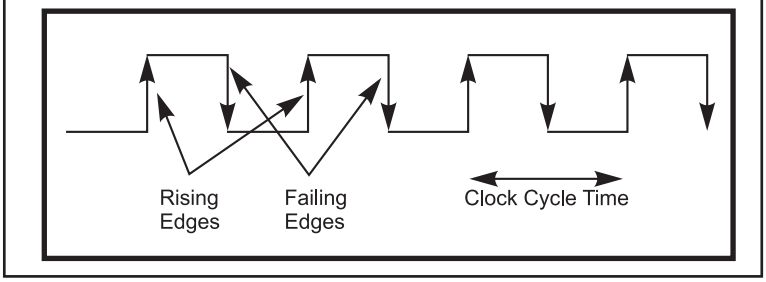


Did u know? When digital device changes its state, asynchronous sequential logic expresses memorizing effect by fixing moments of time.

Synchronous Sequential Circuits

This type of system uses storage elements called flip-flops that are employed to change their binary value only at discrete instants of time. Synchronous sequential circuits use logic gates and flip-flop storage devices. Sequential circuits have a clock signal as one of their inputs. All state transitions in such circuits occur only when the clock value is either 0 or 1 or happen at the rising or falling edges of the clock depending on the type of memory elements used in the circuit. Synchronization is achieved by a timing device called a clock pulse generator. Clock pulses are distributed throughout the system in such a way that the flip-flops are affected only with the arrival of the synchronization pulse. Synchronous sequential circuits that use clock pulses in the inputs are called clocked-sequential circuits. They are stable and their timing can easily be broken down into independent discrete steps, each of which is considered separately.

Figure 10.3: Synchronous Sequential Circuits



A clock signal is a periodic square wave that indefinitely switches from 0 to 1 and from 1 to 0 at fixed intervals. Clock cycle time or clock period: The time interval between two consecutive rising and falling edges of the clock.

Clock frequency = 1/clock cycle time (measured in cycles per second or Hz)

Example: Clock cycle time = 10ns clock frequency = 100 MHz

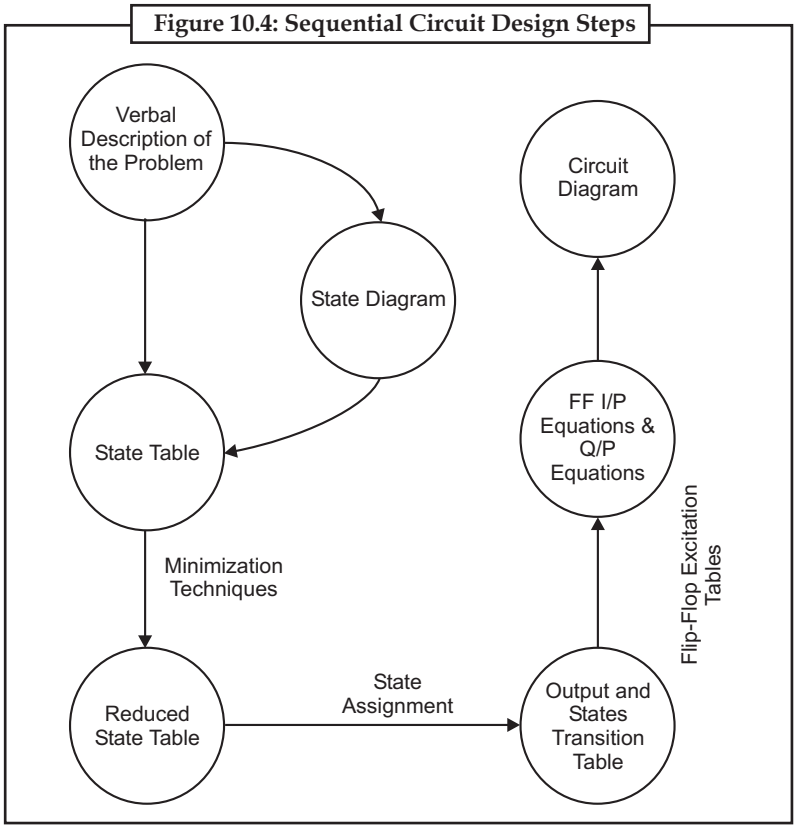
Did u know? The basic storage element in sequential logic is the flip-flop.

Notes

10.2 Analysis of Clocked Sequential Circuits

Analysis of clocked sequential circuits with an example

Figure 10.4: Sequential Circuit Design Steps



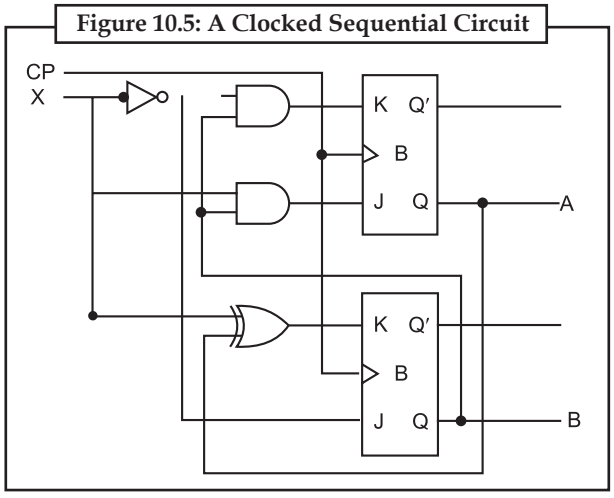
The behaviour of a sequential circuit is determined from the inputs, outputs and states of its flip-flops.

Both the outputs and the next state are a function of the inputs and the present state.

Recall from previous lesson that sequential circuit design involves the flow as shown.

Analysis consists of obtaining a state-table or a state-diagram from a given sequential circuit implementation. In other words analysis closes the loop by forming state-table from a given circuit-implementation. We will show the analysis procedure by deriving the state table of the example circuit we considered in synthesis. The circuit is shown in Figure 10.5.

Figure 10.5: A Clocked Sequential Circuit



Notes

The circuit has

- Clock input, CP.
- One input x
- One output y
- One clocked JK flip-flop
- One clocked D flip-flop (the machine can be in maximum of 4 states)

A State table is representation of sequence of inputs, outputs, and flip-flop states in a tabular form. Two forms of state tables are shown in Table 10.1, (the second form will be used).

Table 10.1: State Table Form 1

Present state		Next state				Output	
		$x = 0$		$x = 1$		$x = 0$	$x = 1$
A	B	A	B	A	B	y	y
0	0						
0	1						
1	0						
1	1						

Analysis is the generation of state table from the given sequential circuit.

The number of rows in the state table is equal to $2^{(\text{number of flip-flops+ number of inputs})}$. For the circuit under consideration, number of rows = $2^{(2+1)} = 2^{(3)} = 8$

Table 10.2: State Table Form 2

Present state		Input	Next state		Output
$A(t)$	$B(t)$	X	$A(t+1)$	$B(t+1)$	y
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

In the present case there are two flip-flops and one input, thus a total of 8 rows as shown in the Table 10.3.

Table 10.3: State Table

Present state		Input	Next state		Output
$A(t)$	$B(t)$	X	$A(t+1)$	$B(t+1)$	y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	1	0	0
0	1	1	0	1	0
1	0	0	0	0	0
1	0	1	1	1	0
1	1	0	1	0	0
1	1	1	0	1	1

Notes

The analysis can start from any arbitrary state. Let us start deriving the state table from the initial state 00. As a first step, the input equations to the flip-flops and to the combinational circuit must be obtained from the given logic diagram. These equations are:

$$J_A = BX'$$
$$K_A = BX + B'X'$$
$$D_B = X$$
$$y = ABX$$

The first row of the state-table is obtained as follows:

When input X = 0; and present states A = 0 and B = 0 (as in the first row);

then, using the above equations we get:

$$y = 0, J_A = 0, K_A = 1, \text{ and } D_B = 0.$$

The resulting state table is exactly same from which we started our design example. Thus, analysis is opposite to design and combined, and they act as a closed loop.

Did u know? A clocked sequential system is a kind of Moore machine, and a Moore machine is a finite-state machine whose output values are determined solely by its current state.

Task Draw a synchronous sequential logic circuit diagram.

Self Assessment

Multiple choice questions:

1. Sequential circuits have a clock signal as one of their ..... .
   (*a*) next-state (*b*) inputs
   (*c*) clock (*d*) outputs
2. The behaviour of a sequential circuit is determined from the ..... .
   (*a*) inputs (*b*) outputs
   (*c*) flip-flops (*d*) all of these
3. A state table is representation of sequence of ..... states in a tabular form.
   (*a*) inputs (*b*) outputs
   (*c*) flip-flops (*d*) All of these.
4. The number of rows in the state table is equal to..... .
   (*a*) $2^{(\text{number of flip-flops + number of inputs})}$
   (*b*) $2^{(\text{number of flip-flops - number of inputs})}$
   (*c*) $2^{(\text{number of flip-flops*number of inputs})}$
   (*d*) $2^{(\text{number of flip-flops/number of inputs})}$
5. The problem of ..... is to find ways of reducing the number of states in a sequential circuit without altering the input-output relationships.
   (*a*) unused states (*b*) states assignment
   (*c*) state reduction (*d*) states hazard

Notes

10.2.1 State Reduction

The problem of state reduction is to find ways of reducing the number of states in a sequential circuit without altering the input-output relationships.

In other words, to reduce the number of states, redundant states should be eliminated. A redundant state S_i is a state which is equivalent to another state S_j.

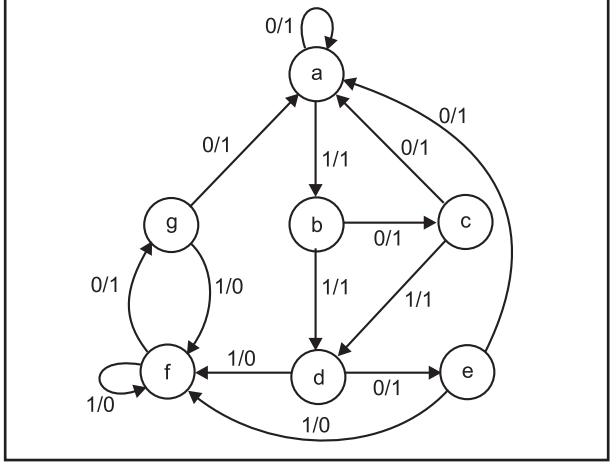
Two states are said to be equivalent if, for each member of the set of inputs, they give exactly the same output and send the circuit either to the same state or to an equivalent state.

Since, 'm' flip-flops can describe a state machine of up to 2^m states, reducing the number of states may (or may not) result in a reduction in the number of flip-flops. For example, if the number of states is reduced from 8 to 5, we still need 3 flip-flops.

However, state reduction will result in more do-not-care states. The increased numbers of do-not-care states can help obtain a simplified circuit for the state machine.

Consider the shown state diagram.

Figure 10.6: State Diagram



The state reduction proceeds by first tabulating the information of the state diagram into its equivalent state-table form (as shown in the table 10.4).

The problem of state reduction requires identifying equivalent states. Each N state is replaced by 1 state.

Consider the following state table.

States 'g' and 'e' produce the same outputs, i.e. '1' and '0', and take the state machine to same next-states, 'a' and 'f', on inputs '0' and '1' respectively. Thus, states 'g' and 'e' are equivalent states.

We can now remove state 'g' and replace it with 'e' as shown.

We next note that the above change has caused the states' and 'f' to be equivalent. Thus, in the next step, we remove state 'f' and replace it with 'd'.

There are no more equivalent states remaining. The reduced state table results in the following reduced state diagram.

Notes

Table 10.4: State Table After Reduction

Present state	Next state		Output	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
a	a	b	1	1
b	c	d	1	1
c	a	d	1	1
d	e	f d	1	0
e	a	f d	1	0
f	g e	f	1	0
g	a	f	1	0

10.2.2 States Assignment

When constructing a state diagram, variable names are used for states as the final number of states is not known a priori. Once the state diagram is constructed, prior to implementation (using gates and flip-flops), we need to perform the step of 'state reduction'. The step that follows state reduction is state assignment. In state assignment, binary patterns are assigned to state variables.

Table 10.5: Possible State Assignments

State	Assignment 1	Assignment 2	Assignment 3
a	001	000	000
b	010	010	100
c	011	011	010
d	100	101	101
e	101	111	011

For a given machine, there are several state assignments possible. Different state assignments may result in different combinational circuits of varying complexities.

State assignment procedures try to assign binary values to states such that the cost (complexity) of the combinational circuit is reduced. There are several heuristics that attempt to choose good state assignments (also known as state encoding) that try to reduce the required combinational logic complexity, and hence cost.

As mentioned earlier, for the reduced state machine obtained in the previous example, there can be a number of possible assignments. As an example, three different state assignments are shown in the Table 10.5 for the same machine.

We use ad-hoc state assignments in this lesson.

10.2.3 Design with Unused States

There are occasions when a sequential circuit, implemented using 'm' flip-flops, may not utilize all the possible 2^m states.

Table 10.6: Reduced Table with Binary Assignments

Present state	Next state		Output	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
a	a	b	1	1
b	c	d	1	1
c	a	d	1	1
d	e	d	1	0
e	a	d	1	0

Notes

In the previous example of machine with 5 states, we need three flip-flops. Let us choose assignment 1, which is binary assignment for our sequential machine example (shown in the Table 10.6).

The unspecified states can be used as do not-cares and will therefore help in simplifying the logic.

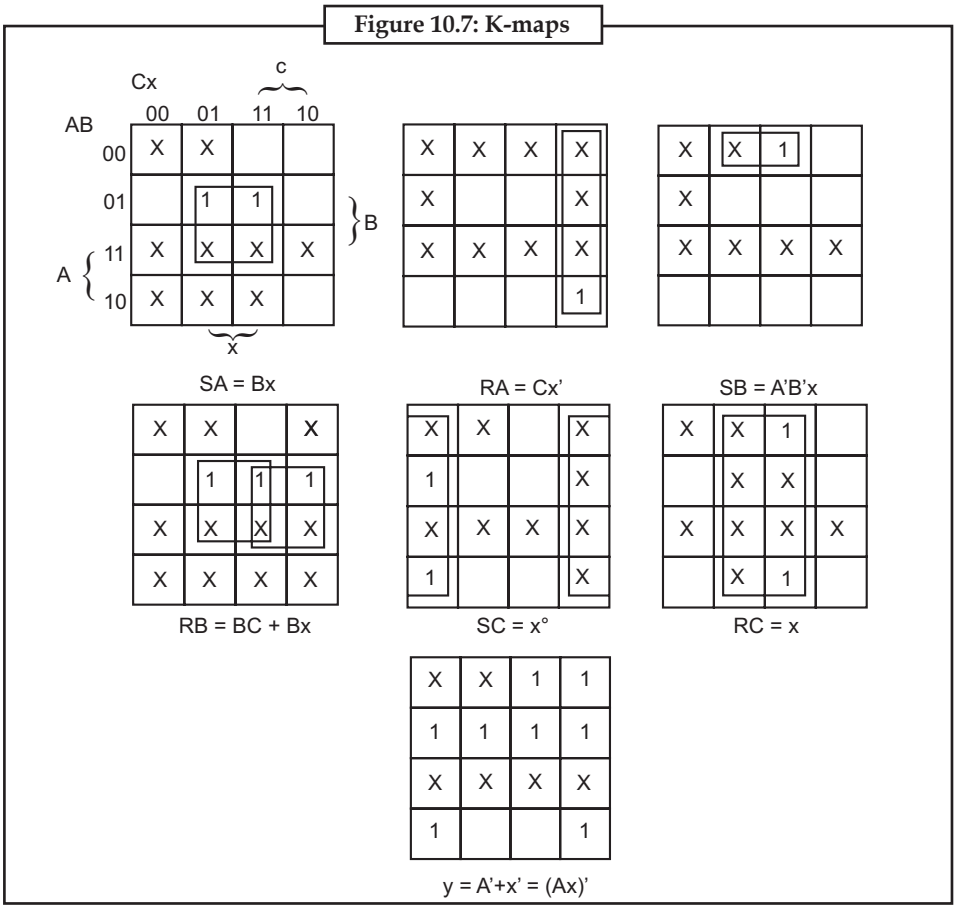
The excitation table of previous example is shown. There are three states, 000, 110, and 111 that are not listed in the Table 10.7 under present state and input.

Table 10.7: Excitation Table

Present state			Input	Next state			Flip-flop inputs						Output
A	*B*	*C*	*x*	*A*	*B*	*C*	*SA*	*RA*	*SB*	*RB*	*SC*	*RC*	*y*
0	0	1	0	0	0	1	0	X	0	X	X	0	1
0	0	1	1	0	0	1	0	X	1	0	0	1	1
0	1	0	0	0	1	0	0	X	X	0	1	0	1
0	1	0	1	0	1	0	1	0	0	1	0	X	1
0	1	1	0	0	1	1	0	X	0	1	X	0	1
0	1	1	1	0	1	1	1	0	0	1	0	1	1
1	0	0	0	1	0	0	X	0	0	X	1	0	1
1	0	0	1	1	0	0	X	0	0	X	0	X	0
1	0	1	0	1	0	1	0	1	0	X	X	0	1
1	0	1	1	1	0	1	X	0	0	X	0	1	0

With the inclusion of input 1 or 0, we obtain six do-not-care minterms: 0, 1, 12, 13, 14, and 15.

Figure 10.7: K-maps



Notes

The K-maps of SA and RA is shown in the Figure 10.7. Other K-maps can be obtained similarly and the equations derived are shown in the Figure 10.9.

The logic diagram thus obtained is shown in the Figure 10.8.

Figure 10.8: Logic Diagram

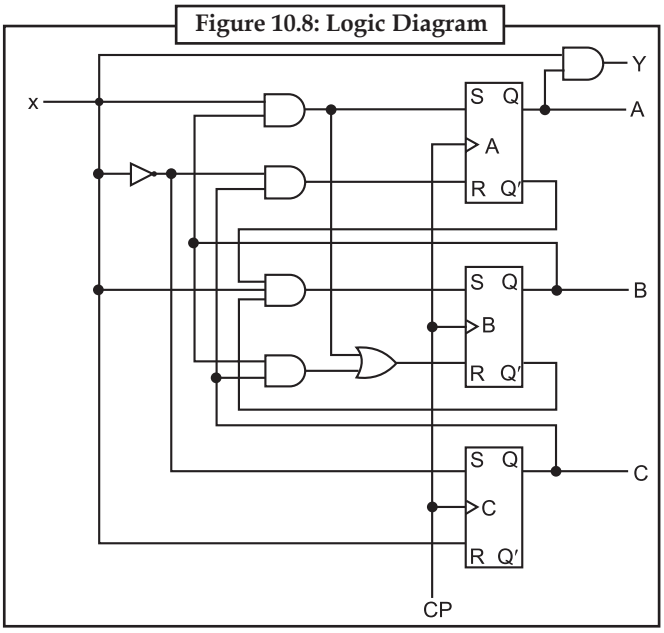


Figure 10.9: Equations

SA = Bx

RA = Cx'

SB = A'B'x

RB = BC + Bx

SC = x'

RC = x

y = A' + x' = (Ax)'

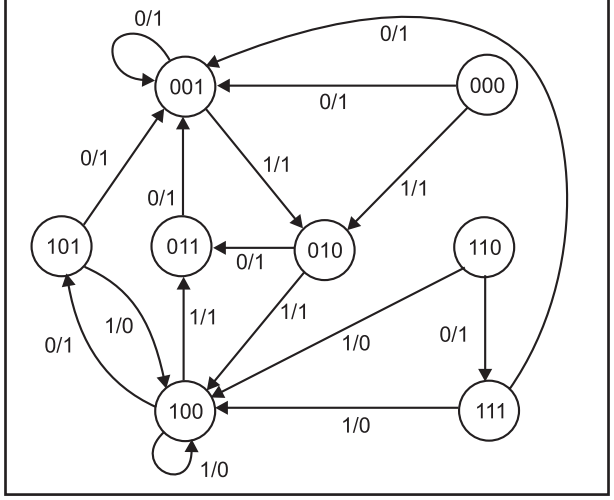
Note that the design of the sequential circuit is dependent on binary codes for states. A different binary state codes set may have resulted in some different combinational circuit.

10.2.4 Unused States Hazard

Sequential circuits with unused states can cause the circuit to produce erroneous behaviour. This may happen when the circuit enters one of the unused states due to some reason, e.g. due to power-on, and continues cycling between the invalid states.

Consider the circuit of the previous example that employed three unused states 000, 110 and 111. We will now investigate its behaviour if it enters in any of these states. The state diagram (from previous example) is shown in the Figure 10.10. We will use the state diagram to derive next state from each of the unused states and derive the state table.

Figure 10.10: State Diagram



Notes

For instance, the circuit enters unused state 000.

On application of input 0, ABCx = 0000, from the equations (Figure 10.9), we see that this minterm is not included in any function except for SC, i.e., the set input of flip-flop C and output y.

Thus, the circuit enters the state ABC = 001 from the unused state 000 when input 0 is applied.

On the other hand, if the input applied is 1 then ABCx combination = 0001. The maps indicate that this minterm is included in the functions for SB, RC and y.

Therefore, B will be set and C gets cleared.

So the circuit enters next state ABC = 010 when input 1 is applied to unused state 000.

Note that both states 001 and 010 are valid states.

Similar analysis is carried out for all other unused states and the derived state diagram is formed (shown in the Figure 10.10).

We note that the circuit converges into one of the valid states if it ever finds itself in one of the invalid states 000, 110, and 111.

Such a circuit is said to be self-correcting, free from hazards due to unused states.

Task Perform the state reduction to achieve fewer flip-flops.

10.3 Clocked Sequential Circuits Design

We saw in the combinational circuits section how to design a combinational circuit from the given problem. We convert the problem into a truth table, then draw K-map for the truth table, and then finally draw the gate level circuit for the problem. Similarly, we have a flow for the sequential circuit design. The steps are given below:

- Draw state diagram.
- Draw the state table (excitation table) for each output.
- Draw the K-map for each output.
- Draw the circuit.

Looks like sequential circuit design flow is very much the same as for combinational circuit.

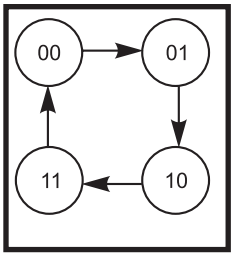
10.3.1 State Diagram

The state diagram is constructed using all the states of the sequential circuit in question. It builds up the relationship between various states and also shows how inputs affect the states.

To ease the following of the tutorial, let's consider designing the 2-bit up counter (Binary counter is one which counts a binary sequence) using the T flip-flop.

Below is the state diagram of the 2-bit binary counter.

Figure 10.11: 2-bit Binary Counter



Notes

10.3.2 State Table

The state table is the same as the excitation table of a flip-flop, i.e. what inputs need to be applied to get the required output. In other words this Table 10.8 gives the inputs required to produce the specific outputs.

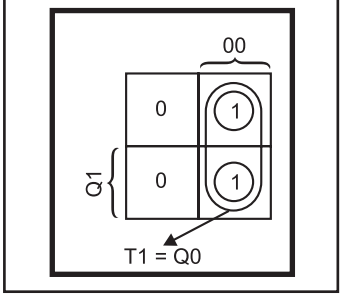
Table 10.8: State Table

Q1	Q0	Q1+	Q0+	T1	T0
0	0	0	1	0	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	0	0	1	1

10.3.3 K-map

The K-map is the same as the combinational circuits K-map. Only difference: we draw K-map for the inputs, i.e. T1 and T0 in the Table 10.8. From the Table 10.8 we deduct that we do not need to draw K-map for T0, as it is high for all the state combinations. But for T1 we need to draw the K-map as shown below, using SOP.

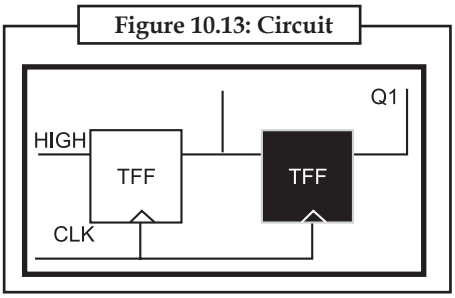
Figure 10.12: K-map



10.3.4 Circuit

There is nothing special in drawing the circuit; it is the same as any circuit drawing from K-map output. Below is the circuit of 2-bit up counter using the T flip-flop.

Figure 10.13: Circuit



Caution

Before design any circuit it must be carefully analyzed to ensure that it converges to some valid state.

10.4 State Minimization

Minimizing states is of interest because fewer states implies fewer flip-flops to implement the circuit (Complexity of combinational logic may also be reduced). Instead of trying to show which states are equivalent, it is often easier to show which states are definitely not equivalent (This can be exploited to define a minimization procedure). For simple FSMs, it is easy to see from the

Notes

state diagram that the number of states used is the minimum possible (FSMs for counters are good examples). For more complex FSMs, it is likely that an initial state diagram may have more states than are necessary to perform a required function.

10.4.1 State Equivalence

Definition: Two states Si and Sj are equivalent if and only if for every possible input sequence, the same output sequence will be produced regardless of whether Si or Sj is the initial state. If an input w = 0 is applied to an FSM in state Si and the FSM transitions to state Su, then Su is termed a 0-successor of Si. Similarly, if w = 1 and the FSM transitions to Sy, then Sy is a 1-successor of Si. The successors of Si are its k-successors. With one input, k can only be 0 or 1, but if there are multiple inputs then k represents all the valuations of the inputs.

10.4.2 Partitioning Minimization

Definition: A partition consists of one or more blocks, where each block comprises a subset of states that may be equivalent, but the states in one block are definitely not equivalent to the states in other blocks. From the equivalence definition, if Si and Sj are equivalent then their corresponding k-successors are also equivalent. Using this, we can construct a minimization procedure that involves considering the states of the machine as a set and then breaking that set into partitions that comprise subsets that are definitely not equivalent.

Example. of Partition minimization.

Consider the following state Table 10.9.

Table 10.9: State Table of Partition Minimization

Present state	Next state		Output
	w = 0,	w = 1	Z
A	B	C	1
B	D	F	1
C	F	E	0
D	B	G	1
E	F	C	0
F	E	D	0
G	F	G	0

An initial partition contains all the states in a single block P_1 = (ABCDEFG)

- The next partition separates the states that have different outputs P_2 = (ABD)(CEFG).
- Now, consider all 0- and 1-successors of the states in each block.
  - For (ABD), the 0-successors are (BDB).

    Since, these are all in the same block we must still consider A, B, and D equivalent.
  - The 1-successors of (ABD) are (CFG).

    We must still consider A, B, and D equivalent.
  - Now consider the (CEFG) block.
- P_2 = (ABD)(CEFG).
- For (CEFG), the 0-successors are (FEFF) which are all in the same block in P_2.
  - C, E, F, and G must still be considered equivalent.
- The 1-successors of (CEFG) are (ECDG).

Notes

- Since, these are not in the same block in P2 then at least one of the states in (CEFG) is not equivalent to the others.

  State F must be different from C, E and G because its 1- successor, D, is in a different block than E, C, and G.

- Therefore, P_3 = (ABD)(CEG)(F).
- At this point, we know that state F is unique since it is in a block by itself.
- P_3 = (ABD)(CEG)(F).
- The process repeats yielding the following:
  - 0-successors of (ABD) are (BDB).

    A, B and D are still considered equivalent.
  - 1-successors of (ABD) are (CFG), which are not in the same block,

    B cannot be equivalent to A and D since F is in a different block than C and G.
  - The 0- and 1-successors of (CEG) are (FFF) and (ECG)

    C, E, and G must still be considered equivalent.
- Thus, P_4 = (AD)(B)(CEG)(F).
- If we repeat the process to check the 0- and 1-successors of the blocks (AD) and (CEG), we find that
  - P_5 = (AD)(B)(CEG)(F).
- Since P_5 = P_4 and no new blocks are generated, it follows that states in each block are equivalent.
  - A and D are equivalent.
  - C, E, and G are equivalent.
- The state table can be rewritten, removing the rows for D, E and G and replacing all occurrences of D with A and all occurrences of E or G with C.
- The resulting state table is as follows:

Table 10.10: Resulting State Table

Present state	Next state		Output
	w = 0, w = 1		Z
A	B	C	1
B	A	F	1
C	F	C	0
F	C	A	0

Caution Always clock has low rise/fall times, then both pMOS and nMOS may conduct.

10.5 State Assignment

A 2n-state finite state machine has n!-different possible ways to assign state variable values to states. Thus, a 16-state finite state machine (which requires at least 4 state variables) has 24 (4!) different assignment of states to state variable values. A 1024-state finite state machine (which requires at least 10 state variables) has over 3 million (10!) state assignments. If the state is not densely encoded in the fewest number of bits, even more encodings are possible!

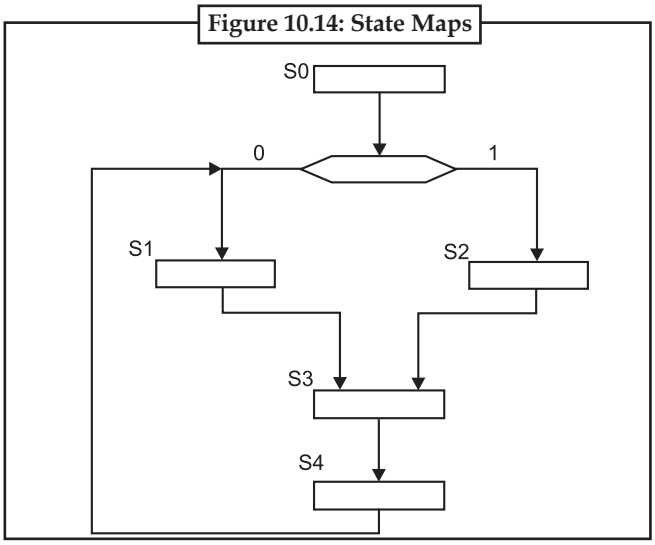
Notes

The number of gates needed to implement a sequential logic network is usually dependent upon the assignments of possible state variable values to states. Unfortunately, the only way to obtain the best possible assignment is to try every choice for encoding, which is tedious (but possible) for small state diagrams and effectively intractable for complex state machines. Luckily, heuristics have been developed that provide "reasonably good" state assignments "most of the time".

10.5.1 State Maps

State maps, similar in concept to k-maps, provide a means of observing adjacencies in the state assignments. The squares of the state maps are indexed by the binary values of the state bits; the state given that encoding is placed in the map square. Obviously the technique is limited to the situations in which k-maps can be used, that is, up to six variables.

Figure 10.14: State Maps



The above Figure 10.14 presents an ASM chart for the finite state machine.

Table 10.11: First State Assignment

State name	Q2	Q1	Q0
S0	0	0	0
S1	1	0	1
S2	1	1	1
S3	0	1	0
S4	0	1	1

Table 10.12: Second State Assignment

State name	Q2	Q1	Q0
S0	0	0	0
S1	0	0	1
S2	0	1	0
S3	0	1	1
S4	1	1	1

Notes

Table 10.13: First State Map

Q2 \ Q1Q0	00	01	11	10
0	S0		S4	S3
1		S1	S2	

Table 10.14: Second State Map

	Q1Q0 00	01	11	10
0	S0	S1	S3	S2
			S4	

The above Table 10.14 gives two alternative state assignments and their representations in the state maps.

10.5.2 Minimum-Bit-Change Strategy

The states are assigned in such a way that number of bit changes for all state transitions are minimized. For example, the assignment for the first Table 10.12 is not as good as the one in the second Table 10.13.

Table 10.15: Minimum-bit-change

Transition	First assignment bit changes	Second assignement bit changes
S0 to S1	2	1
S0 to S2	3	1
S1 to S3	3	1
S2 to S3	2	1
S3 to S4	1	1
S4 to S1	2	2

The first assignment leads to 13 different bit changes in the next state function, the second only 7-bit changes. We derive the first assignment completely in random and the second assignment with minimum transition distance in mind. We made the assignment for S0 first. Because of the way reset logic works, it usually makes sense to assign all zeros to the starting state. We make assignments for S1 and S2 next, placing them next to S0 because they are targets of transitions out of the starting state. Note how we used the edge adjacency of the state map. This is so we can place S3 between the assignments for S1 and S2, since it is the target of transitions from both of these states. Finally, we place S4 adjacent to S3, since it is the destination of S3's only transition. It would be perfect if S4 could also be placed distance 1 from S0, but it is not possible to do this and satisfy the other desired adjacencies. The resulting assignment exhibits only seven bit transitions, and perhaps an assignment that needs even fewer. The minimum bit change, although simple, it is not likely to achieve the best assignment (although it is often "good enough").

10.5.3 Prioritized Adjacency Strategy

Although the criterion of minimum transition distance is simple, it suffers by not considering the primary input and output values in determining the next state. A second set of guidelines makes an effort to consider this in the assignment of the states:

***Highest priority*:** States with the same next state for a given input transition should be given adjacent assignments in the state map.

Notes

***Medium priority*:** Next state of the same state should be given adjacent assignments in the state map.

***Lowest priority*:** States with the same output for a given input should be given adjacent assignments in the state map.

Figure 10.15: Highest Priority

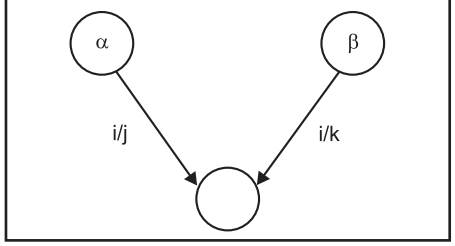


Figure 10.16: Medium Priority

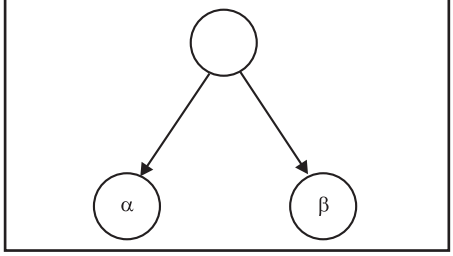
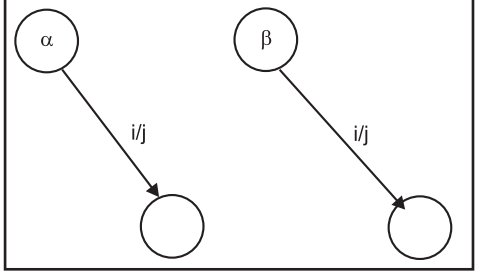


Figure 10 17: Lowest Priority



The guidelines, illustrated in the Figure 10.15, 10.16, and 10.17 above for the candidate states α and β, are ranked from highest to lowest priority. The first two rules attempt to group together ones in the next-state maps, while the third rule performs a similar grouping function for the output maps. The state assignments are done by listing all the state adjacencies implied by the guidelines, satisfying as many of these as possible.

Table10.16: The State Transition Table for a (non-minimal) Mealy Machine

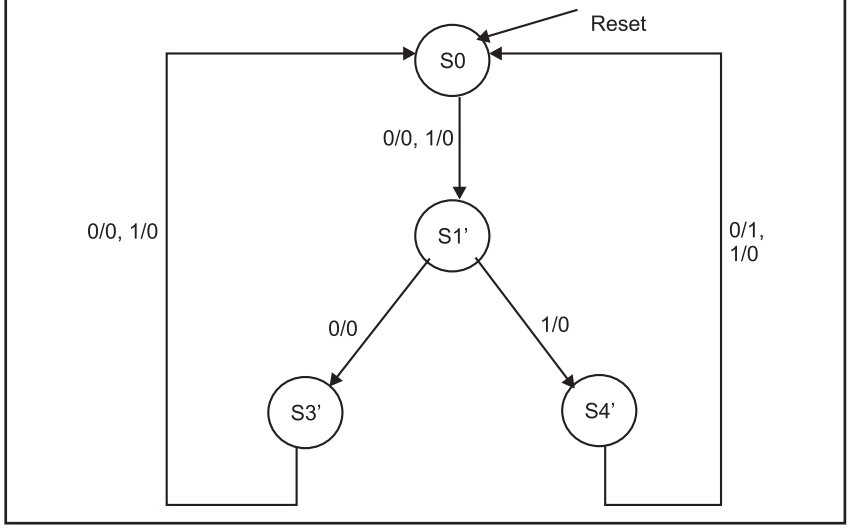
Input sequence	Present state	Next state		Output	
		X = 0	X = 1	X = 0	X = 1
Reset	S0	S1′	S1′	0	0
0 or 1	S1′	S3′	S4′	0	0
00 or 10	S3′	S0	S0	0	0
01 or 11	S4′	S0	S0	1	0

Notes

The corresponding state diagram is shown below:

Apply the state assignment guidelines to the state diagram shown below:

Figure 10.18: The State Assignment Guidelines to the State Diagram



The highest priority constraint for adjacent assignment applies to the states that share a common next state on the same input. In this case, states S3′ and S4′ both have S0 as their next state (under both input conditions). No other states share a common next state.

The medium priority assignment pairs states that have a common ancestor state. Again, S3′ and S4′ are the only states that fit this description.

The lowest priority assignments are made for states that have the same output behavior for a given input. S0, S1′ and S3′ all output 0 when the input is 0. Similarly, S0, S1′ S3′ and S4′ output 0 when the input is 1.

The constraints on the assignments can be summarized as follows:

Highest priority : (S3′, S4′);

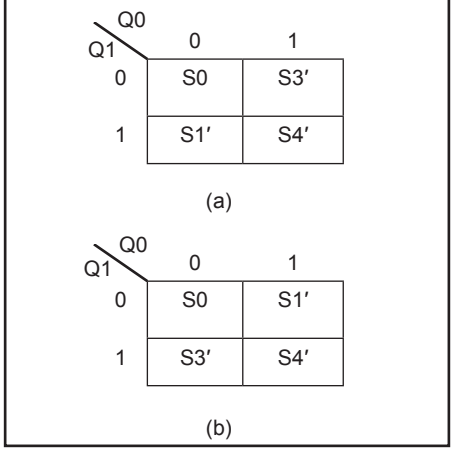
Medium priority : (S3′, S4′);

Lowest priority : 0/0: (S0, S1′, S3′);

1/0 : (S0, S1′, S3′, S4′);

Since the finite state machine has four states, we can make the assignment onto two state bits. In general, it is a good design practice to assign the reset state to state maps square 0.

Figure 10.19: State Maps

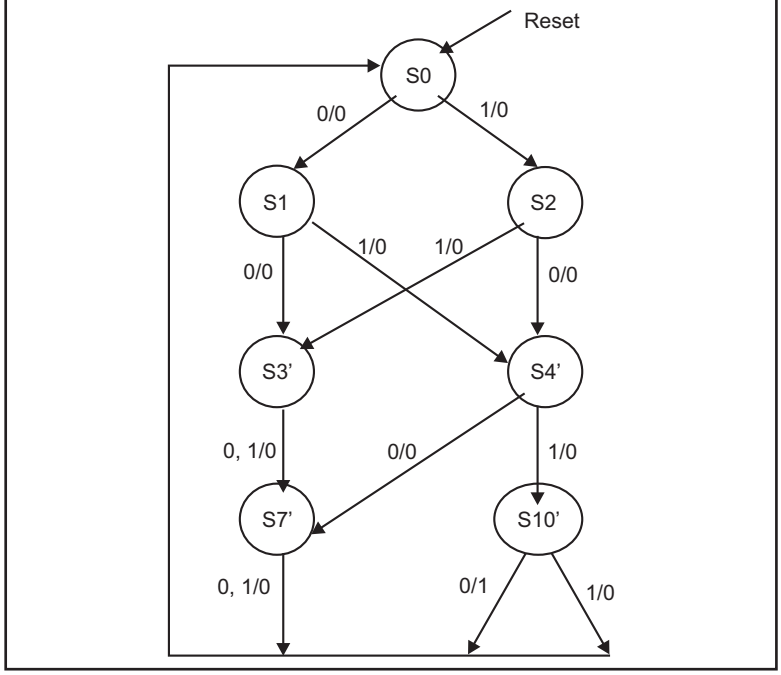


Notes

The Figure 10.19 shows two possible assignments. Both assign S0 to 00 and place S3′ and S4′ adjacent to each other.

Example: Let us consider more complicated case.

Figure 10.20: The State Diagram of the 4-bit String Recognizer



Applying the guidelines yields the following set of assignment constraints:

Highest priority : (S3′, S4′), (S7′, S10′);

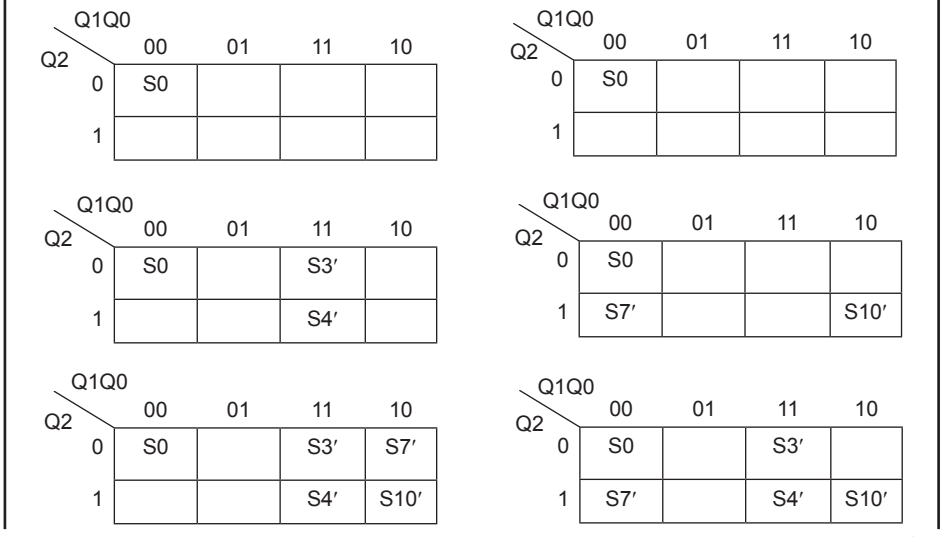
Medium priority : (S1, S2), (S3′, S4′), (S7′, S10′);

Lowest priority : 0/0: (S0, S1, S2, S3′, S4′, S7′);

1/0 : (S0, S1, S3′, S4′, S7′, S10′);

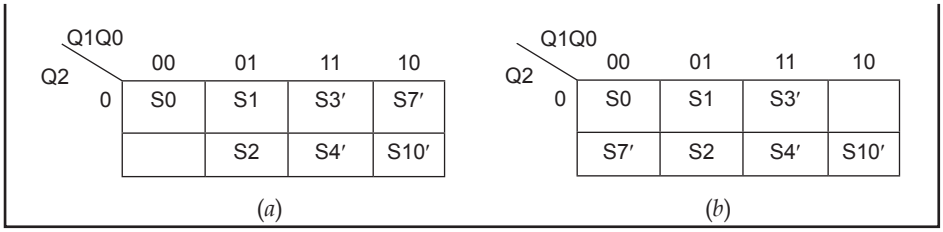
The Figure 10.20 shows two alternative assignments that meet most of these constraints:

Figure 10.21: Two Alternative Assignments



Contd...

Notes



We start with Figure 10.21(*a*) and first assign the reset state to the encoding for 0. Since, {S3′ S4′} is both a high priority and medium priority adjacency, we make their assignments next. S3′ is assigned 001 and S4′ is assigned 111. We assign {S7′, S10′} next because this pair also appears in the high and medium priority lists. We assign them the encodings 010 and 110, respectively. Besides giving them adjacent assignments this places S7 near S0, S3′, and S4′, which satisfies some of the lower priority adjacencies.

The final adjacency is {S1, S2}. We give them the assignments 001 and 101. This satisfies a medium priority placement as well as the lowest priority placements.

The second assignment is shown in Figure 10.21(b). We arrived at it by a similar line of reasoning, except that we assigned S7′ and S10′ the state's 100 and 110. The second assignment does about as good a job as the first, satisfying all the high and medium priority guidelines as well as most of the lowest priority ones.

Case Study

Electronic Flip-Flop

The first electronic flip-flop was invented in 1918 by William Eccles and F. W. Jordan. It was initially called the *Eccles–Jordan trigger circuit* and consisted of two active elements (vacuum tubes). Such circuits and their transistorized versions were common in computers even after the introduction of integrated circuits, though flip-flops made from logic gates are also common now.

Early flip-flops were known variously as trigger circuits or multivibrators. A multivibrator is a two-state circuit; they come in several varieties, based on whether each state is stable or not: an *astable multivibrator* is not stable in either state, so it acts as a relaxation oscillator; a *monostable multivibrator* makes a pulse while in the unstable state, then returns to the stable state, and is known as a *one-shot*; a *bistable multivibrator* has two stable states, and this is the one usually known as a flip-flop. However, this terminology has been somewhat variable, historically. For example:

- **1942 – multivibrator implies astable:** "The multivibrator circuit is somewhat similar to the flip-flop circuit, but the coupling from the anode of one valve to the grid of the other is by a condenser only, so that the coupling is not maintained in the steady state."
- **1942 – multivibrator as a particular flip-flop circuit:** "Such circuits were known as 'trigger' or 'flip-flop' circuits and were of very great importance. The earliest and best known of these circuits was the multivibrator."
- **1943 – flip-flop as one-shot pulse generator:** "It should be noted that an essential difference between the two-valve flip-flop and the multivibrator is that the flip-flop has one of the valves biased to cutoff."
- **1949 – monostable as flip-flop:** "Monostable multivibrators have also been called 'flip-flops'."

Contd...

Notes

- **1949 – monostable as flip-flop:** "... a flip-flop is a monostable multivibrator and the ordinary multivibrator is an astable multivibrator."

According to P. L. Lindley, a JPL engineer, the flip-flop types discussed below (RS, D, T, JK) were first discussed in a 1954 UCLA course on computer design by Montgomery Phister, and then appeared in his book *Logical Design of Digital Computers*. Lindley was at the time working at Hughes Aircraft under Dr. Eldred Nelson, who had coined the term JK for a flip-flop which changed states when both inputs were on. The other names were coined by Phister. They differ slightly from some of the definitions given below. Lindley explains that he heard the story of the JK flip-flop from Dr. Eldred Nelson, who is responsible for coining the term while working at Hughes Aircraft. Flip-flops in use at Hughes at the time were all of the type that came to be known as J-K. In designing a logical system, Dr. Nelson assigned letters to flip-flop inputs as follows: #1: A & B, #2: C & D, #3: E & F, #4: G & H, #5: J & K. Nelson used the notations "j-input" and "k-input" in a patent application filed in 1953.

Questions:

1. What is importance of electronic flip-flop?
2. How many types of flip-flop?

Self Assessment

Multiple choice questions:

6. Sequential circuits with ..... can cause the circuit to produce erroneous behaviour.
   (*a*) unused states (*b*) states assignment
   (*c*) state reduction (*d*) states hazard
7. The ..... is constructed using all the states of the sequential circuit in question.
   (*a*) state diagram (*b*) states assignment
   (*c*) state reduction (*d*) states hazard
8. Combinational logic decodes the state into the ..... signals.
   (*a*) next-state (*b*) input
   (*c*) clock (*d*) output

True or False:

9. The K-map is the same as the combinational circuits K-map.
   (*a*) True (*b*) False
10. In particular, nearly all computers are not designed as clocked sequential systems.
   (*a*) True (*b*) False

10.6 Summary

- Digital electronics is classified into combinational logic and sequential logic.
- A sequential circuit is specified by a time sequence of inputs, outputs, and internal states.
- Synchronization is achieved by a timing device called a clock pulse generator.
- A clock signal is a periodic square wave that indefinitely switches from 0 to 1 and from 1 to 0 at fixed intervals.

Notes

- The behaviour of a sequential circuit is determined from the inputs, outputs and states of its flip-flops.
- Analysis is the generation of state table from the given sequential circuit.

10.7 Keywords

***Combinational logic*:** Combinational logic is a type of digital logic which is implemented by Boolean circuits, where the output is a pure function of the present input only.

***Karnaugh map* (*K-map*):** A Karnaugh map is a grid-like representation of a truth table. It is really just another way of presenting a truth table, but the mode of presentation gives more insight.

***Sequential logic*:** Sequential logic is a type of logic circuit whose output depends not only on the present input but also on the history of the input.

***State diagram*:** A state diagram is a type of diagram used in computer science and related fields to describe the behaviour of systems.

***State table*:** A state table is the representation of sequence of inputs, outputs, and flip-flop states in a tabular form.

***State minimization*:** State minimization is the transformation of a given machine into an equivalent machine with no redundant states.

Lab Exercise

1. Draw the state diagram of ATM.
2. Draw a logical diagram for sequential circuit.

10.8 Review Questions

1. What are sequential circuits? Explain the circuit diagram.
2. Explain the types of sequential circuits?
3. What is the synchronous sequential circuit?
4. What is a state assignment?
5. Discuss the clocked sequential circuits. And draw the truth table.
6. What is state diagram?
7. How to use K-maps for clocked sequential circuits?
8. What is sequential circuits design?
9. Create a state diagram of the 2-bit binary counter.
10. What is the state minimization?

Answers to Self Assessment

1. (*b*) 2. (*d*) 3. (*d*) 4. (*a*) 5. (*c*)
6. (*a*) 7. (*a*) 8. (*d*) 9. (*a*) 10. (*b*)

Notes

10.9 Further Reading


Books

Digital Logic Design, by Brian Holdsworth, R. Clive Woods.


Online link

http://books.google.co.in/books?

Unit 11: Registers and Counters

Notes

CONTENTS

Objectives

After studying this unit, you will be able to:

- Explain the shift registers
- Discuss the ripple counter
- Discuss the synchronous counters

Notes

- Define the memory decoding
- Explain the semiconductor memories

Introduction

Shift registers, like counters, are a form of sequential logic. Sequential logic, unlike combinational logic is not only affected by the present inputs, but also, by the prior history. In other words, sequential logic remembers past events.

Shift registers produce a discrete delay of a digital signal or waveform. A waveform synchronized to a clock, a repeating square wave, is delayed by "n" discrete clock times, and where "n" is the number of shift register stages. Thus, a four stage shift register delays "data in" by four clocks to "data out". The stages in a shift register are delay stages, typically type "D" Flip-Flops or type "JK" Flip-flops.

Formerly, very long (several hundred stages) shift registers served as digital memory. This obsolete application is reminiscent of the acoustic mercury delay lines used as early computer memory.

Serial data transmission, over a distance of meters to kilometers, uses shift registers to convert parallel data to serial form. Serial data communications replaces many slow parallel data wires with a single serial high speed circuit.

Serial data over shorter distances of tens of centimeters, uses shift registers to get data into and out of microprocessors. Numerous peripherals, including analog to digital converters, digital to analog converters, display drivers, and memory, use shift registers to reduce the amount of wiring in circuit boards.

Some specialized counter circuits actually use shift registers to generate repeating waveforms. Longer shift registers, with the help of feedback generate patterns so long that they look like random noise, pseudo-noise.

Basic shift registers are classified by structure according to the following types:

- Serial-in/serial-out
- Parallel-in/serial-out
- Serial-in/parallel-out
- Universal parallel-in/parallel-out
- Ring counter

11.1 Shift Registers

Shift registers are a type of sequential logic circuit, mainly for storage of digital data. They are a group of flip-flops connected in a chain so that the output from one flip-flop becomes the input of the next flip-flop. Most of the registers possess no characteristic internal sequence of states.

All flip-flops are driven by a common clock, and all are set or reset simultaneously.

The basic types of shift registers are studied, such as Serial-In/Serial-Out, Serial-In/Parallel-Out, Parallel-In/Serial-Out, Parallel-In/Parallel-Out, bidirectional shift registers. A special form of counter—the shift register counter, is also introduced.

Register:

- A set of n flip-flops
- Each flip-flop stores one bit

Notes

- Two basic functions: data storage (Figure 11.2) and data movement (Figure 11.1)

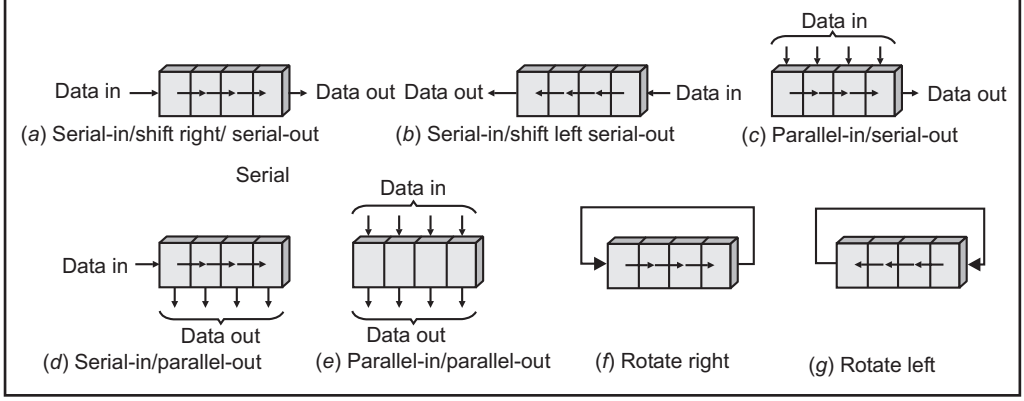
***Shift Register*:**

- A register that allows each of the flip-flops to pass the stored information to its adjacent neighbour
- Figure 11.1 shows the basic data movement in shift registers

Counter:

- A register that goes through a predetermined sequence of states

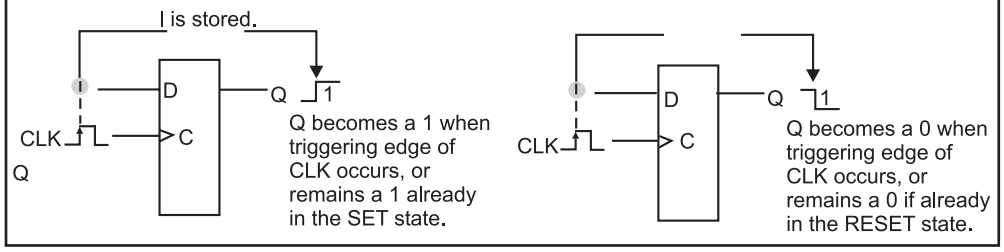
Figure 11.1: Basic Data Movement in Shift Registers



***Storage Capacity*:**

The storage capacity of a register is the total number of bits (1 or 0) of digital data it can retain. Each stage (flip-flop) in a shift register represents one bit of storage capacity. Therefore, the number of stages in a register determines its storage capacity.

Figure 11.2: The Flip-Flop as a Storage Element

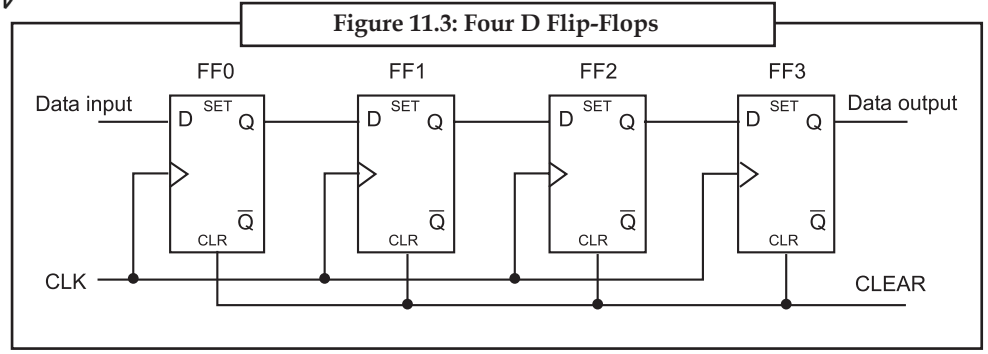


11.1.1 Serial-In/Serial-Out Shift Registers

The serial-in/serial-out shift register accepts data serially that is, one bit at a time on a single line. It produces the stored information on its output also in serial form.

Example: Basic four-bit shift register

Figure 11.3: Four D Flip-Flops



Notes

A basic four-bit shift register can be constructed using four D flip-flops, as shown in Figure 11.3.

The operation of the circuit is as follows:

- The register is first cleared, forcing all four outputs to zero.
- The input data is then applied sequentially to the D input of the first flip-flop on the left (FF0).
- During each clock pulse, one bit is transmitted from left to right.
- Assume a data word to be 1001.
- The least significant bit of the data has to be shifted through the register from FF0 to FF3.

In order to get the data out of the register, they must be shifted out serially. This can be done destructively or non-destructively. For destructive readout, the original data is lost and at the end of the read cycle, all flip-flops are reset to zero.

Table 11.1: Least Significant Bit of the Data

FF0	FF1	FF2	FF3	
0	0	0	0	1001

The data is loaded to the register when the control line is HIGH (i.e. WRITE). The data can be shifted out of the register when the control line is LOW (i.e. READ).

Table 11.2: HIGH and LOW Control Line

Clear	FF0	FF1	FF2	FF3
1001	0	0	0	0

WRITE:

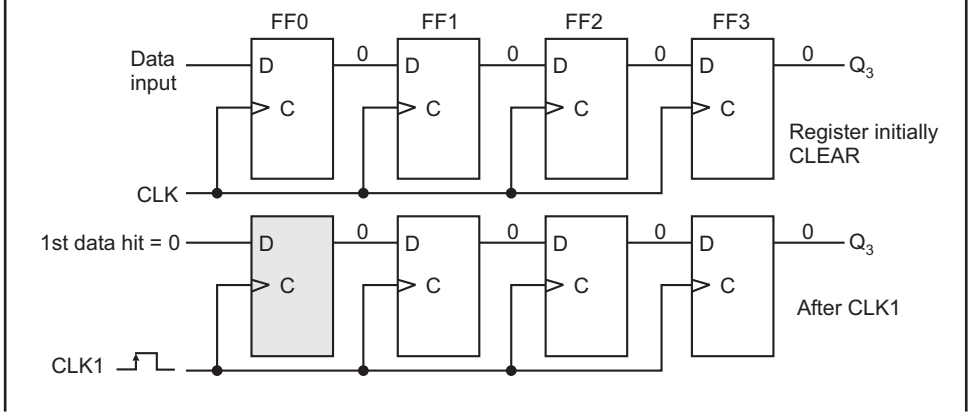
FF0	FF1	FF2	FF3	
1	0	0	1	0000

READ:

FF0	FF1	FF2	FF3	
1	0	0	1	1001

Figure 11.4 illustrates entry of the-four bits 1010 into the register. Figure 11.5 shows the four-bits (1010) being serially shifted out of the register and replaced by all zeros.

Figure 11.4: Four Bits (1010) Being Entered Serially into the Register



Notes

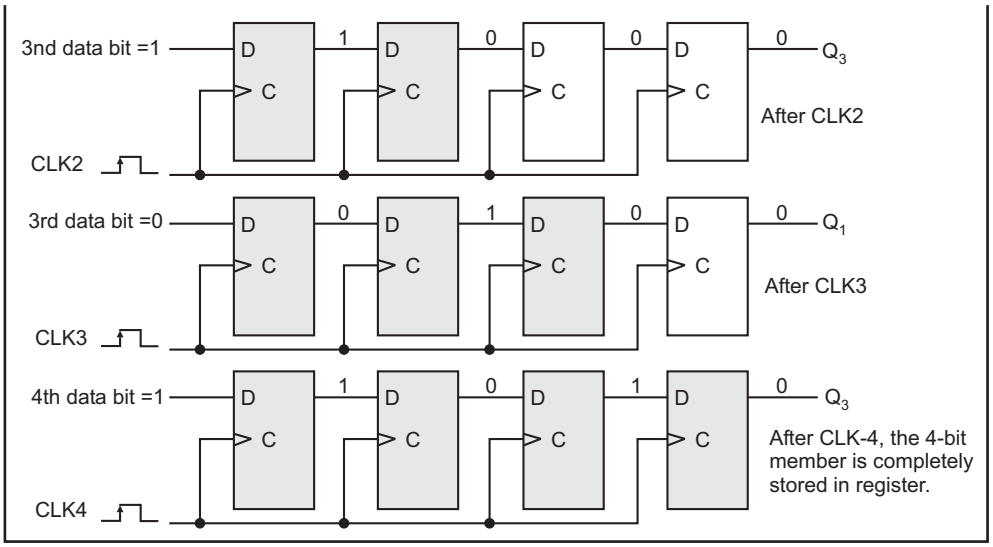
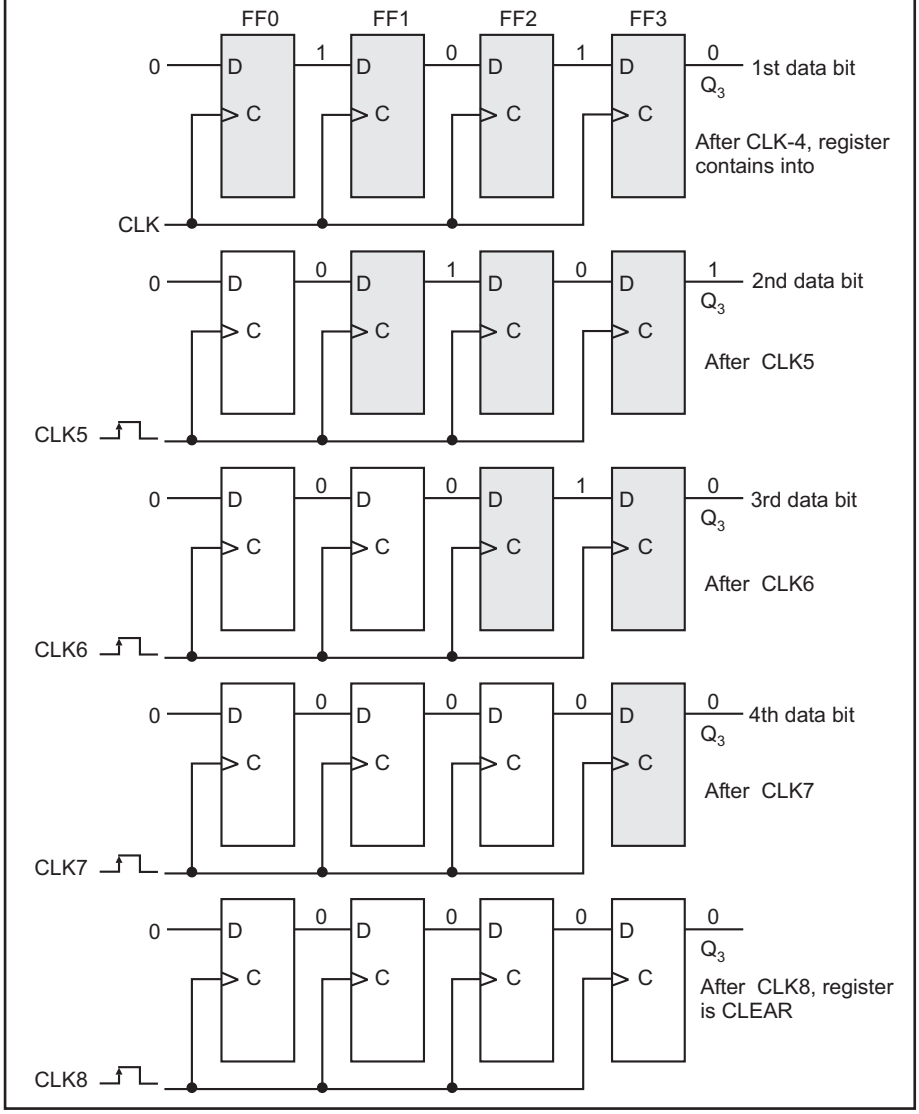


Figure 11.5: Four Bits (1010) Being Serially Shifted Out of the Register and Replaced by All Zeros

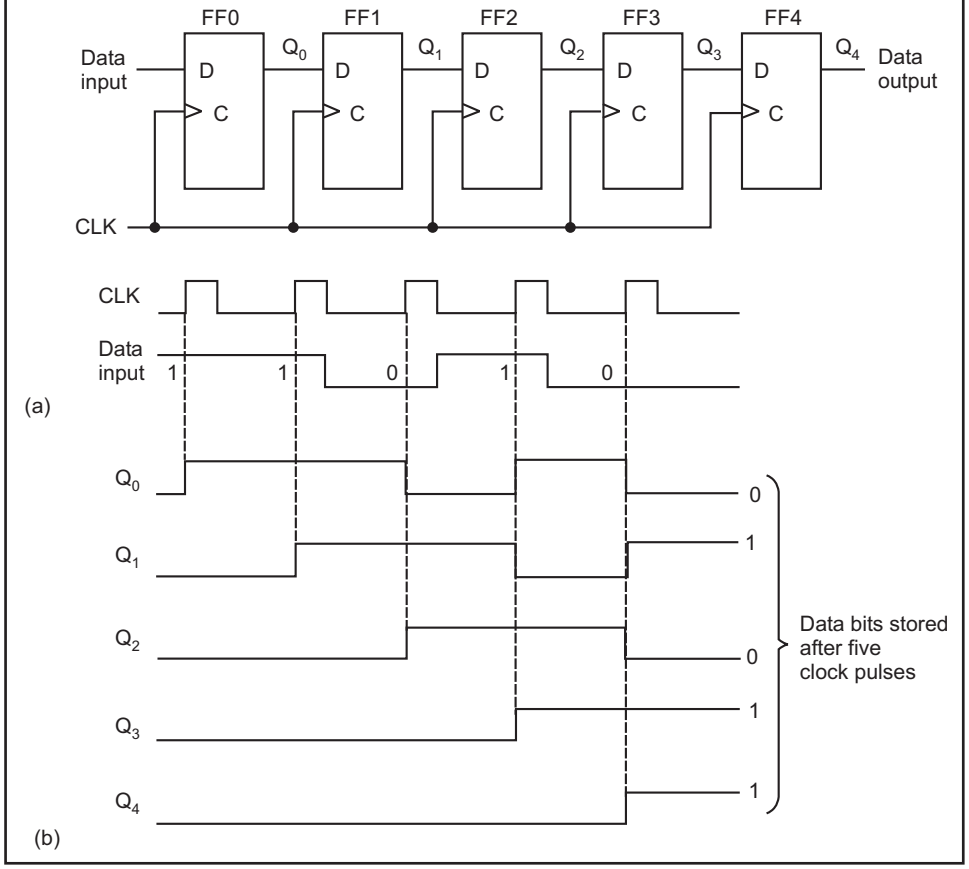


Notes

5-Bit Serial-In/Serial-Out Shift Registers

Figure 11.6 illustrates entry of the five-bits 11010 into the register.

Figure 11.6: Illustrates Entry of the Five Bits 11010 into the Register

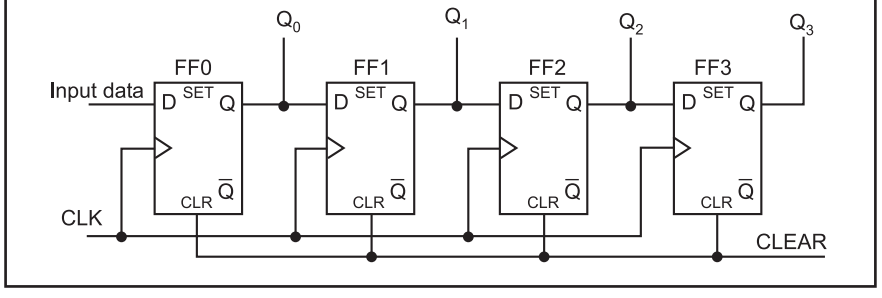


Task Design a 4-bit parallel-in serial-out register with synchronous loading of data, not loosing information at the stage of reading.

11.1.2 Serial-In/Parallel-Out Shift Registers

For this kind of register, data bits are entered serially in the same manner as discussed in the last section. The difference is the way in which the data bits are taken out of the register. Once the data are stored, each bit appears on its respective output line, and all bits are available simultaneously. A construction of a four-bit serial-in/parallel-out register is shown below:

Figure 11.7: Four-Bit Serial-in/Parallel-out Register



Notes

In the table below, we can see how the four-bit binary number 1001 is shifted to the Q outputs of the register.

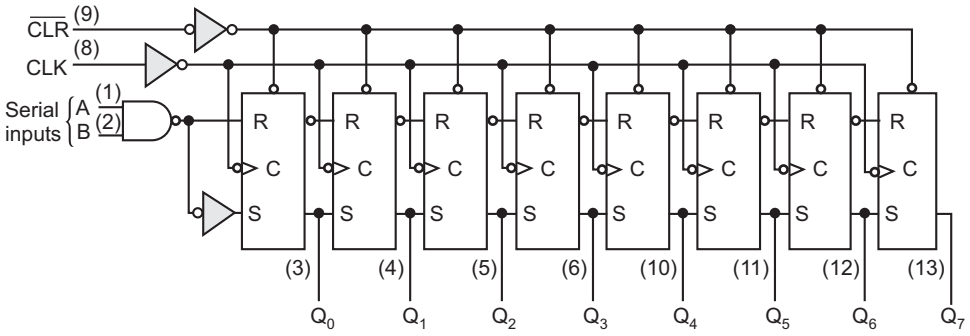
Table 11.3: Four-Bit Binary Number 1001 is Shifted to the Q Outputs of the Register

Clear	FF0	FF1	FF2	FF3
1001	0	0	0	0
	1	0	0	0
	0	1	0	0
	0	0	1	0
	1	0	0	1

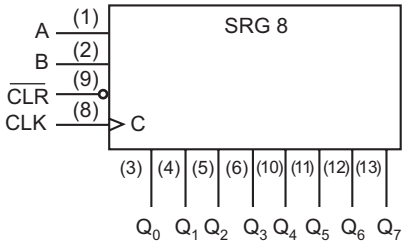
An 8-Bit Serial-In/Parallel-Out Shift Register (74hc164)

The 74HC164 is an example of an IC shift register having serial-in/parallel-out operation. The logic diagram and logic block are shown in Figure 11.8 (a), (b).

Figure 11.8: The Logic Diagram and Logic Block of 74HC164



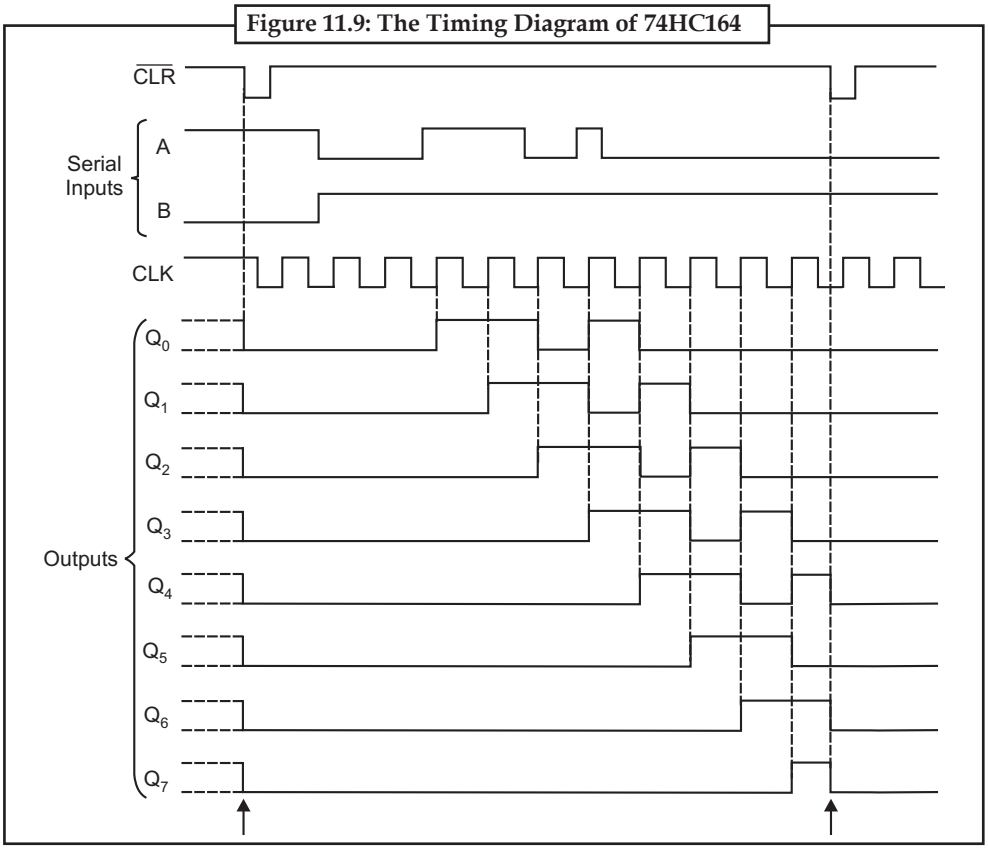
(*a*) Logic diagram



(*b*) Logic symbol

Notes

Figure 11.9: The Timing Diagram of 74HC164



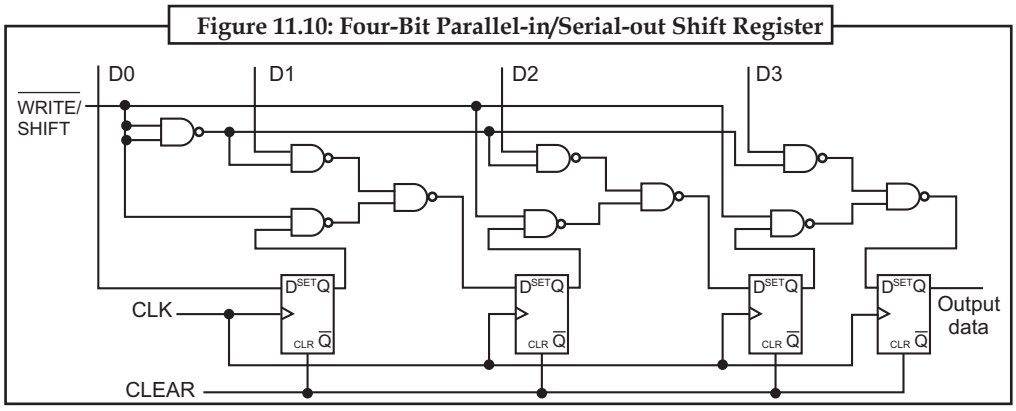
Caution

Take care while using common bus lines not to allow more than one device to be enabled at a time. System noise and incorrect data problems could result, and depending on output drive capability, physical damage to the device could occur.

11.1.3 Parallel-In/Serial-Out Shift Registers

A four-bit parallel-in/serial-out shift register is shown below. The circuit uses D flip-flops and NAND gates for entering data, (i.e. writing) to the register.

Figure 11.10: Four-Bit Parallel-in/Serial-out Shift Register



D0, D1, D2 and D3 are the parallel inputs, where D0 is the most significant bit and D3 is the least significant bit. To write data in, the mode control line is taken to LOW and the data is clocked in. The data can be shifted when the mode control line is HIGH as SHIFT is active high. The register performs right shift operation on the application of a clock pulse, as shown in the table 11.4.

Notes

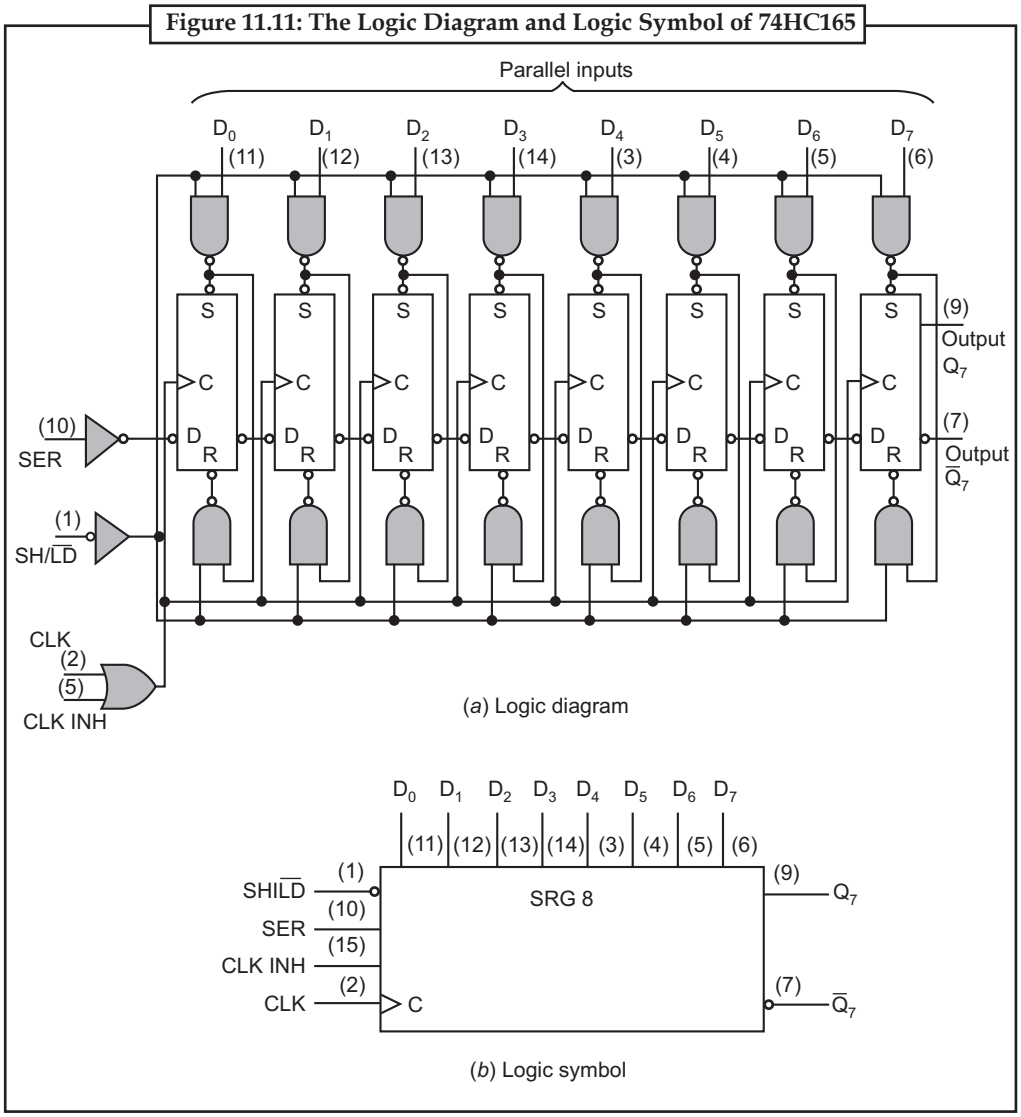
Table 11.4: The Register Performs Right Shift Operation

	Q_0	Q_1	Q_2	Q_3	
Clear	0	0	0	0	
Write	1	0	0	1	
Shift	1	0	0	1	
	1	1	0	0	1
	1	1	1	0	01
	1	1	1	1	001
	1	1	1	1	1001

8-bit Parallel-Load Shift Register (74HC165)

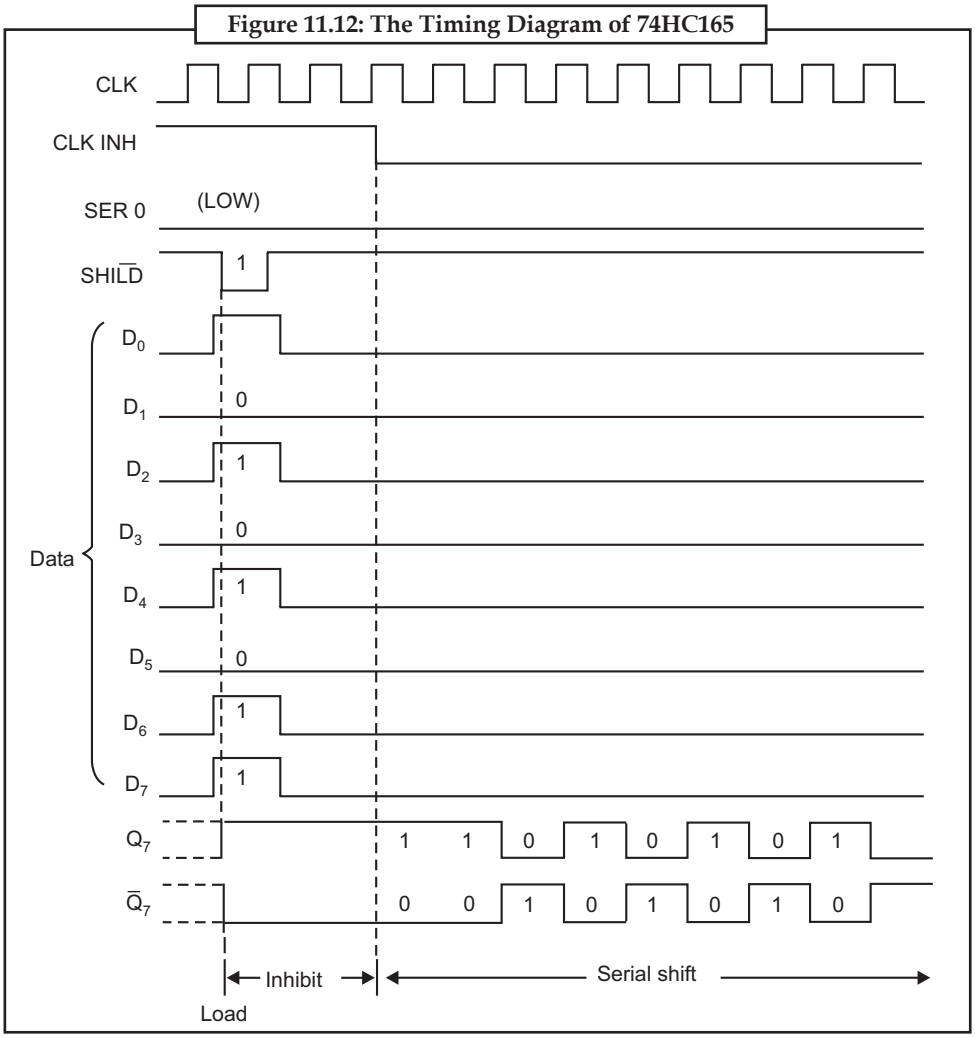
The 74HC165 is an example of an IC shift register that has a parallel-in/serial-out operation. It can also be operated as serial-in/serial-out. Figure 11.11 shows the logic diagram and logic symbol of 74HC165.

Figure 11.11: The Logic Diagram and Logic Symbol of 74HC165



Notes

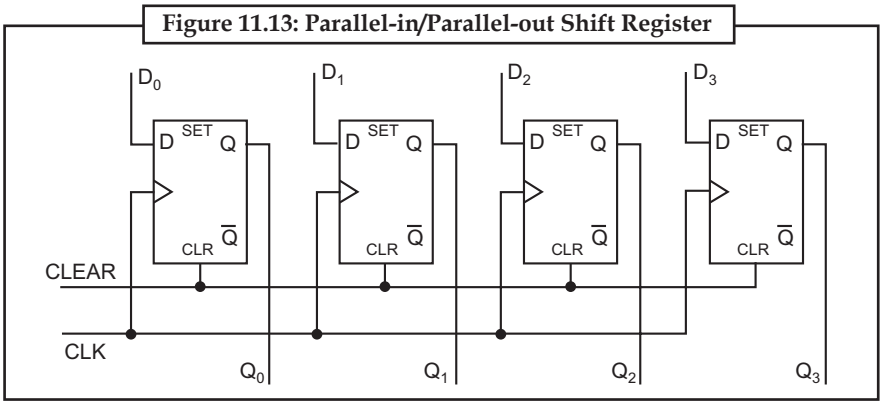
Figure 11.12: The Timing Diagram of 74HC165



11.1.5 Parallel-In/Parallel-Out Shift Registers

For parallel-in/parallel-out shift registers, all data bits appear on the parallel outputs immediately following the simultaneous entry of the data bits. The following circuit is a four-bit parallel-in/parallel-out shift register constructed by D flip-flops.

Figure 11.13: Parallel-in/Parallel-out Shift Register



The D's are the parallel inputs and the Q's are the parallel outputs. Once the register is clocked, all the data at the D inputs appear at the corresponding Q outputs simultaneously.

Notes

4-Bit Parallel-Access Shift Register (74hc195)

The 74HC195 can be used for parallel-in/parallel-out operation, serial-in/serial-out and serial-in/parallel-out operations. Q3 is the output when it is used for parallel-in/serial-out operation.

Figure 11.14: The 74LS195A 4-bit Parallel-Access Shift Register

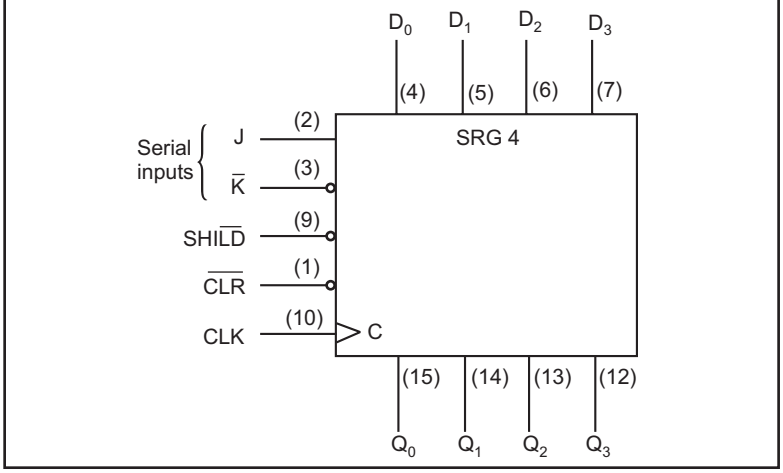
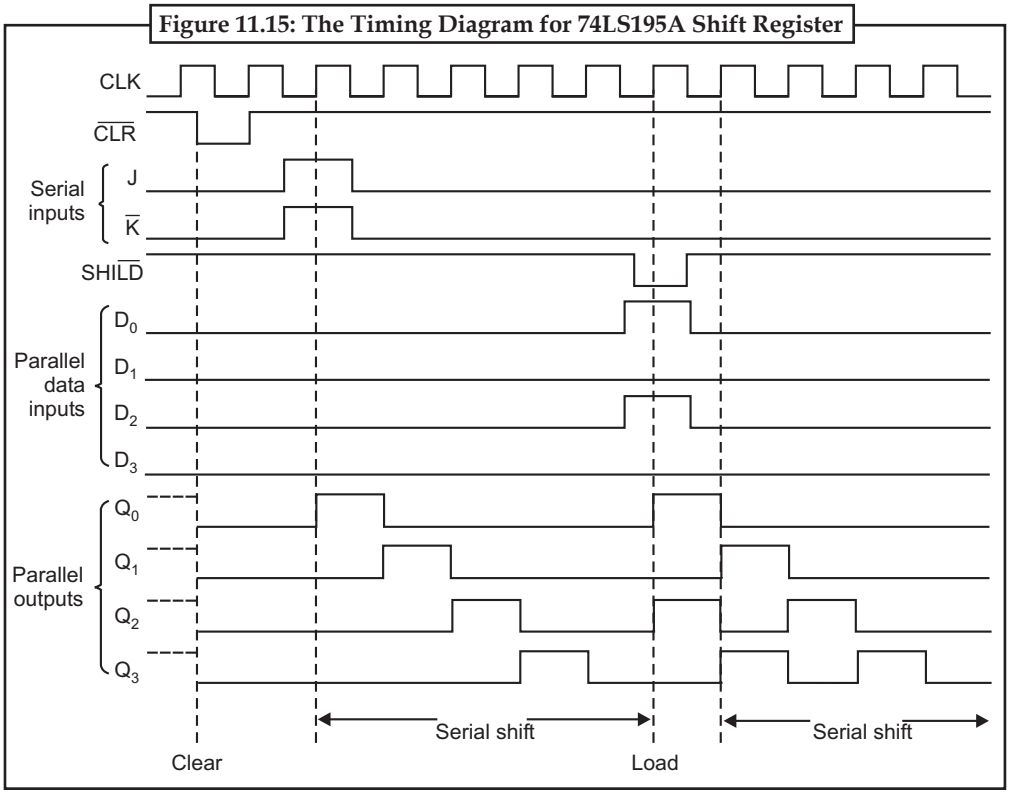


Figure 11.15: The Timing Diagram for 74LS195A Shift Register



Task Design a 4-bit shift register counter with the effect of "circling 1" providing it with synchronous self-correction mechanism.

11.1.6 Bidirectional Shift Registers

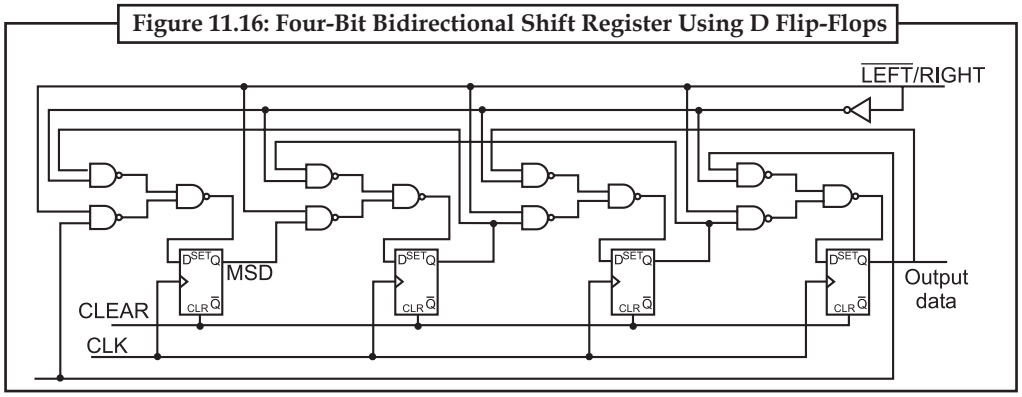
The registers discussed so far involved only right shift operations. Each right shift operation has the effect of successively dividing the binary number by two. If the operation is reversed (left

Notes

shift), this has the effect of multiplying the number by two. With suitable gating arrangement a serial shift register can perform both operations.

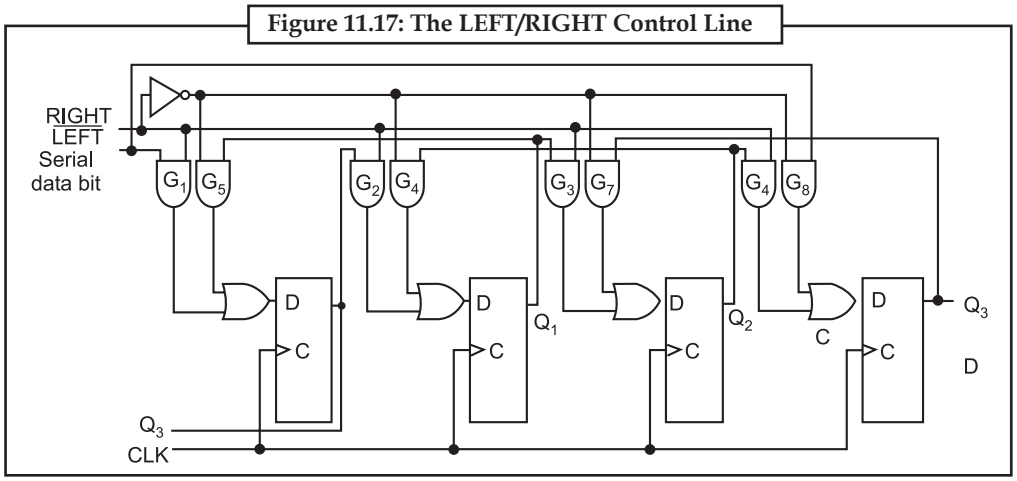
A bidirectional, or reversible, shift register is one in which the data can be shift either left or right. A four-bit bidirectional shift register using D flip-flops is shown in Figure 11.16.

Figure 11.16: Four-Bit Bidirectional Shift Register Using D Flip-Flops



Here a set of NAND gates are configured as OR gates to select data inputs from the right or left adjacent bistables, as selected by the LEFT/RIGHT control line.

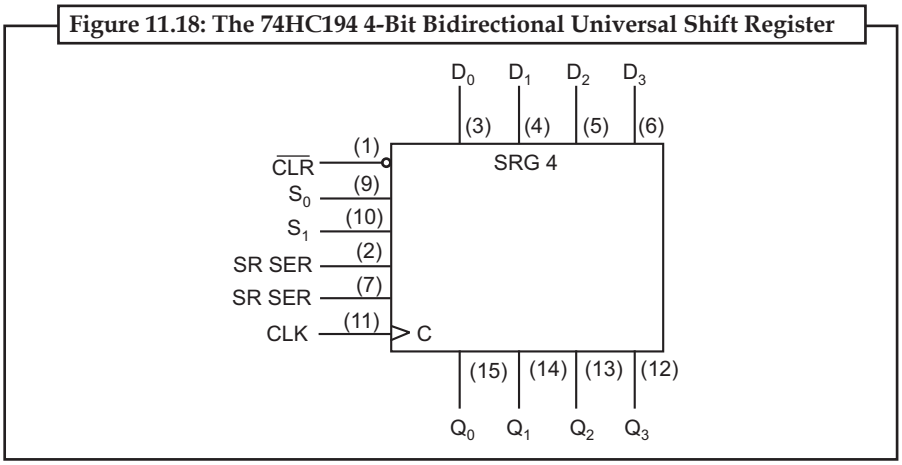
Figure 11.17: The LEFT/RIGHT Control Line



4-Bit Bidirectional Universal Shift Registers (74HC194)

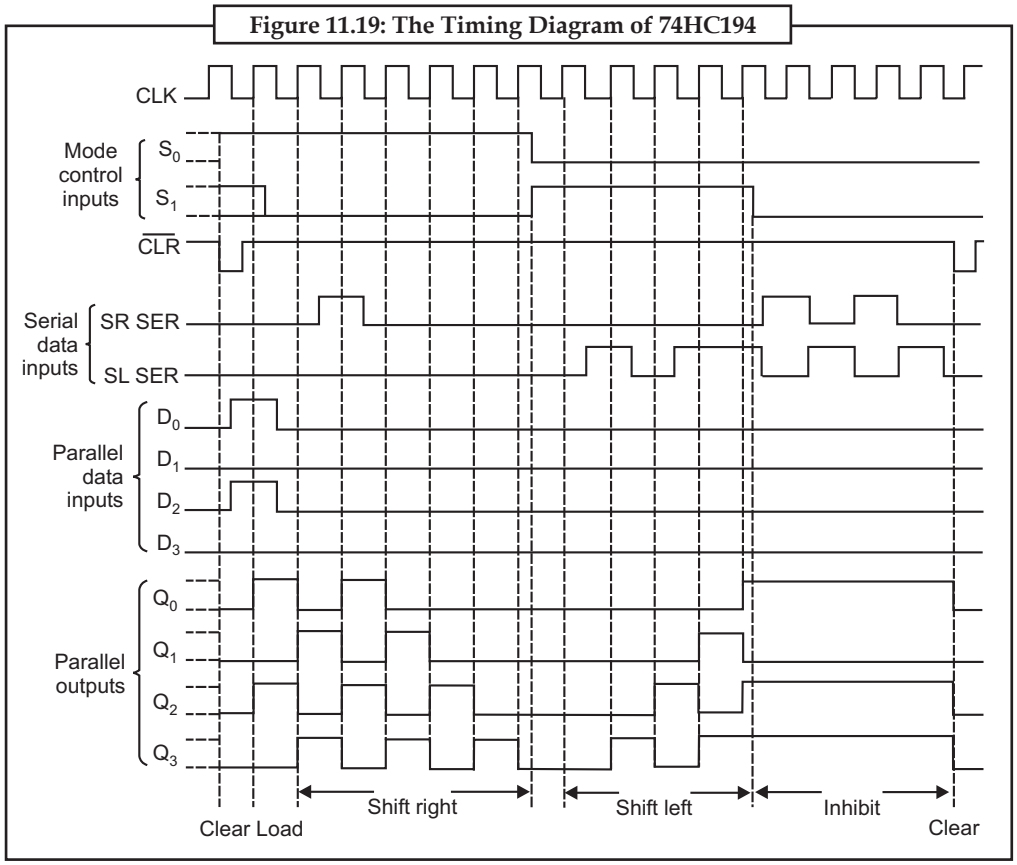
The 74HC194 is a universal bi-directional shift register. It has both serial and parallel input and output capability.

Figure 11.18: The 74HC194 4-Bit Bidirectional Universal Shift Register



Notes

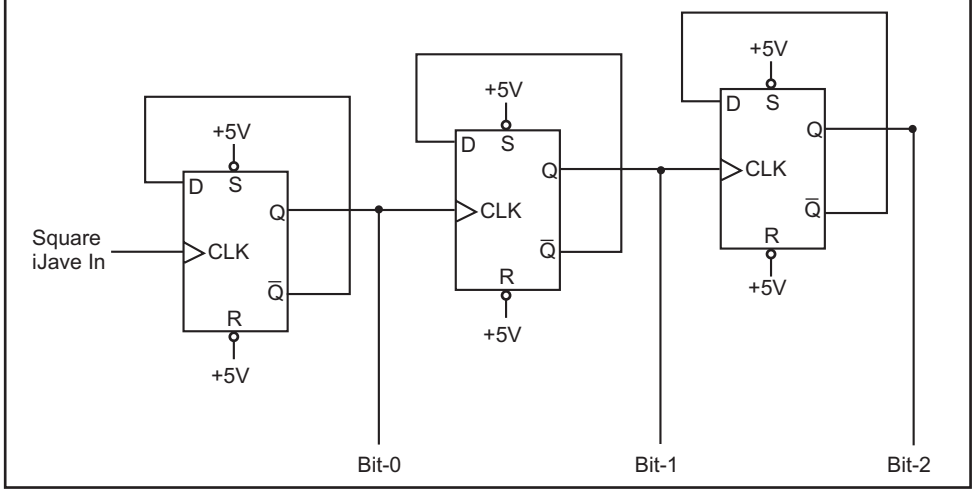
Figure 11.19: The Timing Diagram of 74HC194



11.2 Ripple Counter

A good first thought for making counters that can count higher is to chain Divide-by-2 counters together. We can feed the Q out of one flop into the CLK of the next stage. The result looks something like this:

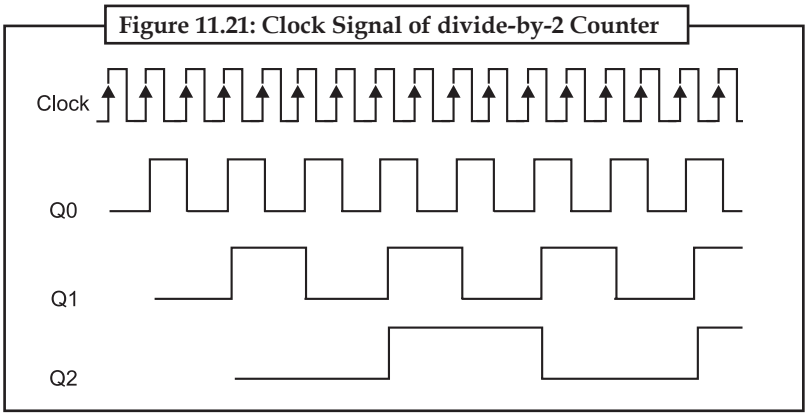
Figure 11.20: Chain Divide-by-2 Counters



The ripple counter is easy to understand. Each stage acts as a Divide-by-2 counter on the stage's signal. The Q out of each stage acts as both an output bit, and as the clock signal for the next stage.

Notes

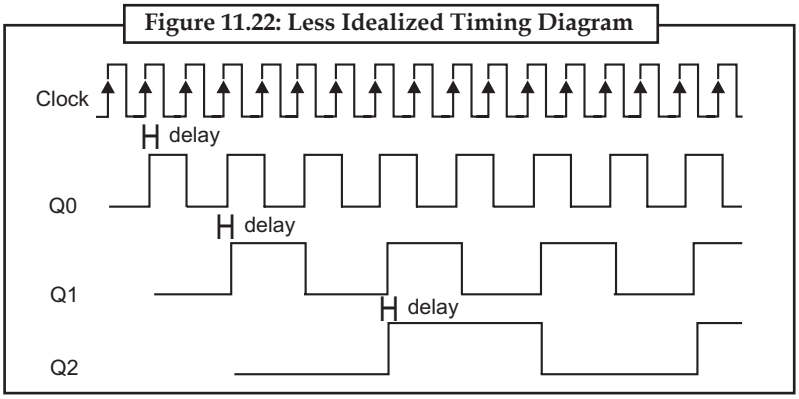
Figure 11.21: Clock Signal of divide-by-2 Counter



We can chain as many ripple counters together as we like. A three-bit ripple counter will count $2^3 = 8$ numbers, and an n-bit ripple counter will count 2^n numbers.

The problem with ripple counters is that each new stage put on the counter adds a delay. This propagation delay is seen when we look at a less idealized timing diagram:

Figure 11.22: Less Idealized Timing Diagram



Now, we can see that the propagation delay does not only slow down the counter, but it actually introduces errors into the system. These errors increase as we add additional stages to the ripple counter.

Caution

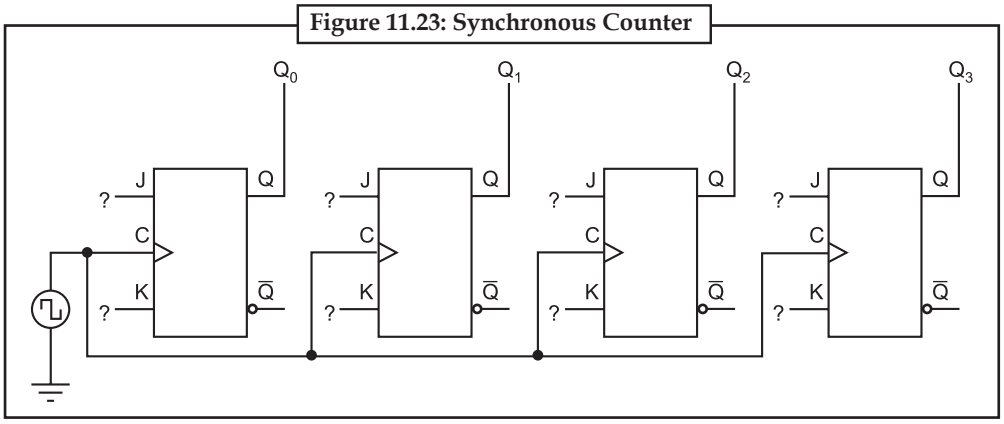
Be aware while using dynamic MOS shift registers (SR's) as low-speed power-saving circuits, since data can be lost if the SR clock speed is reduced instantaneously. The lower frequency limit applies only at high ambient temperatures and not at self-heated high-junction temperatures produced at high clock frequencies.

11.3 Synchronous Counters

A synchronous counter, in contrast to an asynchronous counter, is one whose output bits change state simultaneously, with no ripple. The only way we can build such a counter circuit from J-K flip-flops is to connect all the clock inputs together, so that each and every flip-flop receives the exact same clock pulse at the exact same time:

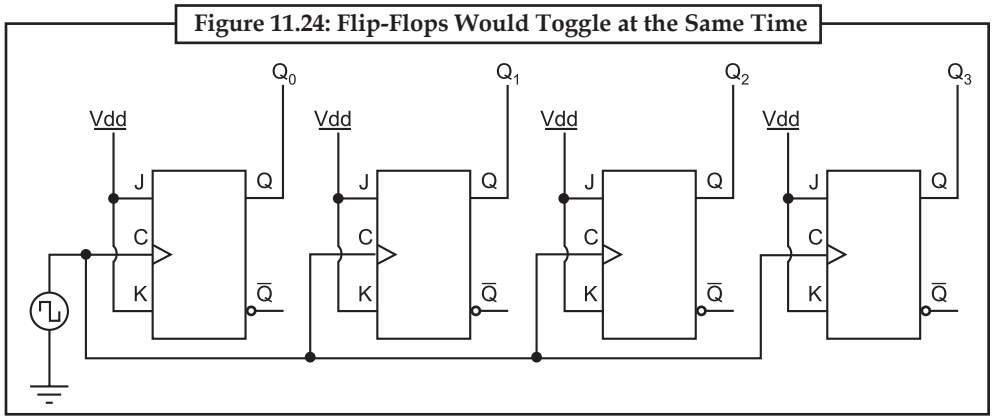
Notes

Figure 11.23: Synchronous Counter



Now, the question is, what do we do with the JK input? We know that we still have to maintain the same divide-by-two frequency pattern in order to count in a binary sequence, and that this pattern is best achieved utilizing the "toggle" mode of the flip-flop. So, the fact that the J and K inputs must both be (at times) "high" is clear. However, if we simply connect all the J and K inputs to the positive rail of the power supply as we did in the asynchronous circuit, this would clearly not work because all the flip-flops would toggle at the same time with each and every clock pulse!

Figure 11.24: Flip-Flops Would Toggle at the Same Time



Different types of Synchronous Counters:

11.3.1 4-bit Synchronous Binary Up Counter

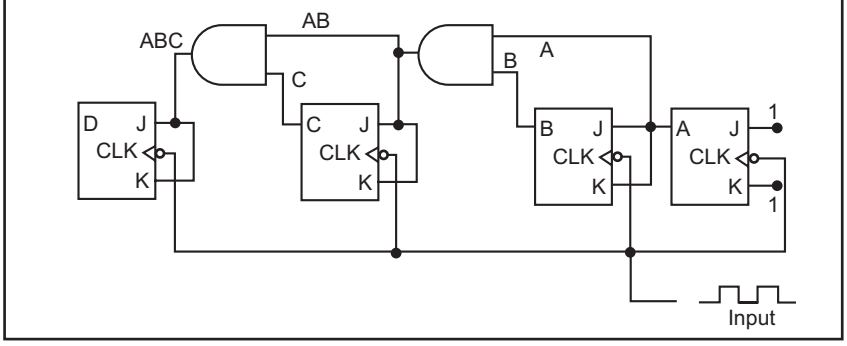
A synchronous binary counter counts from 0 to 2N-1, where N is the number of bits/flip-flops in the counter. Each flip-flop is used to represent one bit. The flip-flop in the lowest-order position is complemented/toggled with every clock pulse and a flip-flop in any other position is complemented on the next clock pulse provided all the bits in the lower-order positions are equal to 1.

Take for example $A_4\ A_3\ A_2\ A_1 = 0011$. On the next count, $A_4\ A_3\ A_2\ A_1 = 0100$. A1, the lowest-order bit, is always complemented. A_2 is complemented because all the lower-order positions (A_1 only in this case) are 1's. A_3 is also complemented because all the lower-order positions, A_2 and A_1 are 1's. But A_4 is not complemented the lower-order positions, $A_3\ A_2\ A_1 = 011$, do not give an all 1 condition.

To implement a synchronous counter, we need a flip-flop for every bit and an AND gate for every bit except the first and the last bit. The diagram (Fig. 11.25) shows the implementation of a 4-bit synchronous up-counter.

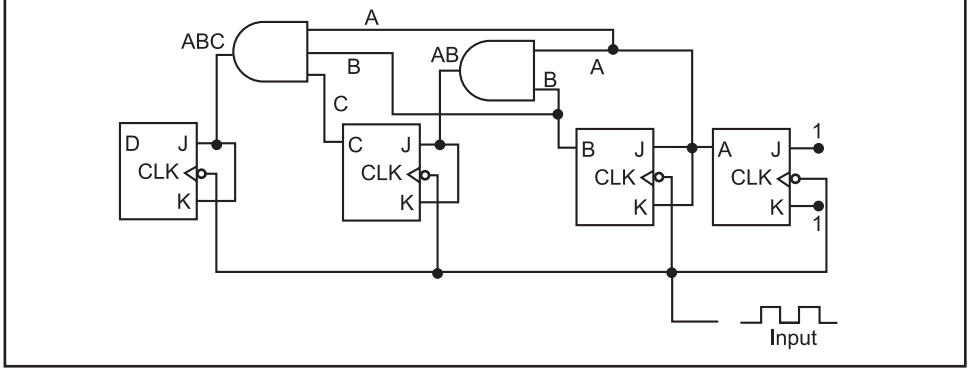
Notes

Figure11.25: 4-bit Synchronous Binary Up Counter



From the diagram above, we can see that although the counter is synchronous and is supposed to change simultaneously, we have a propagation delay through the AND gates which add up to give an overall propagation delay which is proportional to the number of bits of the counter. To overcome this problem, we can feed the outputs from the flip-flops directly to a many-input AND gate as follows:

Figure 11.26: 4-bit Synchronous Binary Up Counter Using Speedup Technique



This method does over comes the problem of additive propagation delay but introduces some other problem of its own. From the diagram above, we can see that the third flip-flop gets its J-K input from the output of a 2-input AND gate and the fourth flip-flop gets its input from a 3-input AND gate and so on. If we have a counter that counts to, for example, 16-bits, we will need to have:

1 * 15-input AND gate,

1 * 14-input AND gate,

...

...

1 * 3-input AND gate and

1 * 2-input AND gate.

This method obviously uses a lot more resources than the first method. Not only that, in the first method, the output from each flip-flop is only used as an input to one AND gate. In the second method, the output from each flip-flop is used as an input to all the higher-order bits. If we have a 12-bit counter, the output of the first flip-flop will have to drive 10 gates (called fan-out). The output from the flip-flop may not have the power to do this.

The "solution" to this is to use a compromise between the two methods. Say we have a 12-bit counter; we can organize it into 3 groups of 4. Within each group of 4, we use the second method and between the 3 groups, use the first method. This way, we only have an overall gate propagation delay and a maximum fan-out of 3 instead of 10 using the first and second method respectively.

Notes

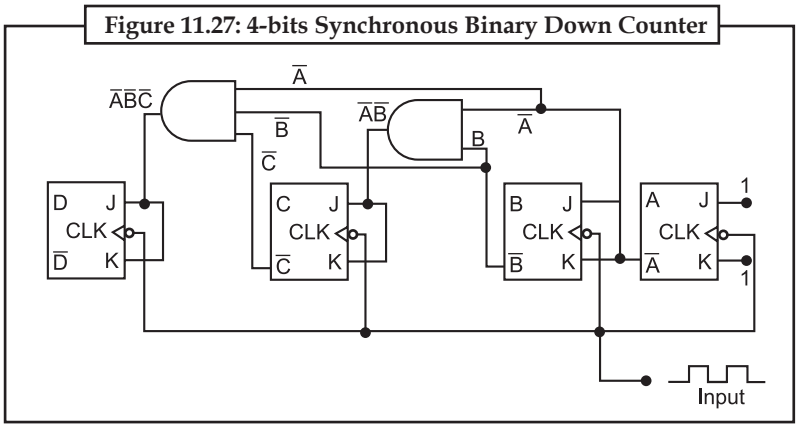
There are many variations to the basic binary counter. The one described above is the binary up counter (counts upwards). Besides the up counter, there is the binary down counter, the binary up/down counter, binary-coded-decimal (BCD) counter, etc. Any counter that counts in binary is called a binary counter.

11.3.2 Binary Down Counters

In a binary up counter, a particular bit, except for the first bit, toggles if all the lower-order bits are 1's. The opposite is true for binary down counters. That is, a particular bit toggles if all the lower-order bits are 0's and the first bit toggles on every pulse.

Taking an example, $A_4\ A_3\ A_2\ A_1 = 0100$. On the next count, $A_4\ A_3\ A_2\ A_1 = 0011$. A_1, the lowest-order bit, is always complemented. A_2 is complemented because all the lower-order positions (A_1 only in this case) are 0's. A_3 is also complemented because all the lower-order positions, A_2 and A_1 are 0's. But A_4 is not complemented the lower-order positions, $A_3\ A_2\ A_1 = 011$, do not give an all 0 condition.

Figure 11.27: 4-bits Synchronous Binary Down Counter

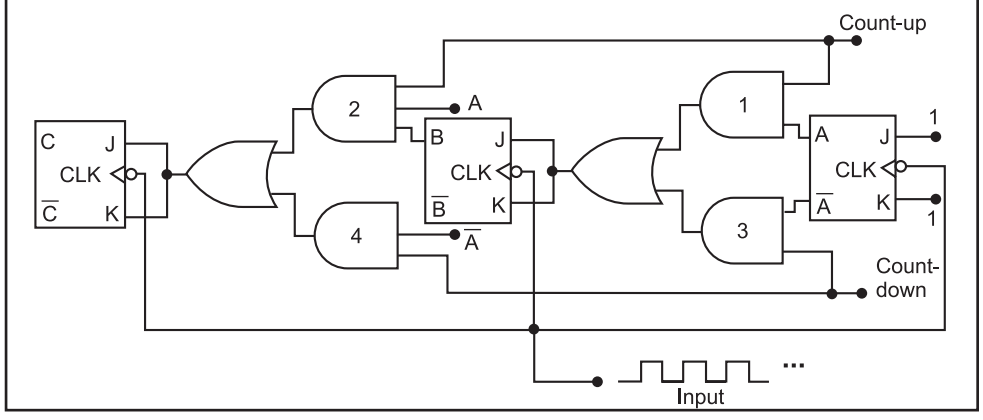


The implementation of a synchronous binary down counter is exactly the same as that of a synchronous binary up counter except that the inverted output from each flip-flop is used. All the methods used improve a binary up counter can be similarly applied here.

11.3.3 Binary Up/Down Counters

The similarities between the implementation of a binary up counter and a binary down counter leads to the possibility of a binary up/down counter, which is a binary up counter and a binary down counter combined into one. Since the difference is only in which output of the flip-flop to use, the normal output or the inverted one, we use two AND gates for each flip-flop to "choose" which of the output to use.

Figure 11.28: 3-bits Synchronous Binary Up/Down Counter



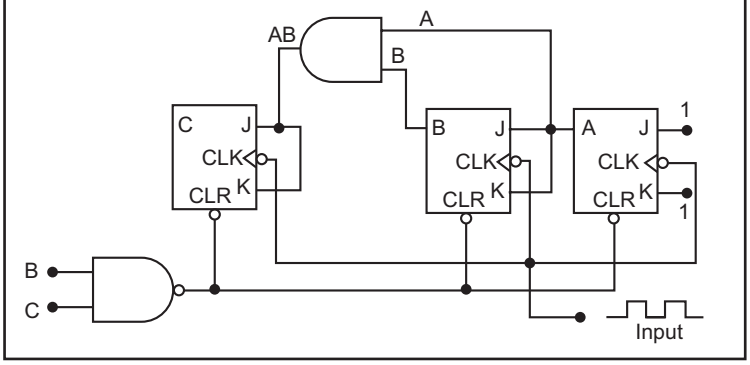
Notes

From the diagram, we can see that COUNT-UP and COUNT-DOWN are used as control inputs to determine whether the normal flip-flop outputs or the inverted ones are fed into the J-K inputs of the following flip-flops. If neither is at logic level 1, the counter does not count and if both are at logic level 1, all the bits of the counter toggle at every clock pulse. The OR gate allows either of the two outputs which have been enabled to be fed into the next flip-flop. As with the binary up and binary down counter, the speed up techniques apply.

11.3.4 MOD-N/Divide-by-N Counters

Normal binary counter counts from 0 to $2^N - 1$, where N is the number of bits/flip-flops in the counter. In some cases, we want it to count to numbers other than $2^N - 1$. This can be done by allowing the counter to skip states that are normally part of the counting sequence. There are a few methods of doing this. One of the most common methods is to use the CLEAR input on the flip-flops.

Figure 11.29: 3-bits Synchronous Binary MOD-6 Counter



In the example above, we have a MOD-6 counter. Without the NAND gate, it is a MOD-8 counter. Now, with the NAND gate, the output from the NAND gate is connected to the asynchronous CLEAR inputs of each flip-flop. The inputs to the NAND gate are the outputs of the B and C flip-flops. So, all the flip-flops will be cleared when B = C = 1 (1102 = 610). When the counter goes from state 101 to state 110, the NAND output will immediately clear the counter to state 000. Once the flip-flops have been cleared, the B = C = 1 condition no longer exists and the NAND output goes back to high. The counter will therefore count from 000 to 101, and for a very short period of time, be in state 110 before the counter is cleared. This state is called the temporary state and the counter usually only remains in a temporary state for a few nanoseconds. We can essentially say that the counter skips 110 and 111 so that it goes only six different states; thus, it is a MOD-6 counter. We also have to note that the temporary state causes a spike or glitch on the output waveform of B. This glitch is very narrow and will not normally be a problem unless it is used to drive other circuitry outside the counter. The 111 state is the unused state here. In a state machine with unused states, we need to make sure that the unused states do not cause the system to hang, i.e. no way to get out of the state. We do not have to worry about this here because even if the system does go to the 111 state, it will go to state 000, a valid state on the next clock pulse.

11.3.5 Binary Coded Decimal (BCD) Counters

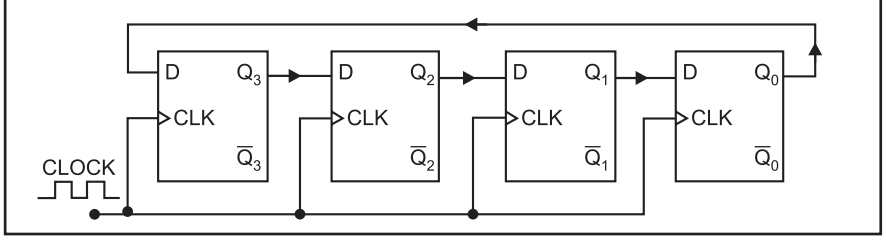
The BCD counter is just a special case of the MOD-N counter (N = 10). BCD counters are very commonly used because most human beings count in decimal. To make a digital clock which can tell the hour, minute and second, for example, we need 3 BCD counters (for the second digit of the hour, minute and second), two MOD-6 counters (for the first digit of the minute and second), and one MOD-2 counter (for the first digit of the hour).

Notes

11.3.6 Ring Counters

Ring counters are implemented using shift registers. It is essentially a circulating shift register connected so that the last flip-flop shifts its value into the first flip-flop. There is usually only a single 1 circulating in the register, as long as clock pulses are applied.

Figure 11.30: 4-bit Synchronous Ring Counter



In the diagram above, assuming a starting state of $Q_3 = 1$ and $Q_2 = Q_1 = Q_0 = 0$. At the first pulse, the 1 shift from Q_3 to Q_2 and the counter is in the 0100 state. The next pulse produces the 0010 state and the third, 0001. At the fourth pulse, the 1 at Q_0 is transferred back to Q_3, resulting in the 1000 state, which is the initial state. Subsequent pulses will cause the sequence to repeat, hence the name ring counter.

The ring counter above functions as a MOD-4 counter since it has four distinct states and each flip-flop output waveform has a frequency equal to one-fourth of the clock frequency. A ring counter can be constructed for any MOD number. A MOD-N ring counter will require N flip-flops connected in the arrangement as the diagram above.

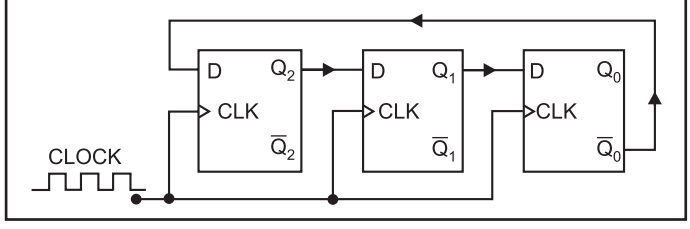
A ring counter requires more flip-flops than a binary counter for the same MOD number. For example, a MOD-8 ring counter requires 8 flip-flops while a MOD-8 binary counter only requires 3 (23 = 8). So, if a ring counter is less efficient in the use of flip-flops than a binary counter, why do we still need ring counters? One main reason is because ring counters are much easier to decode. In fact, ring counters can be decoded without the use of logic gates. The decoding signal is obtained at the output of its corresponding flip-flop.

For the ring counter to operate properly, it must start with only one flip-flop in the 1 state and all the others at 0. Since, it is not possible to expect the counter to come up to this state when power is first applied to the circuit, it is necessary to preset the counter to the required starting state before the clock pulses are applied. One way to do this is to apply a pulse to the PRESET input of one of the flip-flops and the CLEAR inputs of all the others. This will place a single 1 in the ring counter.

11.3.7 Johnson/Twisted-Ring Counters

The Johnson counter, also known as the twisted-ring counter, is exactly the same as the ring counter except that the inverted output of the last flip-flop is connected to the input of the first flip-flop.

Figure 11.31: 4-bit Synchronous Johnson Counter



The Johnson counter works in the following way: Take the initial state of the counter to be 000. On the first clock pulse, the inverse of the last flip-flop will be fed into the first flip-flop, producing the state 100. On the second clock pulse, since the last flip-flop is still at level 0, another 1 will be fed into the first flip-flop, giving the state 110. On the third clock pulse, the state 111 is produced.

Notes

On the fourth clock pulse, the inverse of the last flip-flop, now a 0 will be shifted to the first flip-flop, giving the state 011. On the fifth and sixth clock pulse, using the same reasoning, we will get the states 001 and 000, which is the initial state again. Hence, this Johnson counter has six distinct states: 000, 100, 110, 111, 011 and 001, and the sequence is repeated so long as there is input pulse. Thus, this is a MOD-6 Johnson counter.

The MOD number of a Johnson counter is twice the number of flip-flops. In the example above, three flip-flops were used to create the MOD-6 Johnson counter. So, for a given MOD number, a Johnson counter requires only half the number of flip-flops needed for a ring counter. However, a Johnson counter requires decoding gates whereas a ring counter does not. As with the binary counter, one logic gate (AND gate) is required to decode each state, but with the Johnson counter, each gate requires only two inputs, regardless of the number of flip-flops in the counter. Note that we are comparing with the binary counter using the speed up technique discussed above. The reason for this is that for each state, two of the N flip-flops used will be in a unique combination of states. In the example above, the combination $Q_2 = Q_1 = 0$ occurs only once in the counting sequence, at the count of 0. The state 010 does not occur. Thus, an AND gate with inputs (not Q_2) and (not Q_2) can be used to decode for this state. The same characteristic is shared by all the other states in the sequence.

A Johnson counters represent a middle ground between ring counters and binary counters. A Johnson counter requires fewer flip-flops than a ring counter but generally more than a binary counter; it has more decoding circuitry than a ring counter but less than a binary counter. Thus, it sometimes represents a logical choice for certain applications.

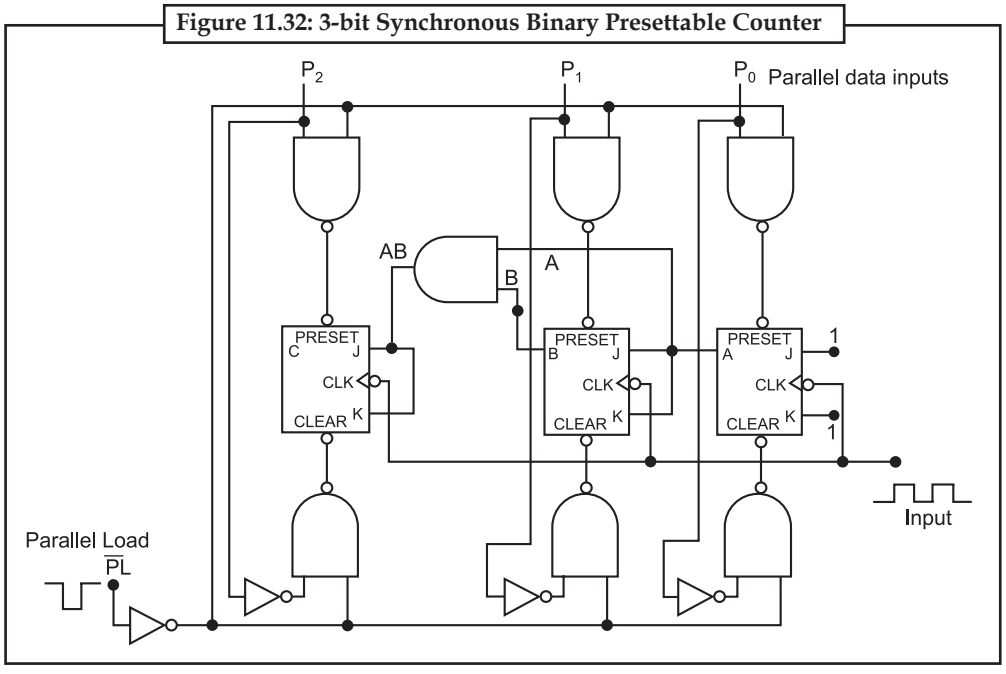

Did u know?

Alan Jones first appeared in summer 1984 in the RRCA publication footnotes using the calibrated bicycle method announced the Jones Counter.

11.3.8 Loadable/Presettable Counters

Many synchronous counters available as ICs are designed to be presettable. This means that they can be preset to any desired starting value. This can be done either asynchronously independent of the clock signal or synchronously (on the active transition of the clock signal). This presetting operation is also known as loading, hence the name loadable counter. The diagram below shows a 3-bit asynchronously presettable synchronous up counter.

Figure 11.32: 3-bit Synchronous Binary Presettable Counter



Notes

In the diagram above, the J, K and CLK inputs are wired the same way as a synchronous up counter. The asynchronous PRESET and CLEAR inputs are used to perform the asynchronous presetting. The counter is loaded by applying the desired binary number to the inputs P_2, P_1 and P_0 and a LOW pulse is applied to the PARALLEL LOAD input, not (PL). This will asynchronously transfer P_2, P_1 and P_0 into the flip-flops. This transfer occurs independently of the J, K, and CLK inputs. As long as not(PL) remains in the LOW state, the CLK input has no effect on the flip-flop. After not(PL) returns to high, the counter resumes counting, starting from the number that was loaded into the counter.

For the example above, say that $P_2 = 1$, $P_1 = 0$, and $P_0 = 1$. When not (PL) is high, these inputs have no effect. The counter will perform normal count-up operations if there are clock pulses. Now, let us say that not (PL) goes low at $Q_2 = 0$, $Q_1 = 1$ and $Q_0 = 0$. This will produce LOW states at the CLEAR input of Q_1, and the PRESET inputs of Q_2 and Q_0. This will make the counter go to state 101 regardless of what is occurring at the CLK input. The counter will remain at state 101 until not (PL) goes back to HIGH. The counter will then continue counting from 101.

11.4 Memory Decoding

The need for memory address decoding arises from the fact that the main memory of a computer system is not constructed from a single component, which uniquely addresses each possible memory location.

Imagine a situation where two 1M memory chips are connected to a 32-bit address bus to make 2M of memory available. Each memory chip will need twenty address lines to uniquely identify each location in it. If the address lines of each memory chip were simply connected to the first twenty CPU address lines, then both memory chips would be accessed simultaneously whenever the CPU referred to any address. There are several memory addressing schemes that address this problem.

11.4.1 Partial Address Decoding

This is the simplest and least expensive form of address decoding. In the above example, we could connect the chip select input of one memory chip to the last CPU address line, and the chip select input of the other to the same address line but via an inverter. In this way the two chips would never be accessed simultaneously.

However, this is very inefficient. Eleven of the address lines are not used, and one of the two memory chips is always selected. The usable address space of the computer has been reduced from 4G to 2K. Partial address decoding is used in small dedicated systems where low cost is the most important factor. The penalty paid is that not all the address space can be used, and future expansion will be difficult.

11.4.2 Full Address Decoding

Full address decoding is when each addressable location within a memory component corresponds to a single address on the CPU's address bus. That is, every address line is used to specify each physical memory location, through a combination of specifying a device and a location within it.

Full address decoding is very efficient in the use of the available address space, but is often impracticable to use because of the excessive hardware needed to implement it. This is particularly true where devices with a small number of addressable locations (for example memory-mapped I/O devices) are used.

11.4.3 Block Address Decoding

Block address decoding is the merger of partial address decoding and full address decoding. The memory space is divided into a number of blocks. For example, in a system with a 32-bit address

Notes

bus, the memory space could be divided into 4096 blocks of 1M. This could be implemented using simple decoding devices.

Many real systems employ a combination of the above decoding techniques. For example, several small devices may reside in the same block by using partial address decoding within that block.

Self Assessment

Multiple choice questions:

1. Shifting a register content to left by one bit position is equivalent to.....
   (*a*) division by two (*b*) addition by two
   (*c*) multiplication by two (*d*) subtraction by two
2. Data can be changed from special code to temporal code by using ..... .
   (*a*) shift registers (*b*) counters
   (*c*) combinational circuits (*d*) A/D converters
3. How many flip-flops are required for mod–16 counter?
   (*a*) 5 (*b*) 6
   (*c*) 3 (*d*) 4
4. Block address decoding is the merger of ..... address decoding and full address decoding.
   (*a*) completely (*b*) partial
   (*c*) Both (*a*) and (*b*) (*d*) None of these
5. Electronic semiconductor memory technology can be split into two main types or categories, according to the way in which the memory ..... .
   (*a*) operates (*b*) runs
   (*c*) stable (*d*) None of these

11.5 Semiconductor Memories

Semiconductor memory technology is an essential element of today's electronics. Normally based around semiconductor technology, memory is used in any equipment that uses a processor of one form or another.

Indeed as processors have become more popular and the number of microprocessor controlled items has increased so has the requirement for semiconductor memory. An additional driver has been the fact that the software associated with the processors and computers has become more sophisticated and much larger, and this too has greatly increased the requirement for semiconductor memory. In view of the pressure on memory, new and improved semiconductor memory technologies are being researched and development can be very rapid. Nevertheless, the more mature semiconductor memory technologies are still in widespread use and will remain so for many years to come. In addition to these new applications such as digital cameras, PDAs and many more applications have given rise to the need to memories. Accordingly, it is not uncommon to see semiconductor memories of 8 Gbyte and much more required for various applications.

Notes

Figure 11.33: Semiconductor Memory

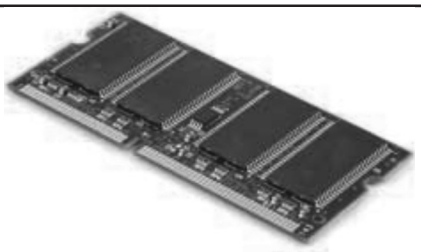


Figure 11.33 is an example of semiconductor memory technology used in computers.

With the rapid growth in the requirement for semiconductor memories there have been a number of technologies and types of memory that have emerged. Names such as ROM, RAM, EPROM, EEPROM, Flash memory, DRAM, SRAM, SDRAM, and the very new MRAM can now be seen in the electronics literature. Each one has its own advantages and area in which it may be used.

11.5.1 Types of Semiconductor Memory

Electronic semiconductor memory technology can be split into two main types or categories, according to the way in which the memory operates:

***Random Access Memory* (*RAM*)**

As the names suggest, the RAM or random access memory is a form of semiconductor memory technology that is used for reading and writing data in any order as required. It is used for such applications as the computer or processor memory where variables and other stored and are required on a random basis. Data is stored and read many times to and from this type of memory.

***Read Only Memory* (*ROM*)**

A ROM is a form of semiconductor memory technology used where the data is written once and then not changed. In view of this it is used where data needs to be stored permanently, even when the power is removed—many memory technologies lose the data once the power is removed. As a result, this type of semiconductor memory technology is widely used for storing programs and data that must survive when a computer or processor is powered down. For example the BIOS of a computer will be stored in ROM. As the name implies, data cannot be easily written to ROM. Depending on the technology used in the ROM, writing the data into the ROM initially may require special hardware. Although it is often possible to change the data, this gain requires special hardware to erase the data ready for new data to be written in.

Both of these categories of semiconductor technology are widely used. Each type being used in different areas and applications within microprocessor based electronics systems.

11.5.2 Semiconductor Memory Technologies

There is a large variety of types of ROM and RAM that are available. These arise from the variety of applications and also the number of technologies available. This means that there is a large number of abbreviations or acronyms and categories for memories ranging from Flash to MRAM, PROM to EEPROM, and many more.

PROM

This stands for Programmable Read-Only Memory. It is a semiconductor memory which can only have data written to it once the data written to it is permanent. These memories are bought in a blank format and they are programmed using a special PROM programmer. Typically a PROM will consist of an array of fusible links some of which are "blown" during the programming process to provide the required data pattern.

Notes

EPROM

This is an Erasable Programmable Read-Only Memory. This form of semiconductor memory can be programmed and then erased at a later time. This is normally achieved by exposing the silicon to ultraviolet light. To enable this to happen there is a circular window in the package of the EPROM to enable the light to reach the silicon of the chip. When the PROM is in use, this window is normally covered by a label, especially when the data may need to be preserved for an extended period. The PROM stores its data as a charge on a capacitor. There is a charge storage capacitor for each cell and this can be read repeatedly as required. However it is found that after many years the charge may leak away and the data may be lost. Nevertheless, this type of semiconductor memory used to be widely used in applications where a form of ROM was required, but where the data needed to be changed periodically, as in a development environment, or where quantities were low.

EEPROM

This is an Electrically Erasable Programmable Read-Only Memory. Data can be written to it and it can be erased using an electrical voltage. This is typically applied to an erase pin on the chip. Like other types of PROM, EEPROM retains the contents of the memory even when the power is turned off. Also like other types of ROM, EEPROM is not as fast as RAM.

Flash memory

Flash memory may be considered as a development of EEPROM technology. Data can be written to it and it can be erased, although only in blocks, but data can be read on an individual cell basis. To erase and re-program areas of the chip, programming voltages at levels that are available within electronic equipment are used. It is also non-volatile, and this makes it particularly useful. As a result Flash memory is widely used in many applications including memory cards for digital cameras, mobile phones, computer memory sticks and many other applications.

DRAM

Dynamic RAM is a form of random access memory. The DRAM uses a capacitor to store each bit of data, and the level of charge on each capacitor determines whether that bit is a logical 1 or 0. However, these capacitors do not hold their charge indefinitely, and therefore the data needs to be refreshed periodically. As a result of this dynamic refreshing it gains its name of being a dynamic RAM. The DRAM is the form of semiconductor memory that is often used in equipment including personal computers and workstations where it forms the main RAM for the computer.

SRAM

Static Random Access Memory. This form of semiconductor memory gains its name from the fact that, unlike DRAM, the data does not need to be refreshed dynamically. It is able to support faster read and write times than DRAM (typically 10 ns against 60 ns for DRAM), and in addition its cycle time is much shorter because it does not need to pause between accesses. However it consumes more power, is less dense and more expensive than DRAM. As a result of this it is normally used for caches, while DRAM is used as the main semiconductor memory technology.

SDRAM

Synchronous DRAM. This form of semiconductor memory can run at faster speeds than conventional DRAM. It is synchronized to the clock of the processor and is capable of keeping two sets of memory addresses open simultaneously. By transferring data alternately from one set of addresses, and then the other, SDRAM cuts down on the delays associated with non-synchronous RAM, which must close one address bank before opening the next.

Notes

MRAM

This is Magneto-resistive RAM, or Magnetic RAM. It is a non-volatile RAM memory technology that uses magnetic charges to store data instead of electric charges. Unlike technologies including DRAM, which require a constant flow of electricity to maintain the integrity of the data, MRAM retains data even when the power is removed. An additional advantage is that it only requires low power for active operation. As a result, this technology could become a major player in the electronics industry now that production processes have been developed to enable it to be produced.

Did u know?

A device for storing digital information that is fabricated by using integrated circuit technology. Also known as integrated-circuit memory; large-scale integrated memory; memory chip; semiconductor storage; transistor memory.

Case Study

Semiconductor History

Memory chips have been called the crude oil of the twenty-first century. They are used in a wide variety of electronic applications from children's toys to sophisticated communication satellites. The current generation of memory chip (64 Mb) is capable of storing 3,355 pages of text on a piece of about the size of a dime.

What is a Semiconductor?

A number of elements are classified as semiconductors including silicon, zinc, and germanium. These elements have the ability to conduct electrical current, and they can be regulated in the amount of their conductivity. Silicon is the most widely used semiconductor material because it is easily obtained.

Silicon is basically extracted from sand. It has been used for centuries to make cast iron, bricks, and pottery. In ultra-pure form, the controlled addition of minute amounts of certain impurities (called dopants) alters the atomic structure of the silicon. The silicon can then be made to act as a conductor or a non-conductor, depending upon the polarity of an electrical charge applied to it. Hence, the generic term semiconductor.

Early Developments

Semiconductor materials were studied in laboratories as early as 1830. The first materials studied were a group of elements and compounds that were usually poor conductors if heated. Shining light on some of them would generate an electrical current that could pass through them in one direction only.

By 1874, electricity was being used not only to carry power, but to carry information. The telegraph, telephone, and later the radio were the earliest devices in an industry that would eventually be called electronics.

Radio receivers required a device called a rectifier to detect signals. Ferdinand Braun used the rectifying properties of the galena crystal, a semiconductor material composed of lead sulfide, to create the cat's whisker diode for this purpose. Thus was born the first semiconductor device.

The Integrated Circuit

Until 1959, all electronic components were discrete: that is, they performed only one function, and many of them had to be wired together to create a functional circuit. Although a great number of identical discrete transistors could be fabricated on a single wafer, they then had

Contd...

Notes

to be cut up and individually packaged in tiny cans. Packaging each component and hand wiring the components into circuits was extremely inefficient. The military sought more efficient methods of making circuits.

New technologies emerged and integrated circuits were soon developed with various components (transistors, resistors, and capacitors) formed on the same chip, but interconnection of the various components still required tedious hand wiring.

In 1959, Jean Hoerni and Robert Noyce developed a new process called planar technology at Fairchild Semiconductor which enabled them to diffuse various layers onto the surface of a silicon wafer to make a transistor, leaving a layer of protective oxide on the junctions. This process allowed metal interconnections to be evaporated onto the flat transistor surface and replaced the hand wiring. The new process used silicon instead of germanium, and made commercial production of ICs possible.

The initial resistance to the new IC technology gave way to enormous popularity. By the end of the 1960s, nearly 90% of all the components manufactured were integrated circuits.

Questions:

1. Explain the brief history of semiconductors.
2. What do you understand by dopants?

Self Assessment

True or False:

6. RAM is the place in a computer where the operating system, application programs, and data in current use are kept.
   (*a*) True (*b*) False
7. RAM is a very slow-speed type of memory.
   (*a*) True (*b*) False

Multiple choice questions:

8. PROM, stands for ..... .
   (*a*) Personal Read Only Memory (*b*) Peripheral Read Only Memory
   (*c*) Programmable Read Only Memory (*d*) None of these
9. Flash memory may be considered as a development of .....technology.
   (*a*) EEPROM (*b*) DRAM
   (*c*) EPROM (*d*) None of these

11.6 Summary

The MOD number of a Johnson counter is twice the number of flip-flops.

- The flip-flop in the lowest-order position is complemented/toggled with every clock pulse and a flip-flop in any other position is complemented on the next clock pulse provided all the bits in the lower-order positions are equal to 1.
- The basic types of shift registers are studied, such as Serial-In/Serial-Out, Serial-In/Parallel-Out, Parallel-In/Serial-Out, Parallel-In/Parallel-Out, bidirectional shift registers.
- Semiconductor memory technology is moving forward apace not only are the sizes of memories being increased, and the data densities improving, but new forms of semiconductor memory such as MRAM are being introduced.

Notes

- Each of plural logic gates has two input terminals each of which is connected to an output of a predetermined step of the shift register, so that an output signal outputted from each logic gate to the corresponding driving circuit for an operating device constitutes timing signal carrying information including starting and terminating times of operation.
- The need for memory address decoding arises from the fact that the main memory of a computer system is not constructed from a single component, which uniquely addresses each possible memory location.

11.7 Keywords

***Random access memory*:** This is the place in a computer where the operating system, application programs, and data in current use are kept so that they can be quickly reached by the computer's processor.

***Read-only memory*:** This is a form of semiconductor memory technology used where the data is written once and then not changed.

***Ring counters*:** These are implemented using shift registers and it is essentially a circulating shift register connected so that the last flip-flop shifts its value into the first flip-flop.

***Ripple counter*:** It is easy to understand. Each stage acts as a Divide-by-2 counter on the stage's signal.

***Semiconductor memory*:** This technology is an essential element of today's electronics. Normally based around semiconductor technology, memory is used in any equipment that uses a processor of one form or another.

***Synchronous binary counter*:** This counts from 0 to $2^N - 1$, where N is the number of bits/flip-flops in the counter.

***Synchronous dynamic random access memory* (*SDRAM*):** The SDRAM, this form of semiconductor memory can run at faster speeds than conventional DRAM.

Lab Exercise

1. Design a 4.bit parallel-in/parallel-out register with loading information in one asynchronous stage.
2. Design a 4.bit maximum length shift register counter.

11.8 Review Questions

1. Using D-flip-flops and waveforms explain the working of a 4-bit SISO shift register.
2. What is a shift register? Can a shift register be used as a counter? If yes, explain how.
3. Explain ripple counter in detail.
4. Describe the synchronous counters. Explain with circuit diagram.
5. What do you understand by 4-bit parallel-access shift register?
6. Explain
   (*a*) Partial Address Decoding
   (*b*) Full Address Decoding
7. What are the semiconductor memories? Also describe the types of semiconductor memory.
8. Describe the semiconductor memory technologies.
9. What is block address decoding?
10. Describe the partial address decoding.

Notes

Answers to Self Assessment

1. (*c*) 2. (*a*) 3. (*d*) 4. (*b*) 5. (*a*)
6. (*a*) 7. (*b*) 8. (*c*) 9. (*a*)

11.9 Further Readings

Books

Digital Fundamentals, Prentice – Hall, Ninth Edition, by Floyd Thomas L.

Digital Electronics – Principles and Applications Tata McGraw Hill, by Tokheim R.L.

Online link

http://www.answers.com/topic/semiconductor-memory

Unit 12: A/D and D/A Converters

Notes

CONTENTS

Objectives

After studying this unit, you will be able to:

- Define variables-resistor networks
- Describe binary ladders
- Discuss D/A converters
- Describe A/D converters and techniques

Notes

- Discuss dual-slope A/D conversion
- Describe the single-slope A/D converter
- Discuss the dual-slope A/D converter
- Describe the successive approximation A/D converter
- Describe the flash converters

Introduction

Digital-to-analog (D/A) and Analog-to-digital (A/D) conversion form two very important aspects of digital data processing. Digital-to-analog conversion involves translation of digital information into equivalent analog information. As an example, the output of a digital system might be changed to analog form for the purpose of driving a pen recorder. Similarly, an analog signal might be required for the servomotors which drive the cursor arms of a plotter. In this respect, a D/A converter is sometimes considered a decoding device.

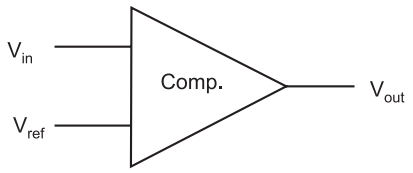
The process of changing an analog signal to an equivalent digital signal is accomplished by the use of an A/D converter. For example, an A/D converter is used to change the analog output signals from transducers (measuring temperature, pressure, vibration, etc.) into equivalent digital signals. These signals would then be in a form suitable for entry into a digital system. An A/D converter is often referred to as an encoding device since it is used to encode signals for entry into a digital system.

Digital-to-analog conversion is a straightforward process and is considerably easier than A/D conversion. In fact, a D/A converter are usually an integral part of any A/D converter.

Principle of analog-to-digital conversion

The process of analog-to-digital conversion always involves comparing two analogue signals: an input signal and some reference signal. The comparison is carried out with a circuit called a comparator as shown in circuit.

A comparator circuit is essentially a high-gain differential amplifier. When $V_{in} > V_{ref}$ the output of the comparator swings to the positive supply rail, and so the output is "1". On the other hand, when $V_{in} < V_{ref}$ the output voltage swings to the earth rail, "0". Thus, the comparator gives a clear indication of which of two voltages is the larger.



12.1 Variables-Resistor Networks

The basic problem in converting a digital signal into an equivalent analog signal change the *n* digital voltage levels into one equivalent analog voltage. This can be easily accomplished by designing a resistive network that will change each digital level into an equivalent binary weighted voltage (or current).

12.1.1 Binary Equivalent Weight

As an example of what is meant by binary equivalent weight, consider the truth table for the 3-bit binary signal shown in Figure 12.1. Suppose that we want to change the eight possible digital

Notes

signals in this Figure 12.1 into equivalent analog voltages. The smallest number represented is 000; let us make this equal to 0 V. The largest number is 111; let us make this equal to +7 V. This then establishes the range of the analog signal to be developed. (There is nothing special about the voltage levels chosen; they were simply selected for convenience.)

Now, notice that between 000 and 111 there are seven discrete levels to be defined. Therefore, it will be convenient to divide the analog signal into seven levels. The smallest incremental change in the digital signal is represented by the least-significant bit (LSB), 2°. Thus we would like to have this bit cause a change in the analog output that is equal to one-seventh of the full-scale analog output voltage. The resistive divider will then be designed such that a 1 in the 2° position will cause +7 *1/7 = + 1 V at the output.

Since $2^1 = 2$ and $2° = 1$, it can be clearly seen that the 2^1 bit represents a number that is twice the size of the 2° bit. Therefore, a 1 in the 2^1 bit position must cause a change in the analog output voltage that is twice the size of the LSB.

Figure 12.1: Binary Equivalent Truth Table

2^2	2^1	2°
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Figure 12.2: Binary Equivalent Weights

Bit	Weight
2^0	1/7
2^1	2/7
2^2	4/7
Sum	7/7

(*a*)

Bit	Weight
2^0	1/15
2^1	2/15
2^2	4/15
2^3	8/15
Sum	15/15

(*b*)

Similarly, $2^2 = 4 = 2 * 2^1 = 4 * 2^0$, and thus the 2^2 bit must cause a change in the output voltage equal to four times that of the LSB. The 2^2 bit must then cause an output voltage change of +7 *4/7 = +4 V.

The process can be continued, and it will be seen that each successive bit must have a weight twice that of the preceding bit. Thus the LSB is given a binary equivalent weight of 1/7 or 1 part in 7. The next LSB is given a weight of 2/7, which is twice the LSB, or 2 parts in 7. The MSB (in the case of this 3-bit system) is given a weight of 4/7, which is 4 times the LSB or 4 parts in 7. Notice that the sum of the weights must equal 1. Thus 1/7 + 2/7 + 4/7 = 7/7 = 1. In general, the binary equivalent weight assigned to the LSB is 1/(2n – 1), where n is the number of bits. The remaining weights are found by multiplying by 2, 4, 8, and so on. Remember,

$$\text{LSB weight} = \frac{1}{(2n-1)}$$

Notes

Example 12.1: Find the binary equivalent weight of each bit in a 4-bit system.

Solution:

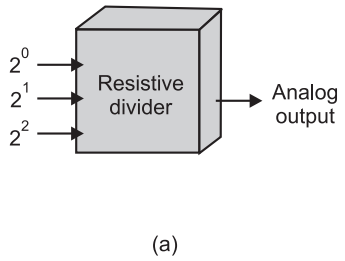
LSB has a weight of $1/(2^4 - 1) = 1/(16 - 1) = 1/15$, or 1 part in 15. The second LSB has a weight of 2 *1/15 = 2/15. The third LSB has a weight of 4 *1/15 = 4/15 and the MSB has a weight of 8 *1/15 = 8/15. As a check, the sum of the weights must equal 1. Thus 1/15 + 2/15 + 4/15 + 8/15 = 15/15 = 1. The binary equivalent weights for 3-bit and 4-bit system are summarized in Figure 12.2.

Caution Always remember to divide the analog signal into seven levels, in binary equivalent weight.

12.1.2 Resistive Divider

What is now desired is a resistive divider that has three digital inputs and one analog output as shown in Figure 12.3a. Assume that the digital input levels are 0= 0 V and l = +7 V. Now, for an input of 001, the output will be + 1 V. Similarly, an input of 010 will provide an output of +2 V, and an input of 100 will provide an output of 4 V. The digital input 011 is seen to be a combination of the signals 001 and 010. If the +1V from the 2^0 bit is added to the $+2^0$ V from the 2^1 bit, the desired +3 V output for the 011 input is achieved. The other desired voltage levels are shown in Figure 12.3b; they, too, are additive combinations of voltages.

Figure 12.3: Resistive Divider



Digital input	Analog output
0 0 0	0 V
0 0 1	+1 V
0 1 0	+2 V
0 1 1	+3 V
1 0 0	+4 V
1 0 1	+5 V
1 1 0	+6 V
1 1 1	+7 V

(b)

Thus the resistive divider must do two things in order to change the digital input into an equivalent analog output voltage:

1. The 2^0 bit must be changed to + l V, and 2^1 bit must be changed to +1 V, 2^2 bit must be changed to +4 V.
2. These three voltages representing the digital bits must be summed together to form the analog output voltage.

A resistive divider that performs these functions is shown in Figure 12.4. Resistors R_0, R_1 and R_2 form the divider network. Resistance R_L represents the load to which the dividers connected and is considered to be large enough that it does not load the divider network.

Assume that the digital input signal 001 is applied to this network. Recalling 0 = 0 V and 1 = +7 V, you can draw the equivalent circuit shown in Figure 12.5. Resistance R_L is considered large and is neglected. The analog output voltage V_A can be most easily found by use of Millman's theorem, which states that the voltage appearing at any node in a resistive network is equal to the summation of the currents entering the node (found by assuming that the node voltage is zero) divided by the summation of the conductances connected to the node. In equation form, Millman's theorem is

$$V = \frac{V_1/R_1 + V_2/R_2 + V_3/R_3 + \ldots}{1/R_1 + 1/R_2 + 1/R_3 + \ldots}$$

Notes

Applying Millman's theorem to Figure 12.5, we obtain

$$V_A = \frac{V_0/R_0 + V_1/(R_0/2) + V_2(R_0/4)}{1/R_0 + 1/(R_0/2) + 1/(R_0/4)}$$

$$= \frac{7/R_0}{1/R_0 + 2/R_0 + 4/R_0} = \frac{7}{7} = +1V$$

Drawing the equivalent circuits for the other 7-input combinations and applying Millman's theorem will lead to the table of voltages shown in Figure 12.3.

Example 12.2:

For a 4-input resistive divider (0 = 0 V, 1 = + 10 V), find (*a*) the full-scale output voltage; (*b*) the output voltage change due to the LSB; (*c*) the analog output voltage for a digital input of 1011.

Figure 12.4: Resistive Ladder

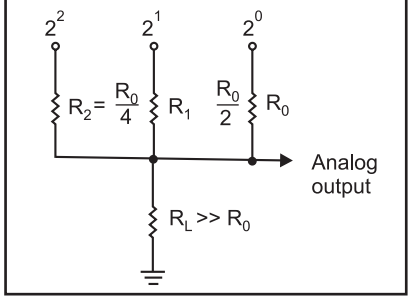
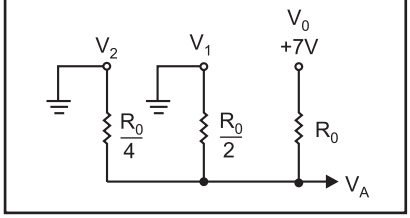


Figure 12.5: Resistance



Solution:

(*a*) The maximum output voltage occurs when all the inputs are at + 10 V. If all four inputs are at + 10 V, the output must also be at + 10 V (ignoring the effects of R_L).

(*b*) For a 4-bit digital number, there are 16 possible states. There are 15 steps between these 16 states, and the LSB must be equal to 1/15 of the full-scale output voltage. Therefore, the change in output voltage due to the LSB is +10 * 1/15 = +2/3 V.

(*c*) According to Millman's theorem, the output voltage for a digital input of 1011 is

$$V_A = \frac{10/R_0 + 10/(R_0/2) + 0/(R_0/4) + 10/(R_0/8)}{1/R_0 + 1/(R_0/2) + 1/(R_0/4) + 1/(R_0/8)}$$

$$= \frac{110}{15} = \frac{22}{3} = +7\frac{1}{3}V$$

To summarize, a resistive divider can be built to change a digital voltage into an equivalent analog voltage. The following criteria can be applied to this divider:

1. There must be one input resistor for each digital bit.
2. Beginning with the LSB, each following resistor value is one-half the size of the previous resistor.

Notes

3. The full-scale output voltage is equal to the positive voltage of the digital input signal. (The divider would work equally well with input voltages of 0 and V.)
4. The LSB has a weight of $1/(2^n - 1)$, where n is the number of input bits.
5. The change in output voltage due to a change in the LSB is equal to $V/(2^n - 1)$, where V is the digital input voltage level.
6. The output voltage V_A can be found for any digital input signal by using the following modified form of Millman's theorem:

$$V_A = \frac{V_0 2^0 + V_1 2^1 + V_2 2^2 + V_3 2^3 + ... + V_{n-1} 2^{n-1}}{2^n - 1} \quad ...(12.1)$$

Where V_0, V_1, V_2, V_3,……, V_{n-1} are the digital input voltage levels (0 or V) and n is the number of input bits.

Example 12.3:

For a 5-bit resistive divider, determine the following: (*a*) the weight assigned to the LSB; (*b*) the weight assigned to the second and third LSB; (*c*) the change in output voltage due to a change in the LSB, the second LSB, and the third LSB; (*d*) the output voltage for a digital input of 10101. Assume 0 = 0 V and 1 = + 10 V.

Solution:

(*a*) The LSB weight is $1/(2^5 - 1) = 1/31$.

(*b*) The second LSB weight is 2/31, and the third LSB weight is 4/31.

(*c*) The LSB causes a change in the output voltage of 10/31 V. The second LSB causes an output voltage change of 20/31 V, and the third LSB causes an output voltage change of 40/31 V.

(*d*) The output voltage for a digital input of 10101 is

$$V_A = \frac{10 * 2^0 + 10 * 2^1 + 10 * 2^2 + 0 * 2^3 + 10 * 2^4}{2^5 - 1}$$

$$= \frac{10(1 + 4 + 16)}{32 - 1} = \frac{210}{31} = +6.77 \text{ V}$$

This resistive divider has two serious drawbacks. The first is the fact that each resistor in the network has a different value. Since these dividers are usually constructed by using precision resistors, the added expense becomes unattractive. Moreover, the resistor used for the MSB is required to handle a much greater current than that used for the LSB resistor. For example, in a 10-bit system, the current through the MSB resistor is approximately 500 times as large as the current through the LSB resistor. For these reasons, a second type of resistive network, called a ladder, has been developed.

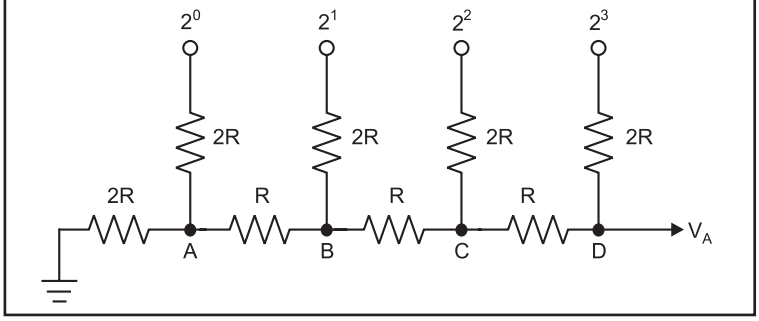
12.2 Binary Ladders

The binary ladder is a resistive network whose output voltage is a properly weighted sum of the digital inputs. Such a ladder, designed for 4 bits, is shown in Figure 12.6. It is constructed of resistors that have only two values and thus overcomes one of the objections to the resistive divider previously discussed. The left end of the ladder is terminated in a resistance of 2R, and we shall assume for the moment that the right end of the ladder (the output) is open-circuited.

Let us now examine the resistive properties of the network, assuming that all the digital inputs are at ground. Beginning at node A, the total resistance looking into the terminating resister is 2R.

Notes

Figure 12.6: Binary Ladder



The total resistance looking out toward the 2^0 input is also 2R. These two resistors can be combined to form an equivalent resistor of value R as shown in Figure 12.7a.

Now, moving to node B, we see that the total resistance looking into the branch toward node A is 2R, as is the total resistance looking out toward the 2^1 input. These resistors can be combined to simplify the network as shown in Figure 12.7b.

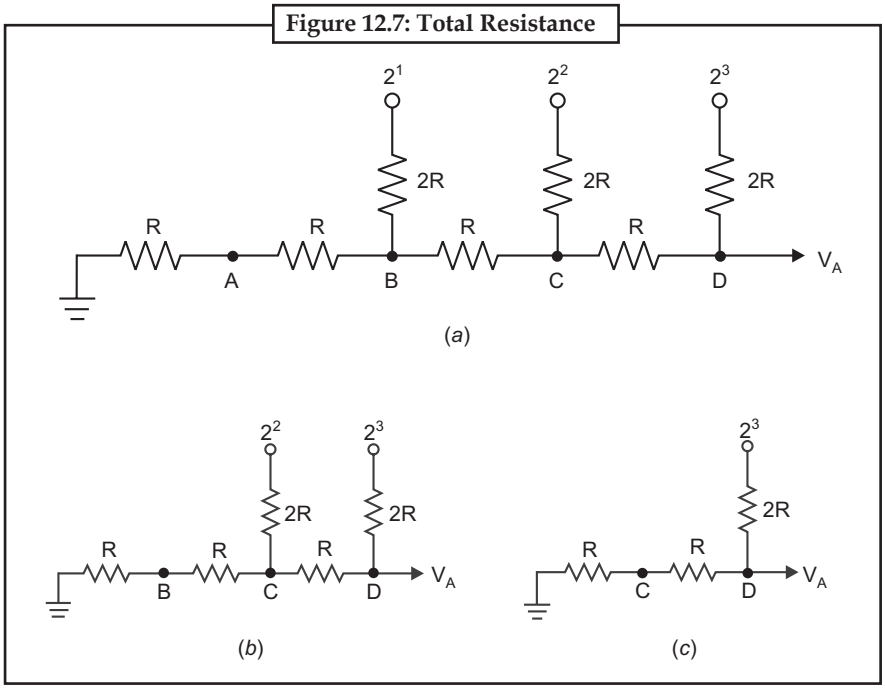
From Figure 12.7b, it can be seen that the total resistance looking from node C down the branch toward node B or out the branch toward the 2^2 input is still 2R. The circuit in Figure 12.7b can then be reduced to the equivalent as shown in Figure 12.7c.

From this equivalent circuit, it is clear that the resistance looking back toward node C is 2R, as is the resistance looking out toward the 2^3 input.

From the preceding discussion, we can conclude that the total resistance looking from any node back toward the terminating resistor or out toward the digital input is 2R. Notice that this is true regardless of whether the digital inputs are at ground or + V. The justification for this statement is the fact that the internal impedance of an ideal voltage source is 0 Ω, and we are assuming that the digital inputs are ideal voltage sources.

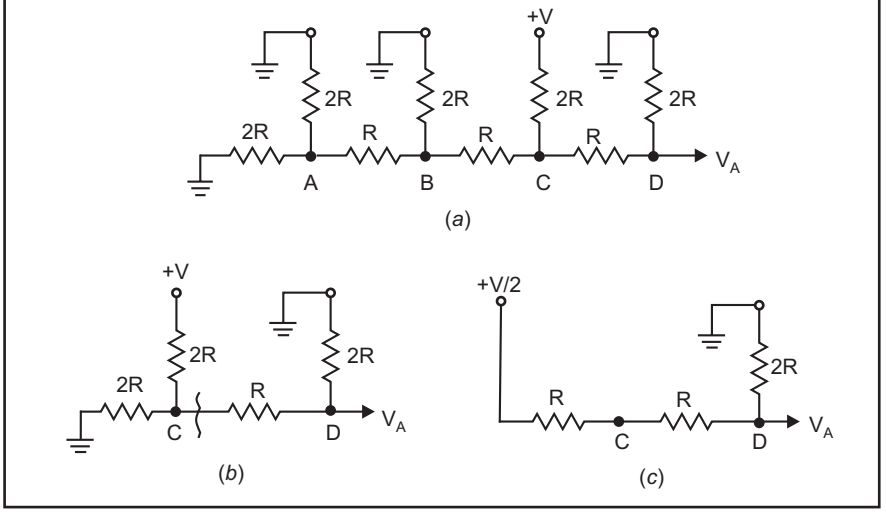
We can use the resistance characteristics of the ladder to determine the output voltages for the various digital inputs. First, assume that the digital input signal is 1000.

Figure 12.7: Total Resistance



Notes

Figure 12.8: (*a*) Binary Ladder with a Digital Input of 1000; (*b*) Equivalent Circuit for a Digital Input of 1000



This input signal, the binary ladder, can be drawn as shown in Figure 12.8a. Since there is no voltage sources to the left of node D, the entire network to the left of this node can be replaced by a resistance of 2R to form the equivalent circuit shown in Figure 12.8b. For this equivalent circuit, it can be easily seen that the output voltage is

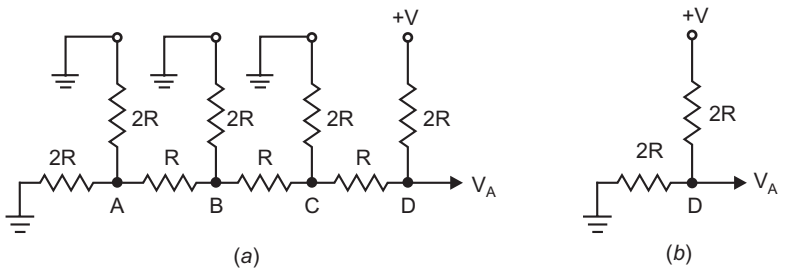
$$V_A = V * \frac{2R}{2R + 2R} = \frac{+V}{2}$$

Thus, a 1 in the MSB position will provide an output voltage of +V/2.

To determine the output voltage due to the second MSB, assume a digital input signal of 0100. This can be represented by the circuit shown in Figure 12.9a. Since there are voltage sources to the left of node C, the entire network to the left of this node can be replaced by a resistance of 2R, as shown in Figure 12.9b. Let us now replace the network to the left of node C with its Thevenin equivalent by cutting the circuit on the jagged line shown in Figure 12.9b. The Thevenin equivalent is clearly a resistance R in series with a voltage source + V/2. The final equivalent circuit with the Thevenin equivalent include shown in Figure 12.9c. From this circuit, the output voltage is clearly

$$V_A = \frac{+V}{2} * \frac{2R}{R + R + 2R} = \frac{+V}{4}$$

Figure 12.9 (*a*) Binary Ladder with a Digital Input of 0100. (*b*) Partially Reduced Equivalent circuit. (*c*) Final Equivalent Circuit Using Thevenin's Theorem



Thus the second MSB provides an output voltage of +V/4.

This process can be continued, and it can be shown that the third MSB provides an output voltage of +V/8, the fourth MSB provides an output voltage of +V/16, and so on. The output voltages for

Notes

the binary ladder are summarized in Figure. 12.10 notice that each digital input is transformed into a properly weighted binary output voltage.

Example 12.4**:**

What are the output voltages caused by each bit in a 5-bit ladder if the input levels are 0 = 0V and l = +l0 V?

Solution:

The output voltages can be easily calculated by using Figure. 12.10. They are

First MSB $V_A = \frac{V}{2} = +10/2 = +5V$

Second MSB $V_A = \frac{V}{4} = +10/4 = +2.5V$

Third MSB $V_A = \frac{V}{8} = +10/8 = +1.25V$

Fourth MSB $V_A = \frac{V}{16} = +10/16 = +0.625V$

LSB = fifth MSB $V_A = \frac{V}{32} = +10/32 = +0.3125V$

Since this ladder is composed of linear resistors, it is a linear network and the principle of superposition can be used. This means that the total output voltage due to a combination of input digital levels can be found by simply taking the sum of the output levels caused by each digital input individually.

Figure 12.10: Binary Ladder Output Voltages

Bit position	Binary weight	Output voltage
MSB	1/2	V/2
2nd MSB	1/2	V/4
3rd MSB	1/2	V/8
4th MSB	1/2	V/16
5th MSB	1/2	V/32
6th MSB	1/2	V/64
7th MSB	1/2	V/128
•	•	•
•	•	•
•	•	•
Nth MSB	$1/2^N$	$V/2^N$

In equation form, the output voltage is given by

$$V_A = \frac{V}{2} + \frac{V}{4} + \frac{V}{8} + \frac{V}{16} + ... + \frac{V}{2^n} \quad ...(12.2)$$

where n is the total number of bits at the input.

This equation can be simplified somewhat by factoring and collecting terms, the output voltage can then be given in the form

$$V_A = \frac{V_0 2^0 + V_1 2^1 + V_2 2^2 + V_3 2^3 + ... + V_{n-1} 2^{n-1}}{2^n} \quad ...(12.3)$$

where V_0, V_1, V_2, V_3,……,V_{n-1} are the digital input voltage levels. Equation (12.3) can be used to find the output voltage from the ladder for any digital input signal.

Notes

Example 12.5: Find the output voltage from a 5-bit ladder that has a digital input of 11010. Assume that 0 = 0V and l = +l0V.

Solution:

By eq. (11.3):

$$V_A = \frac{0*2^0 + 10*2^1 + 0*2^2 + 10*2^3 + 10*2^4}{2^5}$$

$$= \frac{10(2+8+16)}{32} = \frac{10*26}{32} = +8.125 \text{ V}$$

This solution can be checked by adding the individual bit contributions calculated in Example 12.4.

Notice that Eq. (12.3) is very similar to Eq. (12.1), which was developed for the resistive divider. They are, in fact, identical with the exception of the denominators. This is a subtle but very important difference. Recall that the full-scale voltage for the resistive divider is equal to the voltage level of the digital input 1. On the other hand, examination of Eq. (12.2) reveals that the full-scale voltage for the ladder is given by

$$V_A = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \ldots\ldots + \frac{1}{2^n}$$

The terms inside the brackets form a geometric series whose sum approaches 1, given a sufficient number of terms. However, it never quite reaches 1. Therefore, the full-scale output voltage of the ladder approaches V in the limit, but never quite reaches it.

Example 12.6: What is the full-scale output voltage of the 5-bit ladder in Example 12.4?

Solution:

The full-scale voltage is simply the sum of the individual bit voltages. Thus

$$V = 5 + 2.5 + 1.25 + 0.625 + 0.3125 = +9.6875 \text{ V}$$

To keep the ladder in perfect balance and to maintain symmetry, the output of the ladder should be terminated in a resistance of 2R. This will result in a lowering of the output voltage, but if the 2R load is maintained constant, the output voltages will still be a properly weighted sum of the binary input bits. If the load is varied, the output voltage will not be a properly weighted sum, and care must be exercised to ensure that the load resistance is constant.

Terminating the output of the ladder with a load of 2R also ensures that the input resistance to the ladder seen by each of the digital voltage sources is constant. With the ladder balanced in this manner, the resistance looking into any branch from any node has a value of 2R. Thus the input resistance seen by any input digital source is 3R. This is a definite advantage over the resistive divider, since the digital voltage sources can now all be designed for the same load.

Example 12.7: Suppose that the value of R for the 5-bit ladder described in Example 12.4 is 1000 Ω. Determine the current that each input digital voltage source must be capable of supplying. Also determine the full-scale output voltage, assuming that the ladder is terminated with a load resistance of 2000 Ω.

Solution:

The input resistance into the ladder seen by each of the digital sources is 3R = 3000 Ω. Thus, for a voltage level of + 10 V, each source must be capable of supplying I = 10/(3 * 10^3) = 3⅓ mA (without the 2R load resistor, the resistance looking into the MSB terminal is actually 4R). The no-load output voltage of the ladder has already been determined in Example 12.6. This open-circuit output voltage along with the open-circuit output resistance can be used to form

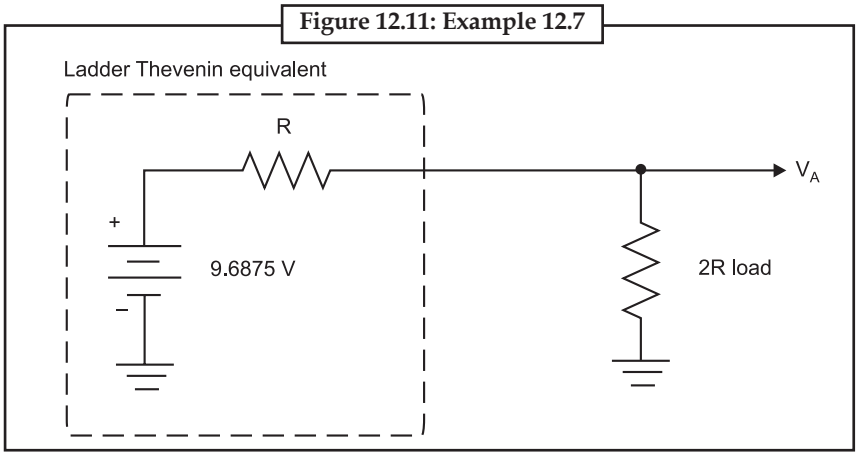
Notes

a Thevenin equivalent circuit for the output of the ladder. The resistance looking back into the ladder is clearly R = 1000Ω. Thus the Thevenin equivalent is as shown in Figure. 12.11. From this figure, the output voltage is

$$V_A = +9.6875 * \frac{2R}{2R+R} = +6.4583$$

The operational amplifier (OA) shown in Figure. 12.12a is connected as a unity-gain non-inverting amplifier. It is thus a good buffer amplifier for connection to the output of a resistive ladder. It will not load down the ladder and thus will not disturb the ladder output voltage V_A; V_A will then appear at the output of the OA.

Figure 12.11: Example 12.7

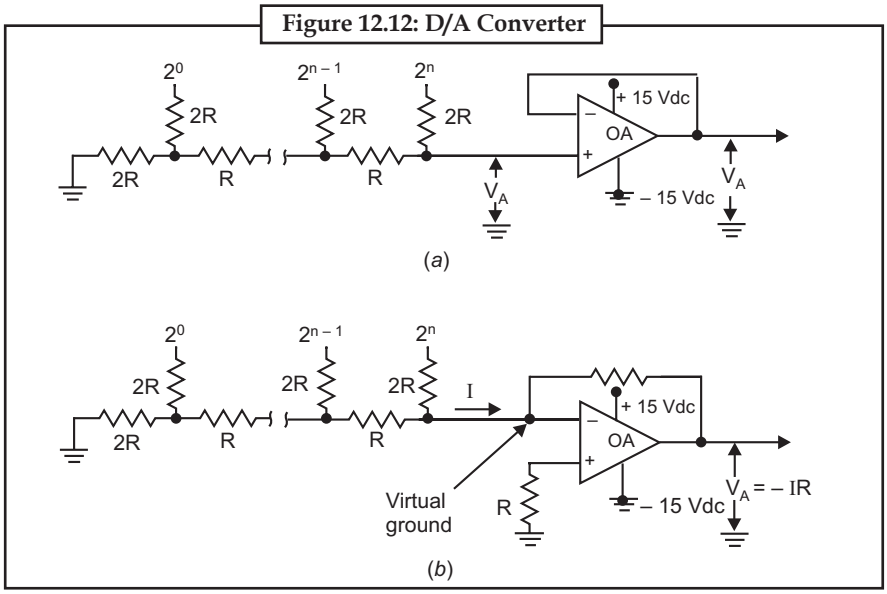


Connecting an OA with a feedback resistor R as shown in Figure 12.12b results in an amplifier that acts as an inverting current-to-voltage amplifier. That is, the output voltage V_A is equal to the negative of the input current I multiplied by R. The input impedance to this amplifier is essentially 0Ω; thus, when it is connected to an R-2R ladder, the connecting point is virtually at ground potential. In this configuration, the R-2R ladder will produce a current output. It is a binary weighted sum of the input digital levels. For instance, the MSB produces a current of V/2R. The second MSB produces a current of V/4V, and so on. But the OA multiplies these currents by -R, and thus V_A is

$$V_A = (-R)\left(\frac{V}{2R} + \frac{V}{4R} + ...\right) = -\frac{V}{2} - \frac{V}{4} - ...$$

This is exactly the same expression given in Eq. (12.2) and (12.3) except for the sign. Thus the D/A converter in Figure 12.12a and b will provide the same output voltage V_A

Figure 12.12: D/A Converter



Notes

Except for sign In Figure 12.12a, the R-2R ladder and OA are said to operate in a voltage mode, while the connection in Figure 12.12*b* is said to operate in a current mode.


Caution

Be careful while the operational amplifier (OA) is connected as a unity-gain non-inverting amplifier. It has very high input impedance, and the output voltage is equal to the input voltage.


Task Draw the truth table of Binary ladder output voltages.

12.3 D/A Converters

Either the resistive divider or the ladder can be used as the basis for a digital-to-analog (D/A) converter. It is in the resistive network that the actual translation from a digital signal to an analog voltage takes place. There is, however, the need for additional circuitry to complete the design of the D/A converter.

As an integral part of the D/A converter there must be a register that can be used to store the digital information. The simplest register is formed by use of RS flip-flops, with one flip-flop per bit. There must also be level amplifiers between the register and the resistive network to ensure that the digital signals presented to the network are all of the same level and are constant. Finally, there must be some form of gating on the input of the register such that the flip-flops can be set with the proper information from the digital system. A complete D/A converter in block-diagram form are shown in Figure 12.13a.

Let us expand on the block diagram shown in this Figure 12.13a by drawing the complete schematic for a 4-bit D/A converter as shown in Figure 12.13b. You will recognize that the resistor network used is of the ladder type.

The level amplifiers each have two inputs: one input is the + 10 V from the precision voltage source, and the other is from a flip-flop. The amplifiers work in such a way that when the input from a flip-flop is high, the output of the amplifier is at + 10 V. When the input from the flip-flop is low, the output is 0 V.

The four flip-flops form the register necessary for storing the digital information. The flip-flop on the right represents the MSB, and the flip-flop on the left represents the LSB. Each flip-flop is a simple RS latch and requires a positive level at the R or S input to reset or set it. The gating scheme for entering information into the register is straightforward and should be easy to understand. With this particular gating scheme, the flip-flops need not be reset (or set) each time new information is entered. When the READ IN line goes high, only one of the two gate outputs connected to each flip-flop is high, and the flip-flop is set or reset accordingly. Thus data are entered into the register each time the READ IN (strobe) pulse occurs. D flip-flops could be used in place of the RS flip-flops.

12.3.1 Multiple Signals

Quite often it is necessary to decode more than one signal–for example, the X and Y coordinates for a plotting board. In this event, there are two ways in which to decode the signals.

Notes

Figure 12.13: 4-bit D/A Converter

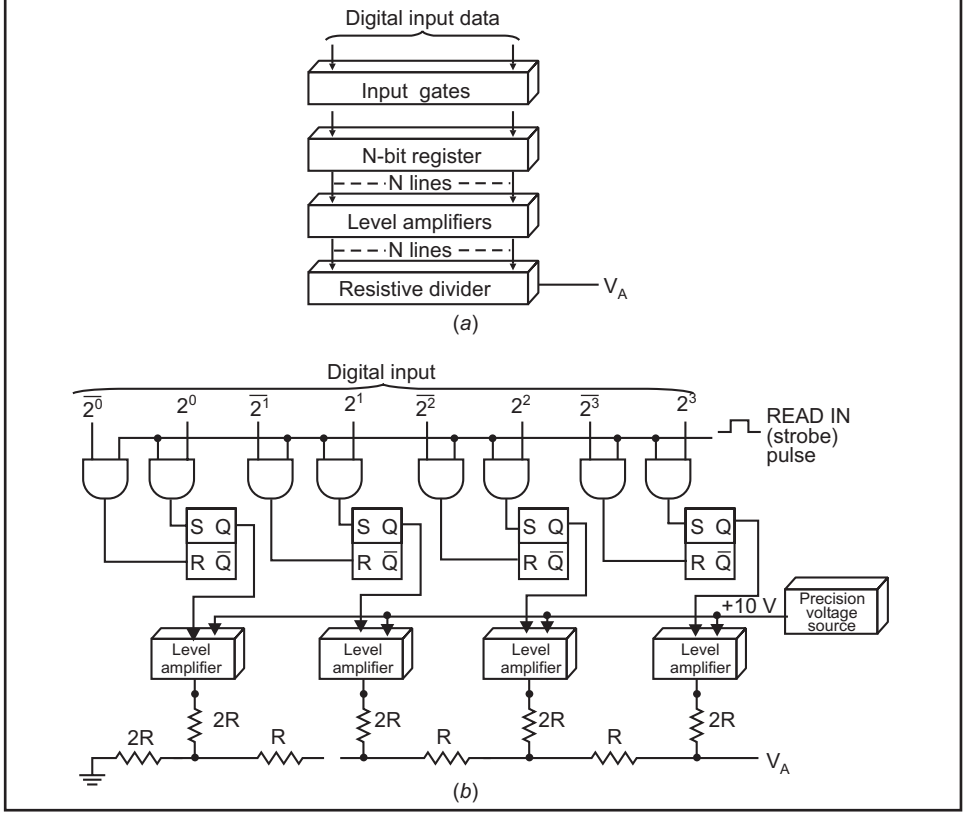
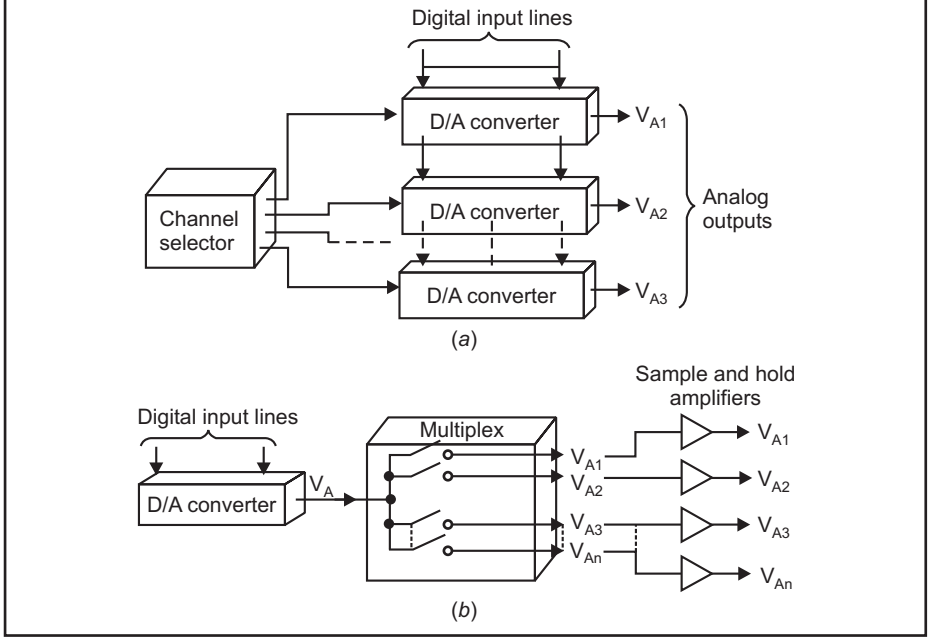


Figure 12.14: Decoding a Number of Signals – (*a*) Channel Selection Method, (*b*) Multiplex Method



The first and most obvious method is simply to use one D/A converter for each multiplexing signal. This method, shown in Figure 12.14a, has the advantage that each signal to be decoded is held in its register and the analog output voltage is then held fixed. The digital input lines are

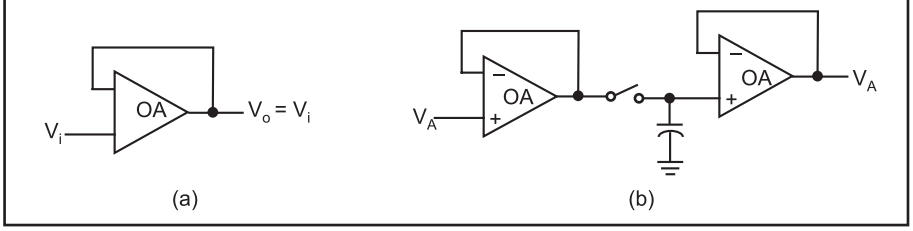
Notes

connected in parallel to each converter. The proper converter is then selected for decoding by the select lines.

The second method involves the use of only one D/A converter and switching its output. This is called multiplexing, and such a system is shown in Figure 12.14b.

An OA connected as in Figure 12.15a is a unity-gain non-inverting voltage amplifier, that is $V_0 = V_i$. Two such OAs are used with a capacitor in Figure 12.15b.

Figure 12.15: (*a*) Unity gain amplifier, (*b*) Sample-and-hold Circuit



***Steady-state*:** Sample-and-hold amplifier. When the switch is closed, the capacitor charges to the D/A.

***Accuracy test*:** Converter output voltage. When the switch is opened, the capacitor holds the voltage level until the next sampling time. The operational amplifier provides large input impedance.

***Monotonicity test*:** So as not to discharge the capacitor appreciably and at the same time offers gain to drive external circuits.

When the D/A converter is used in conjunction with a multiplexer, the maximum rate at which the converter can operate must be considered. Each time data is shifted into the register, transients appear at the output of the converter. This is due mainly to the fact that each flip-flop has different rise and fall times. Thus a settling time must be allowed between the time data is shifted into the register and the time the analog voltage is read out. This settling time is the main factor in determining the maximum rate of multiplexing the output. The worst case is when all bits change (e.g. from 1000 to 0111).

Naturally, the capacitors on the sample-and-hold amplifiers are not capable of holding a voltage indefinitely; therefore, the sampling rate must be sufficient to ensure that these voltages do not decay appreciably between samples. The sampling rate is a function of the capacitors as well as the frequency of the analog signal which is expected at the output of the converter.

At this point, you might be curious to know just how fast a signal must be sampled in order to preserve its integrity. Common sense leads to the conclusion that the more often the signal is sampled, the less the sample degrades between samples. On the other hand, if too few samples are taken, the signal degrades too much (the sample-and-hold capacitors discharge too much), and the signal information is lost. We would like to reduce the sampling rate to the minimum necessary to extract all the necessary information from the signal. The solution to this problem involves more than we have time for here, but the results are easy enough to apply.

First, if the signal in question is sinusoidal, it is necessary to sample at only twice the signal frequency. For instance, if the signal is a 5-kHz sine wave, it must be sampled at a rate greater than or equal to 10 kHz. In other words, a sample must be taken every 1/10000 s = 100 μs. What if the waveform is not sinusoidal? Any waveform that is periodic can be represented by a summation of sine and cosine terms, with each succeeding term having a higher frequency. In this case, it will be necessary to sample at a rate equal to twice the highest frequency of interest.

12.3.2 D/A Converter Testing

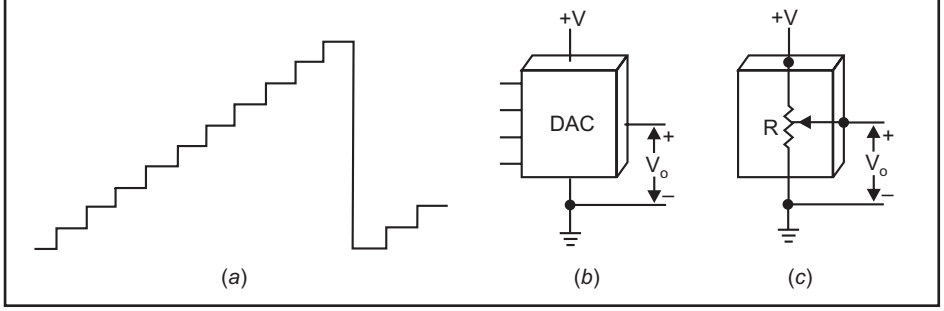
Two simple but important tests that can be performed to check the proper operation of the D/A converter are the steady-state accuracy test and the monotonicity test.

Notes

The steady-state accuracy test involves setting a known digital number in the input register, measuring the analog output with an accurate meter, and comparing with the theoretical value.

Checking for monotonicity means checking that the output voltage increases regularly as the input digital signal increases: This can be accomplished by using a counter as the digital input signal and observing the analog output on an oscilloscope. For proper monotonicity, the output waveform should be a perfect staircase waveform, as shown in Figure 12.16. The steps on the staircase waveform must be equally spaced and of the exact same amplitude. Missing steps, steps of different amplitude, or steps in a downward fashion indicate malfunctions.

Figure 12.16: Correct Output Voltage Waveform for Monotonicity Test

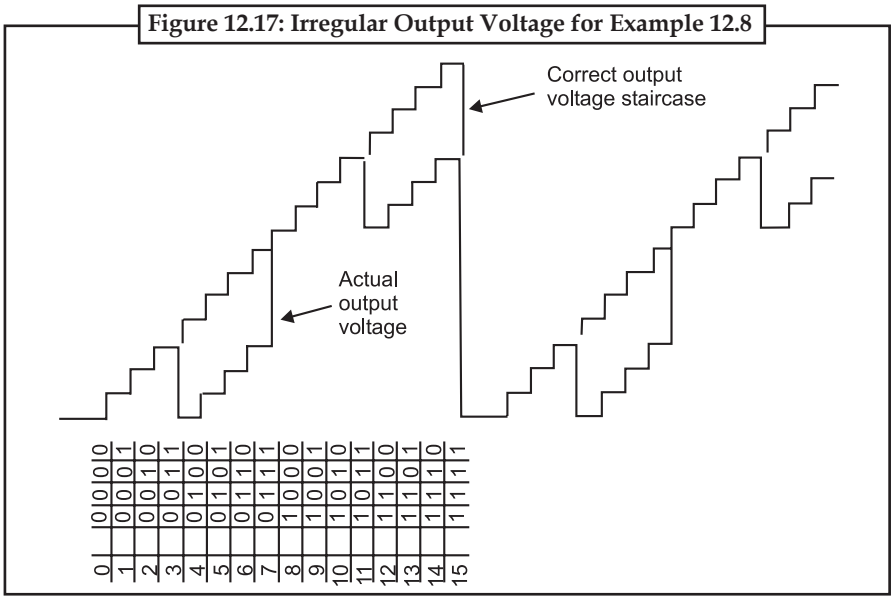


The monotonicity test does not check the system for accuracy, but if the system passes the test, it is relatively certain that the converter error is less than 1 LSB. Converter accuracy and resolution are the subjects of the next section.

A D/A converter can be regarded as a logic block having numerous digital inputs and a single analog output as seen in Figure 12.16b. It is interesting to compare this logic block with the potentiometer shown in Figure 12.16c. The analog output voltage of the D/A converter is controlled by the digital input signals, while the analog output voltage of the potentiometer is controlled by mechanical rotation of the potentiometer shaft. Considered in this fashion, it is easy to see how a D/A converter could be used to generate a voltage waveform (saw tooth, triangular, sinusoidal, etc.). It is, in effect, a digitally controlled voltage generator!

Example 12.8: Suppose that in the course of a monotonicity check on the 4-bit converter in Figure 12.13, the waveform shown in Figure 12.17 is observed. What is the probable malfunction in the converter?

Figure 12.17: Irregular Output Voltage for Example 12.8



Notes

Solution:

There is obviously some malfunction since the actual output waveform is not continuously increasing as it should be. The actual digital inputs are shown directly below the waveform. Notice that the converter functions correctly up to count 3. At count 4, however, the output should be 4 units in amplitude. Instead, it drops to 0. It remains 4 units below the correct level until it reaches count 8. Then, from count 8 to 11, the output level is correct. But again at count 12 the output falls 4 units below the correct level and remains there for the next four levels. If you examine the waveform carefully, you will note that the output is 4 units below normal during the time when the 2^2 bit is supposed to be high. This then suggests that the 2^2 bit is being dropped (i.e. the 2^2 input to the ladder is not being held high). This means that the 2^2-level amplifier is malfunctioning or the 2^2 AND gate is not operating properly. In any case, the monotonicity check has clearly shown that the second MSB is not being used and that the converter is not operating properly.

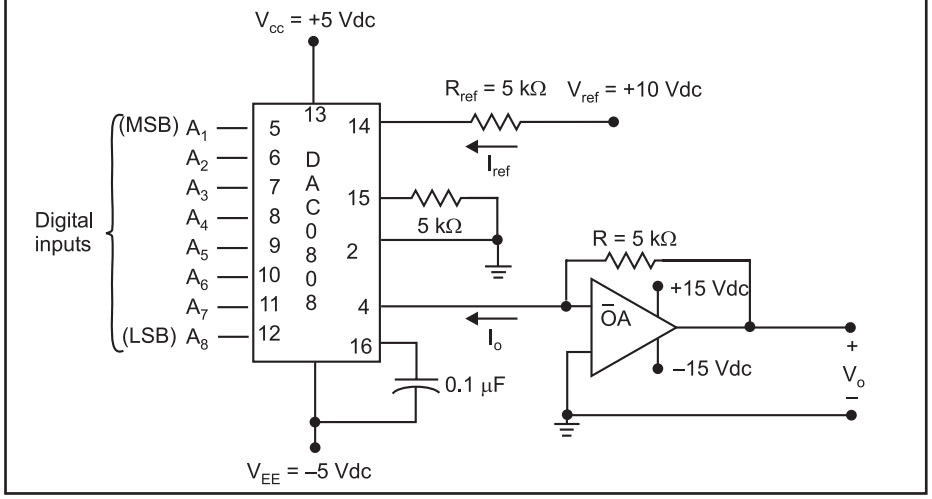
12.3.3 Available D/A Converters

D/A converters, as well as sample-and-hold amplifiers, are readily obtainable commercial products. Each unit is constructed in a single package; general-purpose economy units are available with 6-, 8-, 10-, and 12-bit resolution, and high-resolution units with up to 16-bit resolution are available.

An inexpensive and very popular D/A converter is the DAC0808, an 8-bit D/A converter available from National Semiconductor. Motorola manufactures an 8-bit D/A converter, the MC1508/1408. In Figure 12.18, a DAC0808 is connected to provide a full-scale output voltage of V_0 = +10 Vdc when all 8 digital inputs are 1s (high). If the 8 digital inputs are all 0s (low), the output voltage will be V_0 = 0 Vdc. Let us look at this circuit in detail.

First of all, two dc power-supply voltages are required for the DAC0808: Vcc = +5 Vdc and Vee = -15 Vdc. The 0.1-μF capacitor is to prevent unwanted circuit oscillations, and to isolate any variations in Vee. Pin 2 is ground (GND), and pin 15 is also referenced to ground through a resistor.

Figure 12.18: D/A Converters



The output of the D/A converter on pin 4 has a very limited voltage range (+0.5 to -0.6 V). Rather, it is designed to provide an output current I_0. The minimum current (all digital inputs low) is 0.0 mA, and the maximum current (all digital inputs high) is Iref. This reference current is established with the resistor at pin 14 and the reference voltage as

$$I_{ref} = V_{ref}/R_{ref} \quad ...(12.4)$$

Notes

The D/A converter output current I_0, is given as

$$I_0 = I_{ref}(A1/2 + A2/4 + A3/8 + ...+A8/256) \quad ...(12.5)$$

where A1, A2, A3, ..., A8 are the digital input levels (1 or 0).

The OA is connected as a current-to-voltage converter, and the output voltage is given as

$$V_0 = I_0 * R \quad ...(12.6)$$

Substituting Eqs. (12.4) and (12.5) into Eq. (12.6),

$$V_0 = V_{ref}/R_{Rref} * (A1/2 + A2/4 + A3/8 + ...+A8/256) * R \quad ...(12.7)$$

If we set the OA feedback resistor R equal to R_{ref}, then

$$V_0 = V_{ref} * (A1/2 + A2/4 + A3/8 + ...+A8/256) \quad ...(12.8)$$

Let us try out Eq. (12.8). Suppose all digital inputs are 0s (all low). Then

$$V_0 = V_{ref} * (0/2 + 0/4 + 0/8 +.....+0/256)$$

$$= V_{ref} * 0 = 0.0Vdc$$

Now, suppose all digital inputs are 1s (all high). Then

$$V_0 = V_{ref} * (1/2 + ¼ + 1/8 + ... + 1/256)$$

$$(V_{ref}) * (256/256) = 0.996 * V_{ref}$$

Since V_{ref} in Figure. 12.18 is + 10 Vdc, the output voltage is seen to have a range between 0.0 and +9.96 Vdc. It does not quite reach + 10 Vdc, but this is characteristic of this type of circuit. This circuit is essentially the current-mode operation discussed in the previous section and illustrated in Figure 12.12*b*.

Example 12.9: In Figure 12.18, A1 is high, A2 is high, A5 is high, A7 is high. The other digital inputs are all low. What is the output voltage V_0?

Solution:

$$V_0 = 10 * (1/2 + ¼ + 1/32 + 1/128) = 10 * 0.789 = 7.89V$$

Self Assessment

True or False:

1. The basic advantage in converting a digital signal into an equivalent analog signal change the *n* digital voltage levels into one equivalent analog voltage.

   (*a*) True (*b*) False

2. Resistive divider has two serious drawbacks: one of the first is the fact that each resistor in the network has a different value.

   (*a*) True (*b*) False

3. Either the resistive divider or the ladder can be used as the basis for a digital-to-analog (D/A) converter.

   (*a*) True (*b*) False

4. D/A converter can be regarded as a logic block having numerous digital inputs and a single analog output.

   (*a*) True (*b*) False

Notes

5. Two simple but important tests that can be performed to check the proper operation of the D/A converter are the steady-state accuracy test and the monotonicity test.

   (*a*) True (*b*) False

12.4 D/A Accuracy and Resolution

Two very important aspects of the D/A converter are the resolution and the accuracy of the conversion. There is a definite distinction between the two, and you should clearly understand the differences.

The accuracy of the D/A converter is primarily a function of the accuracy of the precision resistors used in the ladder and the precision of the reference voltage supply used. Accuracy is a measure of how close the actual output voltage is to the theoretical output value.

For example, suppose that the theoretical output voltage for a particular input should be +10 V. An accuracy of 10% means that the actual output voltage must be somewhere between +9 and + 11 V. Similarly, if the actual output voltage were somewhere between +9.9 and +10.1 V, this would imply an accuracy of 1%.

Resolution, on the other hand, defines the smallest increment in voltage that can be discerned. Resolution is primarily a function of the number of bits in the digital input signal; that is, the smallest increment in output voltage is determined by the LSB.

In a 4-bit system using a ladder, for example, the LSB has a weight of 1s. This means that the smallest increment in output voltage is of the input voltage. To make the arithmetic easy, let us assume that this 4-bit system has input voltage levels of + 16 V. Since the LSB has a weight of 1s a change in the LSB results in a change of 1 V in the output. Thus the output voltage changes in steps (or increments) of 1 V. The output voltage of this converter is then the staircase shown in Figure 12.16 and ranges from 0 to + 15 V in 1-V increments. This converter can be used to represent analog voltages from 0 to + 15 V, but it cannot resolve voltages into increments smaller than 1 V. If we desired to produce +4.2 V using this converter, therefore, the actual output voltage would be +4.0 V. Similarly, if we desired a voltage of +7.8 V, the actual output voltage would be +8.0 V. It is clear that this converter is not capable of distinguishing voltages finer than 1 V, which is the resolution of the converter.

If we wanted to represent voltages to a finer resolution, we would have to use a converter with more input bits. As an example, the LSB of a 10-bit converter has a weight of 1/1024. Thus the smallest incremental change in the output of this converter is approximately 1/1000 of the full-scale voltage. If this converter has a + 10-V full-scale output, the resolution is approximately + 10 * 1/l000 = 10 mV. This converter is then capable of representing voltages to within 10 mV.

Example 12.10: What is the resolution of a 9-bit D/A converter which uses a ladder network? What is this resolution expressed as a percent? If the full-scale output voltage of this converter is + 5 V, what is the resolution in volts?

Solution:

The LSB in a 9-bit system has a weight of 1/512. Thus this converter has a resolution of 1 part in 512. The resolution expressed as a percentage is 1/512 * 100% ≡ 0.2%. The voltage resolution is obtained by multiplying the weight of the LSB by the full-scale output voltage. Thus the resolution in volts is 1/512 * 5 ≡ 10 mV.

Example 12.11: How many bits are required at the input of a converter if it is necessary to resolve voltages to 5 mV and the ladder has +10 V full scales?

Notes

Solution:

The LSB of an 11-bit system has a resolution of 1/2048. This would provide a resolution at the output of 1/2048 * (+10) = +5 mV.

It is important to realize that resolution and accuracy in a system should be compatible. For example, in the 4-bit system previously discussed, the resolution was found to be 1 V. Clearly it would be unjustifiable to construct such a system to an accuracy of 0. 1%. This would mean that the system would be accurate to 16 mV but would be capable of distinguishing only to the nearest 1 V.

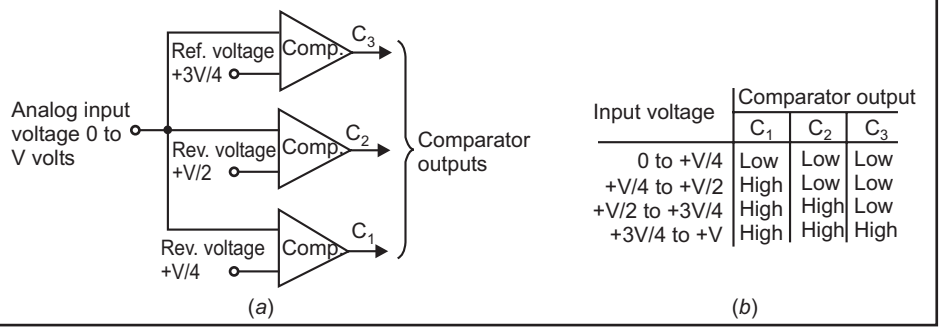
Similarly, it would be wasteful to construct the 11-bit system described in Example 12.11 to an accuracy of only 1 percent. This would mean that the output voltage would be accurate only to 100 mV, whereas it is capable of distinguishing to the nearest 5 mV.

12.5 A/D Converter—Simultaneous Conversion

The process of converting an analog voltage into an equivalent digital signal is known as analog-to-digital (A/D) conversion. This operation is somewhat more complicated than the converse operation of D/A conversion. A number of different methods have been developed, the simplest of which is probably the simultaneous method.

The simultaneous method of A/D conversion is based on the use of a number of comparator circuits. One such system using three comparator circuits is shown in Figure 12.19 on the next page. The analog signal to be digitized serves as one of the inputs to each comparator. The second input is a standard reference voltage. The reference voltages used are +V/4, +V/2, and +3V/4. The system is then capable of accepting an analog input voltage between 0 and +V.

Figure 12.19: Simultaneous A/D Conversion—(*a*) Logic Diagram, (*b*) Comparator Outputs for Input Voltage Ranges



Input voltage	Comparator output		
	C_1	C_2	C_3
0 to +V/4	Low	Low	Low
+V/4 to +V/2	High	Low	Low
+V/2 to +3V/4	High	High	Low
+3V/4 to +V	High	High	High

(*b*)

If the analog input signal exceeds the reference voltage to any comparator, that comparator turns on. (Let us assume that this means that the output of the comparator goes high.) Now, if all the comparators are off, the analog input signal must be between 0 and +V/4. If C1 is high (comparator C1 is on) and C2 and C3 are low, the input must be between +V/4 and +V/2 V. If C1 and C2 are high while C3 is low, the input must be between +V/2 and +3V/4. Finally, if all comparator outputs are high, the input signal must be between +3V/4 and +V. The comparator output levels for the various ranges of input voltages are summarized in Figure 12.19b.

Examination of Figure 12.19 reveals that there are four voltage ranges that can be detected by this converter. Four ranges can be effectively discerned by two binary digits (bits). The three comparator outputs can then be fed into a coding network to provide 2 bits which are equivalent to the input analog voltage. The bits of the coding network can then be entered into a flip-flop register for storage. The complete block diagram for such an A/D converter is shown in Figure 12.20.

In order to gain a clear understanding of the operation of the simultaneous A/D converter, let us investigate the 3-bit converter shown in Figure 12.21*a*. Notice that in order to convert the input

Notes

signal to a digital signal having 3 bits; it is necessary to have seven comparators (this allows a division of the input into eight ranges).

Figure 12.20: 2-Bit Simultaneous A/D Converter

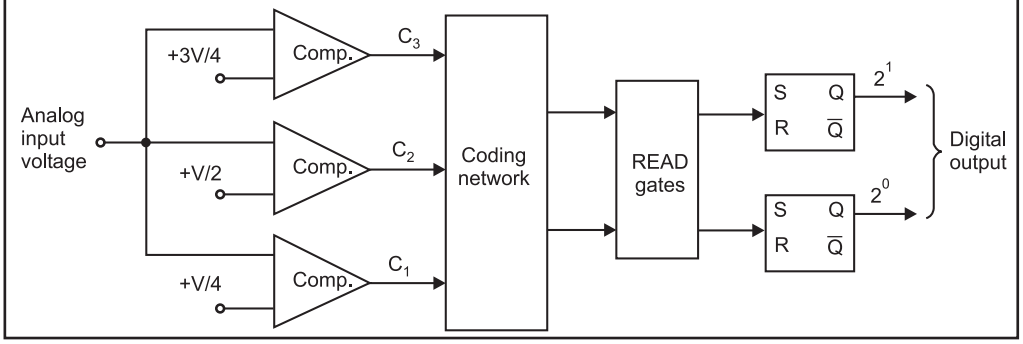
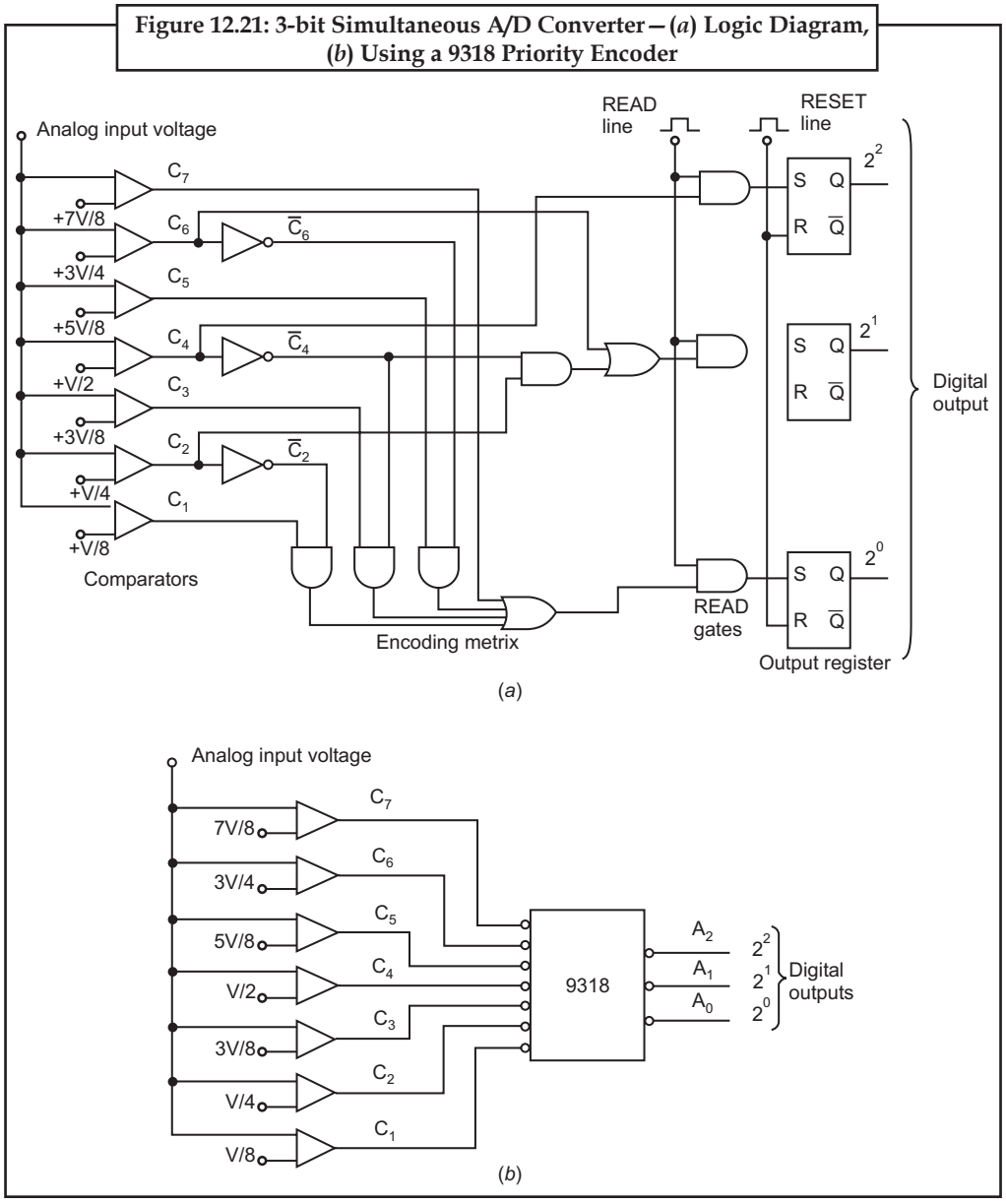


Figure 12.21: 3-bit Simultaneous A/D Converter – (*a*) Logic Diagram, (*b*) Using a 9318 Priority Encoder



Notes

Flash converter: Remember that three comparators were necessary for defining four ranges. In general, it can be said that $2^n - 1$ comparators are required to convert to a digital signal that has n bits. Some of the comparators have inverters at their outputs since both C and $\bar{C}$ are needed for the encoding matrix.

The encoding matrix must accept seven input levels and encode them into a 3-bit binary number (having eight possible states). Operation of the encoding matrix can be most easily understood by examination of the table of outputs in Figure 12.22.

The 2^2 bit is easiest to determine since it must be high (the 2^2 flip-flop must be set) whenever C_4 is high.

The 2^1 line must be high whenever C_2 is high and $\bar{C}_4$ is high, or whenever C_6 is high. In equation form, we can write $2^1 = C_2\bar{C}_4 + C_6$.

The logic equation for the 2^0 bit can be found in a similar manner; it is

$$2^0 = C_1\bar{C}_2 + C_3C_4 + C_5\bar{C}_6 + C_7$$

The transfer of data from the encoding matrix into the register must be carried out in two steps. First, a positive reset pulse must appear on the RESET line to reset all the flip-flops low. Then, a positive READ pulse allows the proper READ gates to go high and thus transfer the digital information into the flip-flops.

Interestingly, a convenient application for a 9318 priority encoder is to use it to replace all the digital logic as shown in Figure 12.21b. Of course, the inputs C_1, C_2, ..., C_7 must be TTL-compatible. In essence, the output of the 9318 is a digital number that reflects the highest-order zero input; this corresponds to the lowest reference voltage that still exceeds the input analog voltage.

The construction of a simultaneous A/D converter is quite straightforward and relatively easy to understand. However, as the number of bits in the desired digital number increases, the number of comparators increases very rapidly ($2^n - 1$), and the problem soon becomes unmanageable. Even though this method is simple and is capable of extremely fast conversion rates, there are preferable methods for digitizing numbers having more than 3 or 4 bits. Because it is so fast, this type of converter is frequently called a flash converter.

Figure 12.22: Logic Table for the Converter

Input voltage	Comparator for level							Binary output		
	C_1	C_2	C_3	C_4	C_5	C_6	C_7	2^2	2^1	2^0
0 to V/8	Low	Low	Low	Low	Low	Low	Low	0	0	0
V/8 to V/4	High	Low	Low	Low	Low	Low	Low	0	0	1
V/4 to 3V/8	High	High	Low	Low	Low	Low	Low	0	1	0
3V/8 to V/2	High	High	High	Low	Low	Low	Low	0	1	1
V/2 to 5V/8	High	High	High	High	Low	Low	Low	1	0	0
5V/8 to 3V/4	High	High	High	High	High	Low	Low	1	0	1
3V/4 to 7V/8	High	High	High	High	High	High	Low	1	1	0
7V/8 to V	High	High	High	High	High	High	High	1	1	1

The outputs are tri-state TTL = compatible. The flash A/D converter is capable of operation with a 25-MHz clock! It comes in a 24-pin DIP and requires two dc supply voltages—typically +5 Vdc and –5 Vdc. Possible applications include radar signal processing, video displays, high-speed instrumentation, and television broadcasting.

Did u know?

A time-stretch analog-to-digital converter (TS-ADC) digitizes a very wide bandwidth analog signal that cannot be digitized by a conventional electronic ADC, by time-stretching the signal prior to digitization.

Notes

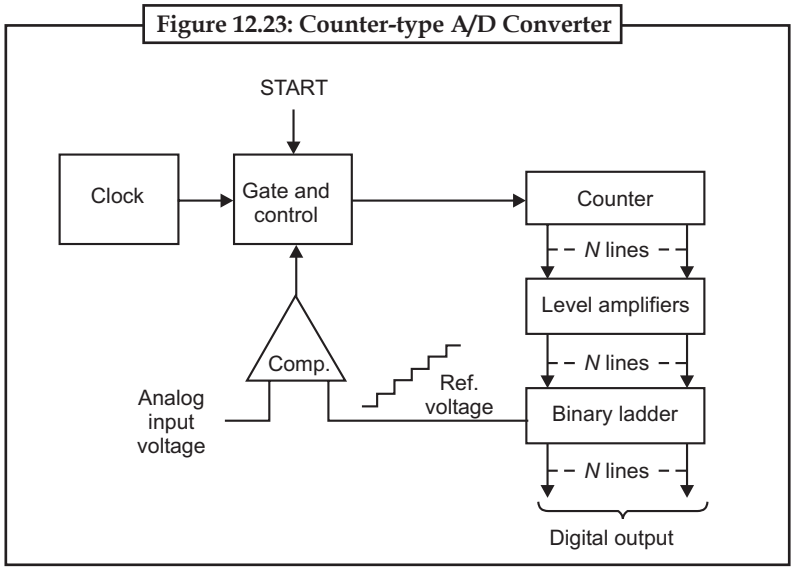
12.6 A/D Converter-Counter Method

A higher-resolution A/D converter using only one comparator could be constructed if a variable reference voltage were available. This reference voltage could then be applied to the comparator, and when it became equal to the input analog voltage, the conversion would be complete.

To construct such a converter, let us begin with a simple binary counter. The digital output signals will be taken from this counter, and thus we want it to be an *n*-bit counter, where n is the desired number of bits. Now let us connect the output of this counter to a standard binary ladder to form a simple D/A converter. If a clock is now applied to the input of the counter, the output of the binary ladder is the familiar staircase waveform shown in Figure 12.16. This waveform is exactly the reference voltage signal we would like to have for the comparator! With a minimum of gating and control circuitry, this simple D/A converter can be changed into the desired A/D converter.

Figure 12.23 shows the block diagram for a counter-type A/D converter. The operation of the counter is as follows. First, the counter is reset to all 0s. Then, when a convert signal appears on the START line, the gate opens and clock pulses are allowed to pass through to the input of the counter. The counter advances through its normal binary count sequence, and the staircase waveform is generated at the output of the ladder.

Figure 12.23: Counter-type A/D Converter



This wave form is applied to one side of the comparator, and the analog input voltage is applied to the other side. When the reference voltage equals (or exceeds) the input analog voltage, the gate is closed, the counter stops, and the conversion is complete. The number stored in the counter is now the digital equivalent of the analog input voltage.

Notice that this converter is composed of a D/A converter (the counter, level amplifiers, and the binary ladder), one comparator, a clock, and the gate and control circuitry. This can really be considered as a closed-loop control system. An error signal is generated at the output of the comparator by taking the difference between the analog input signal and the feedback signal (staircase reference voltage). The error is detected by the control circuit, and the clock is allowed to advance the counter. The counter advances in such a way as to reduce the error signal by increasing the feedback voltage. When the error is reduced to zero, the feedback voltage is equal to the analog input signal, the control circuitry stops the clock from advancing the counter, and the system comes to rest.

The counter-type A/D converter provides a very good method for digitizing to a high resolution. This method is much simpler than the simultaneous method for high resolution, but the conversion

Notes

time required is longer. Since the counter always begins at zero and counts through its normal binary sequence, as many as 2^n counts may be necessary before conversion is complete. The average conversion time is, of course, $2^n/2$ or 2^{n-1} counts.

The counter advances one count for each cycle of the clock, and the clock therefore determines the conversion rate. Suppose, for example, that we have a 10-bit converter. It requires 1024 clock cycles for a full-scale count. If we are using a 1-MHz clock, the counter advances 1 count every microsecond. Thus, to count full scale requires $1024 * 10^{-6} = 1.024$ ms. The converter reaches one-half full scale in half this time, or in 0.512 ins. The time required to reach one-half full scale can be considered the average conversion time for a large number of conversions.

Example 12.12: Suppose that the converter shown in Figure 12.23 is an 8-bit converter driven by a 500-kHz clock. Find (*a*) the maximum conversion time; (*b*) the average conversion time; (*c*) the maximum conversion rate.

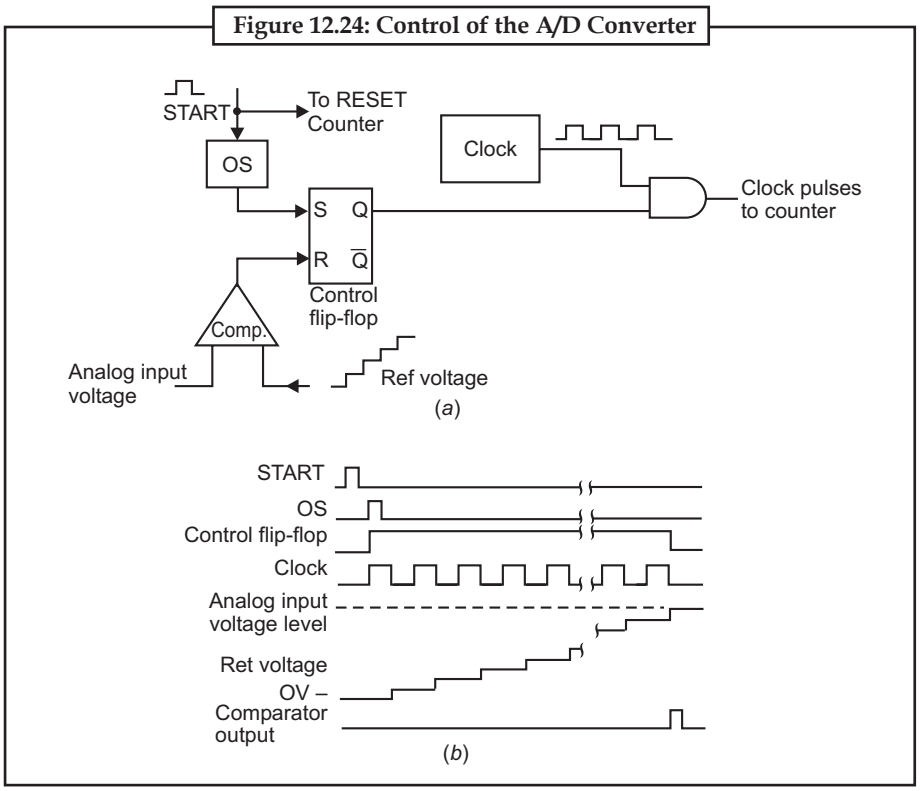
Solution:

(*a*) An 8-bit converter has a maximum of $2^8 = 256$ counts. With a 500-kHz clock, the counter advances at the rate of 1 count each 2 μs. To advance 256 counts requires $256 \times 2 \times 10^{-6} = 512 \times 10^{-6} = 512$ μs.

(*b*) The average conversion time is one-half the maximum conversion time. Thus it is $1/2 \times 0.512 \times 10^{-3} = 0.256$ ms.

(*c*) The maximum conversion rate is determined by the longest conversion time. Since the converter has a maximum conversion time of 0.512 ms, it is capable of making at least $1/(0.512 * 10^{-3}) \equiv 1953$ conversions per second.

Figure 12.24: Control of the A/D Converter



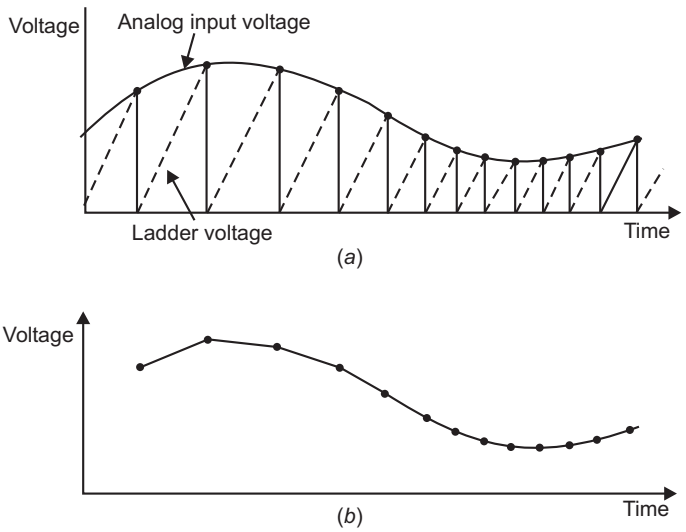
Notes

Figure 12.24 shows one method of implementing the control circuitry for the converter shown in Figure 12.23. The waveforms for one conversion are also shown. A conversion is initiated by the receipt of a START signal. The positive edge of the START pulse is used to reset all the flip-flops in the counter and to trigger the one-shot. The output of the one-shot sets the control flip-flop, which makes the AND gate true and allows clock pulses to advance the counter.

The delay between the RESET pulse to the flip-flops and the beginning of the clock pulses (ensured by the one-shot) is to ensure that all flip-flops are reset before counting begins. This is a definite attempt to avoid any racing problems.

With the control flip-flop set, the counter advances through its normal count sequence until the staircase voltage from the ladder is equal to the analog input voltage. At this time, the comparator output changes state, generating a positive pulse which resets the control flip-flop. Thus the AND gate is closed and counting ceases. The counter now holds a digital number which is equivalent to the analog input voltage. The converter remains in this state until another conversion signal is received.

Figure: 12.25: (*a*) Digitizing an Analog Voltage, (*b*) Reconstructed Signal from the Digital Data

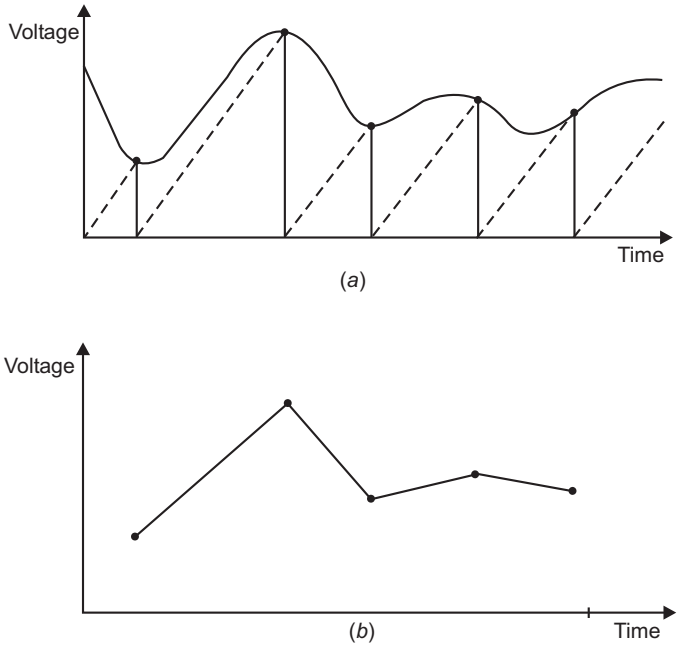


If a new start signal is generated immediately after each conversion is completed, the converter will operate at its maximum rate. The converter could then be used to digitize a signal as shown in Figure 12.25a. Notice that the conversion times in digitizing this signal are not constant but depend on the amplitude of the input signal. The analog input signal can be reconstructed from the digital information by drawing straight lines from each digitized point to the next. Such a reconstruction is shown in Figure 12.25b; it is, indeed, a reasonable representation of the original input signal. In this case, it is important to note that the conversion times are smaller than the transient time of the input waveform.

On the other hand, if the transient time of the input waveform approaches the conversion time, the reconstructed output signal is not quite so accurate. Such a situation is shown in Figure 12.26a and b. In this case, the input waveform changes at a rate faster than the converter is capable of recognizing. Thus the need for reducing conversion time is apparent.

Notes

Figure 12.26 (*a*) Digitizing an Analog Voltage, (*b*) Reconstructed Signal from the Digital Data



Task Prepare a truth table and circuit diagram for D/A converter.

12.7 Continuous A/D Conversion

An obvious method for speeding up the conversion of the signal as shown in Figure 12.26 is to eliminate the need for resetting the counter each time a conversion is made. If this were continuous-type done, the counter would not begin at zero each time, but instead would begin at the value A/D converter of the last converted point. This means that the counter would have to be capable of counting either up or down. This is no problem; we are already familiar with the operation of up-down counters.

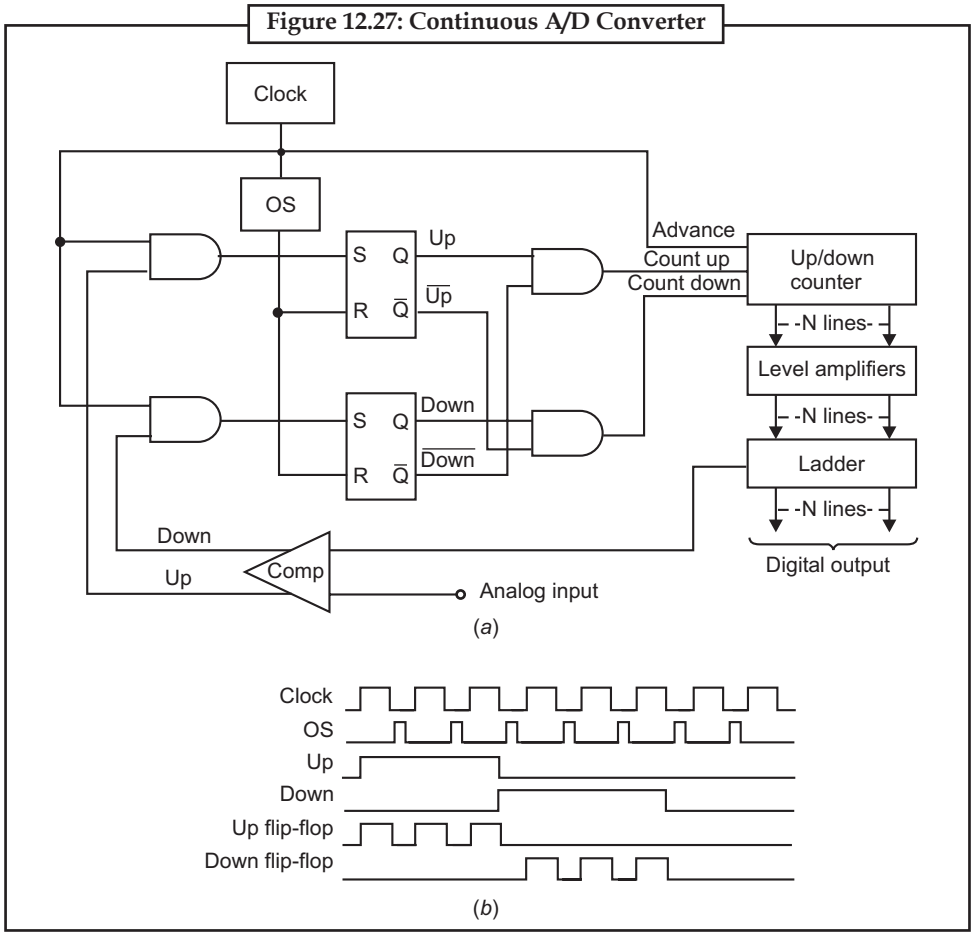
There is, however, the need for additional logic circuitry, since we must decide whether to count up or down by examining the output of the comparator. An A/D converter which uses an up-down counter is shown in Figure 12.27. This method is known as continuous conversion, and thus the converter is called a continuous type A/D converter.

The D/A portion of this converter is the same as those previously discussed, with the exception of the counter. It is an up-down counter and has the up and down count controls lines in addition to the advance line at its input.

The output of the ladder is fed into a comparator which has two outputs instead of one as before. When the analog voltage is more positive than the ladder output, the up output of the comparator is high.

Notes

Figure 12.27: Continuous A/D Converter



When the analog voltage is more negative than the ladder output, the down output is high. If the up output of the comparator is high, the AND gate at the input of the up flip-flop is open, and the first time the clock goes positive, the up flip-flop is set. If we assume for the moment that the down flip-flop is reset, the AND gate which controls the count-up line of the counter will be true and the counter will advance one count. The counter can advance only one count since the output of the one-shot resets both the up and the down flip-flops just after the clock goes low. This can then be considered as one count-up conversion cycle.

Notice that the AND gate which controls the count-up line has inputs of up and down. Similarly, the count-down line AND gate has inputs of down and up. This could be considered an exclusive-OR arrangement and ensures that the count-down and count-up lines cannot both be high at the same time.

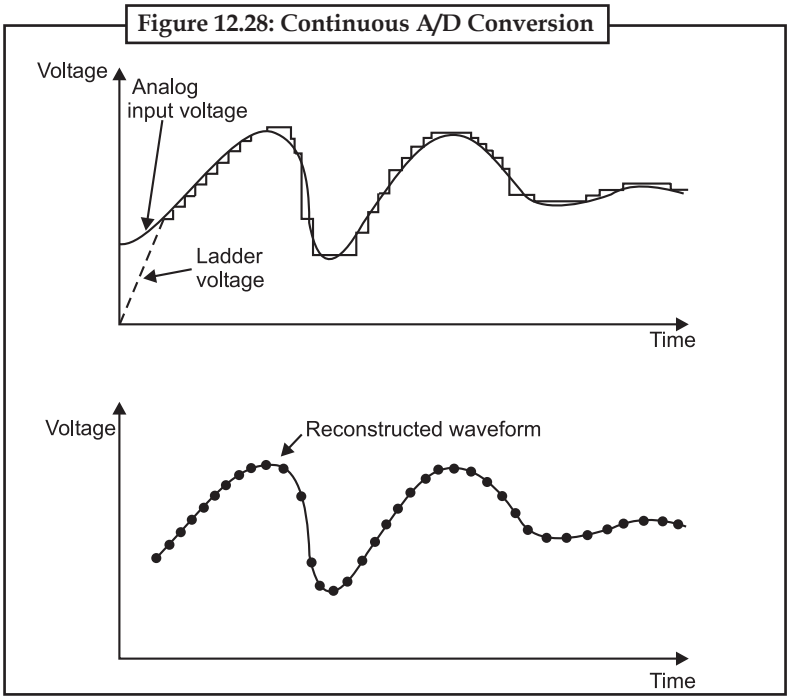
As long as the up line out of the comparator is high, the converter continues to operate one conversion cycle at a time. At the point where the ladder voltage becomes more positive than the analog input voltage, the up line of the comparator goes low and the down line goes high. The converter then goes through a count-down conversion cycle. At this point, the ladder voltage is within 1 LSB of the analog voltage, and the converter oscillates about this point. This is not desirable since we want the converter to cease operation and not jump around the final value. The trick here is to adjust the comparator such that its outputs do not change at the same time.

We can accomplish this by adjusting the comparator such that the up output will not go high unless the ladder voltage is more than 1/2 LSB below the analog voltage. Similarly, the down output will not go high unless the ladder voltage is more than 1/2 LSB above the analog voltage. This is called centering on the LSB and provides a digital output which is within 1/2 LSB.

Notes

A waveform typical of this type of converter is shown in Figure 12.28. You can see that this converter is capable of following input voltages that change at a much faster rate.

Figure 12.28: Continuous A/D Conversion



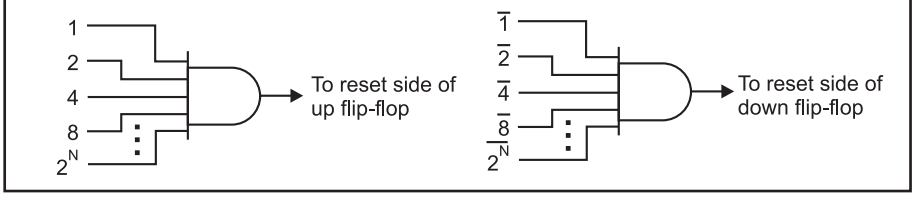
Example 12.13: Quite often, additional circuitry is added to a continuous converter to ensure that it cannot count off scale in either direction. For example, if the counter contained all ls, it would be undesirable to allow it to progress through a count-up cycle, since the next count would advance it to all Os. We would like to design the logic necessary to prevent this.

Solution:

The two limit points which must be detected are all 1s and all 0s in the counter. Suppose that we construct an AND gate having the 1 sides of all the counter flip-flops as its inputs. The output of this gate will be true whenever the counter contains all 1s. If the gate is then connected to the reset side of the up flip-flop, the counter will be unable to count beyond all 1s.

Similarly, we might construct an AND gate in which the inputs are the 0 sides of all the counter flip-flops. The output of this gate can be connected to the reset side of the down flip-flop, and the counter will then be unable to count beyond all 0s. The gates are shown in Figure 12.29.

Figure 12.29: Count-Limiting Gates for the Converter



12.8 A/D Techniques

There are a variety of other methods for digitizing analog signals-too many to discuss in detail. Nevertheless, we shall take the time to examine two more techniques and the reasons for their importance.

Notes

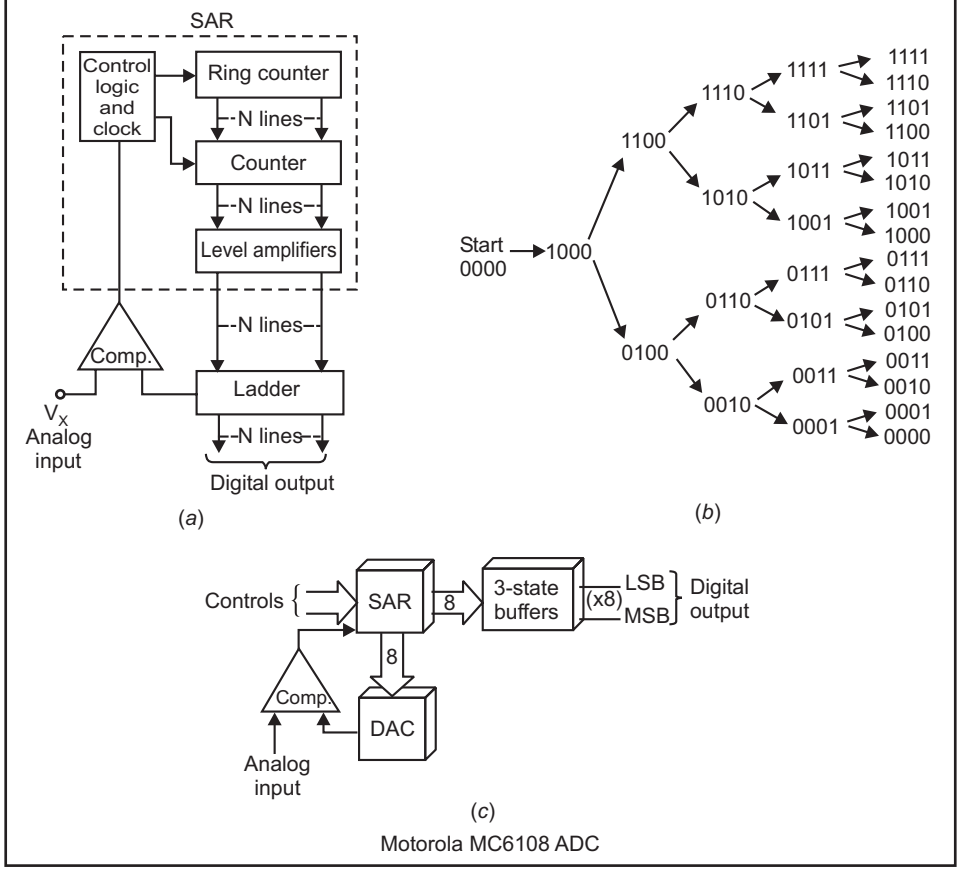
Probably the most important single reason for investigating other methods of conversion is to determine ways to reduce the conversion time. Recall that the simultaneous converter has a very fast conversion time. The counter converter is simple logically but has a relatively long conversion time. The continuous converter has a very fast conversion time once it is locked on the signal but loses this advantage when multiplexing inputs.

12.8.1 Successive Approximation

If multiplexing is required, the successive-approximation converter is most useful. The block diagram for this type of converter is shown in Figure 12.30a. The converter operates by successively dividing the voltage ranges in half. The counter is first reset to all 0s, and the MSB is then set. The MSB is then left in or taken out (by resetting the MSB flip-flop) depending on the output of the comparator. Then the second MSB is set in, and a comparison is made to determine whether to reset the second MSB flip-flop. The process is repeated down to the LSB, and at this time the desired number is in the counter. Since the conversion involves operating on one flip-flop at a time, beginning with the MSB, a ring counter may be used for flip-flop selection.

The successive-approximation method thus is the process of approximating the analog log voltage by trying 1 bit at a time beginning with the MSB. The operation is shown in diagram form in Figure 12.30b. It can be seen from this diagram that each conversion takes the same time and requires one conversion cycle for each bit. Thus the total conversion time is equal to the number of bits, n times the time required for one conversion cycle. One conversion cycle normally requires one cycle of the clock. As an example, a 10-bit converter operating with a 1-MHz clock has a conversion time of $10 * 10^{-6} = l0^{-5} = 10$ μs.

Figure 12.30: Successive Approximation Converter



Notes

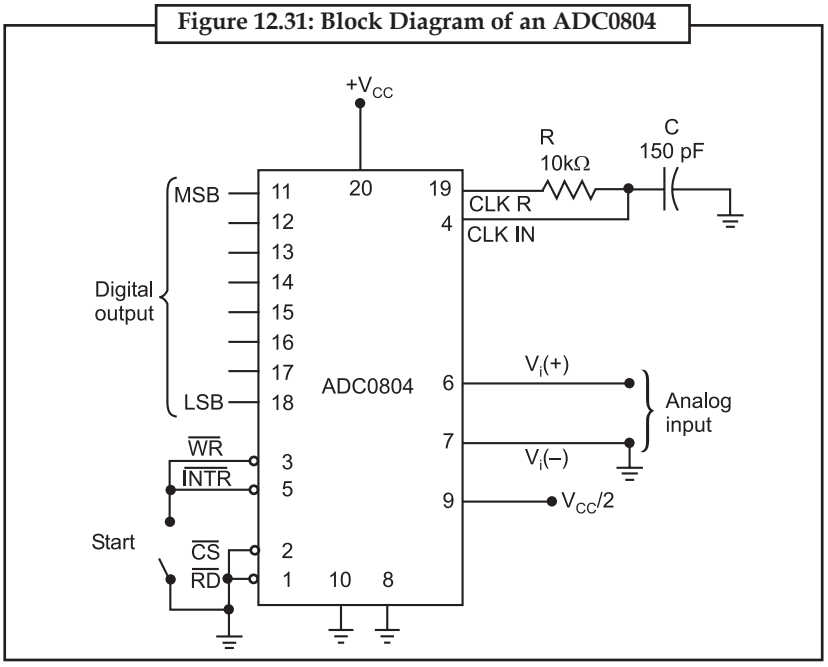
Successive-approx imation register: When dealing with conversion times this short, it is usually necessary to take into account the other delays in the system (e.g., switching time of the multiplexer, settling time of the ladder network, comparator delay, and settling time).

Free-running mode: All the logic blocks inside the dashed line in Figure 12.30a, or some equivalent arrangement, are frequently constructed on a single MSI chip; this chip is called a successive approximation register (SAR). For example, the Motorola MC6108 shown in Figure 12.30c is an 8-bit microprocessor-compatible A/D converter that includes an SAR, D/A conversion capabilities, control logic, and buffered digital outputs, in a 28-pin DIP.

12.8.2 The ADC0804

The ADC0804 is an inexpensive and very popular A/D converter which is available from a number of different manufacturers, including National Semiconductor. The ADC0804 is an 8-bit CMOS microprocessor-compatible successive-approximation A/D converter that is supplied in a 20-pin DIP. It is capable of digitizing an analog input voltage within the range 0 to +5 Vdc, and it only requires a single dc supply voltage—usually +5 Vdc. The digital outputs are both TTL- and CMOS-compatible.

Figure 12.31: Block Diagram of an ADC0804



The block diagram of an ADC0804 is shown in Figure 11.31. In this case, the controls are wired such that the converter operates continuously. This is the so-called free-running mode. The 10–kΩ resistor, along with the 150–pF capacitor, establishes the frequency of operation according to $f \approx 1/1.1(RC)$. In this case,

$f \approx 1/(1.1 * (10^4 * 1.5 * 150pF))$

$=1/(1.1 * (10^4 * 1.5 * 10^{-12}))$

A momentary activation of the START switch is necessary to begin operation.

Notes

12.8.3 Section Counters

Another method for reducing the total conversion time of a simple counter converter is to divide the counter into sections. Such a configuration is called a section counter. To determine how the total conversion time might be reduced by this method, assume that we have a standard 8-bit counter. If this counter is divided into two equal counters of 4 bits each, we have a section converter. The converter operates by setting the section containing the four LSBs to all 1s and then advancing the other sections until the ladder voltage exceeds the input voltage. At this point the four LSBs are all reset, and this section of the counter is then advanced until the ladder voltage equals the input voltage.

Notice that a maximum of $2^4 = 16$ counts is required for each section to count full scale. Thus this method requires only $2 * 2^4 = 2^5 = 32$ counts to reach full scale. This is a considerable reduction over the $2^8 = 256$ counts required for the straight 8-bit counter. There is, of course, some extra time required to set the counters initially and to switch from counter to counter during the conversion. This logical operation time is very small, however, compared with the total time saved by this method.

This type of converter is quite often used for digital voltmeters, since it is very convenient to divide the counters by counts of 10. Each counter is then used to represent one of the digits of the decimal number appearing at the output of the voltmeter.


Did u know?

An ADC can resolve a signal to only a certain number of bits of resolution, called the effective number of bits (ENOB).

12.9 Dual-Slope A/D Conversion

Up to this point, our interest in different methods of A/D conversion has centred on reducing the actual conversion time. If a very short conversion time is not a requirement, there are other methods of A/D conversion that are simpler to implement and much more economical. Basically, these techniques involve comparison of the unknown input voltage with a reference voltage that begins at zero and increases linearly with time. The time required for the reference voltage to increase to the value of the unknown voltage is directly proportional to the magnitude of the unknown voltage, and this time period is measured with a digital counter. This is referred to as a single-ramp method, since the reference voltage is sloped like a ramp. A variation on this method involves using an operational amplifier integrating circuit in a dual-ramp configuration. The dual-ramp method is very popular, and widely used in digital voltmeters and digital panel meters. It offers good accuracy, good linearity, and very good noise-rejection characteristics.

12.9.1 Single-Ramp A/D Converter

Let us take a look at the single-ramp A/D converter in Figure 12.32. The heart of this converter is the ramp generator. This is a circuit that produces an output voltage ramp as shown in Figure 12.33a. The output voltage begins at zero and increases linearly up to a maximum voltage Vm. It is important that this voltage be a straight line-that is, it must have a constant slope. For instance, if V_m = 1.0 Vdc, and it takes 1.0 ms for the ramp to move from 0.0 up to 1.0 V, the slope is 1 V/ms, or 1000 V/s.

Notes

Figure 12.32: Single-ramp A/D Converters

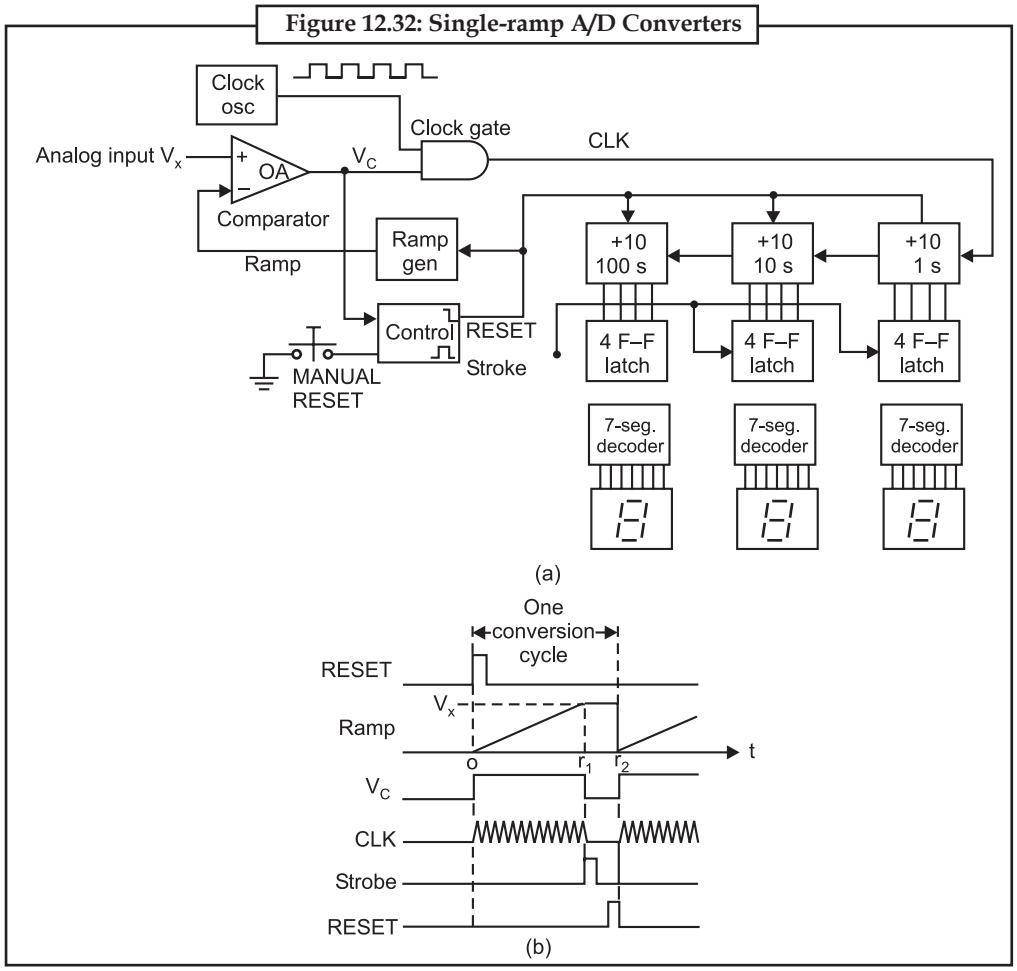
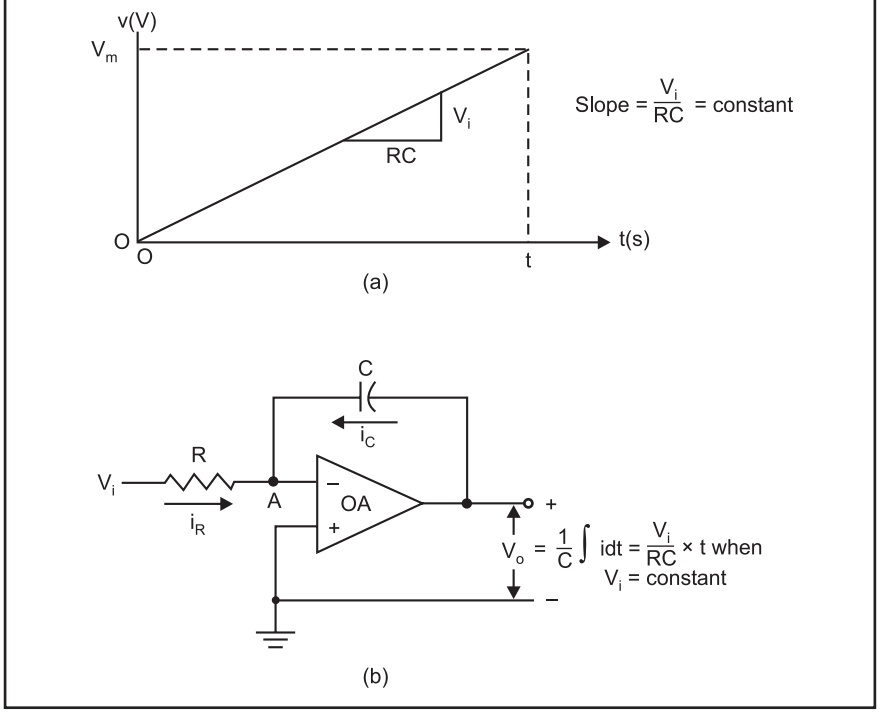


Figure 12.33: An Integrating Circuit



Notes

This ramp generator can be constructed in a number of different ways. One way might be to use a D/A converter driven by a simple binary counter. This would generate the staircase waveform previously discussed and shown in Figure 12.16a. A second method is to use an operational amplifier (OA) connected as an integrator as shown in Figure 12.33b. For this circuit, if V_i is a constant, the output voltage is given by the relationship $V_o = (V_i/RC)t$. Since V_i, R, and C are all constants, this is the equation of a straight line that has a slope (V_i/RC) as shown in Figure 12.33a. Now that we have a way to generate a voltage ramp and we understand its characteristics, let us return to the converter in Figure 12.32.

We assume that the clock is running continuously and that any input voltage Vx that we wish to digitize is positive. If it is not, there are circuits that we can use to adjust for negative input signals. The three decade counters are connected in cascade, and their outputs can be strobed into three 4-flip-flop latch circuits. The latches are then decoded by seven-segment decoders to drive the LED displays as units, tens, and hundreds of counts. We can begin a conversion cycle by depressing the MANUAL RESET switch.

Refer carefully to the logic diagram and the waveforms in Figure 12.32. MANUAL RESET generates a RESET pulse that clears all the decade counters to 0s and resets the ramp voltage to zero. Since VX is positive and RAMP begins at zero, the output of the comparator OA, V_c must be high. This voltage enables the CLOCK gate allowing the clock, CLK, to be applied to the decade counter. The counter begins counting upward, and the RAMP continues upward until the ramp voltage is equal to the unknown input V_x. At this point, time t_1, the output of the comparator V, goes low, thus disabling the CLOCK gate and the counters cease to advance. Simultaneously, this negative transition on V_c generates a STROBE signal in the CONTROL box that shifts the contents of the three decade counters into the three 4-flip-flop latch circuits. Shortly thereafter, a reset pulse is generated by the CONTROL box that resets the RAMP and clears the decade counters to O_s, and another conversion cycle begins. In the meantime, the contents of the previous conversion are contained in the latches and are displayed on the seven-segment LEDs.

As a specific example, suppose that the clock in Figure 12.32 is set at 1.0 MHz and the ramp voltage slope is 1.0 V/ms. Note that the decade counters have the ability to store and display any decimal number from 000 up to 999. From the beginning of a conversion cycle, it will require 999 clock pulses (999 μs) for the counters to advance full scale. During this same time period, the ramp voltage will have increased from 0.0 V up to 999 mV. So, this circuit as it stands will display the value of any input voltage between 0.0 V and 999 mV.

In effect, we have a digital voltmeter! For instance, if V_x = 345 mV, it will require 345 clock pulses for the counter to advance from 000 to 345, and during the same time period the ramp will have increased to 345 mV. So, at the end of the conversion cycle, the display output will read 345—we supply the units of mille-volts.

One weakness of the single-slope A/D converter is its dependency on an extremely accurate ramp voltage. This in turn is strongly dependent on the values of R and C and variations of these values with time and temperature. The dual-slope A/D converter overcomes these problems.

12.9.2 Dual-Slope A/D Converter

The logic diagram for a basic dual-slope A/D converter is given in Figure 12.34. With the exception of the ramp generator and the comparator, the circuit is similar to the single slope A/D converter in Figure 12.32. In this case, the integrator forms the desired, ramp—in fact, two different ramps—as the input is switched first to the unknown input voltage VX and then to a known reference voltage V_r. Here is how it works:

Notes

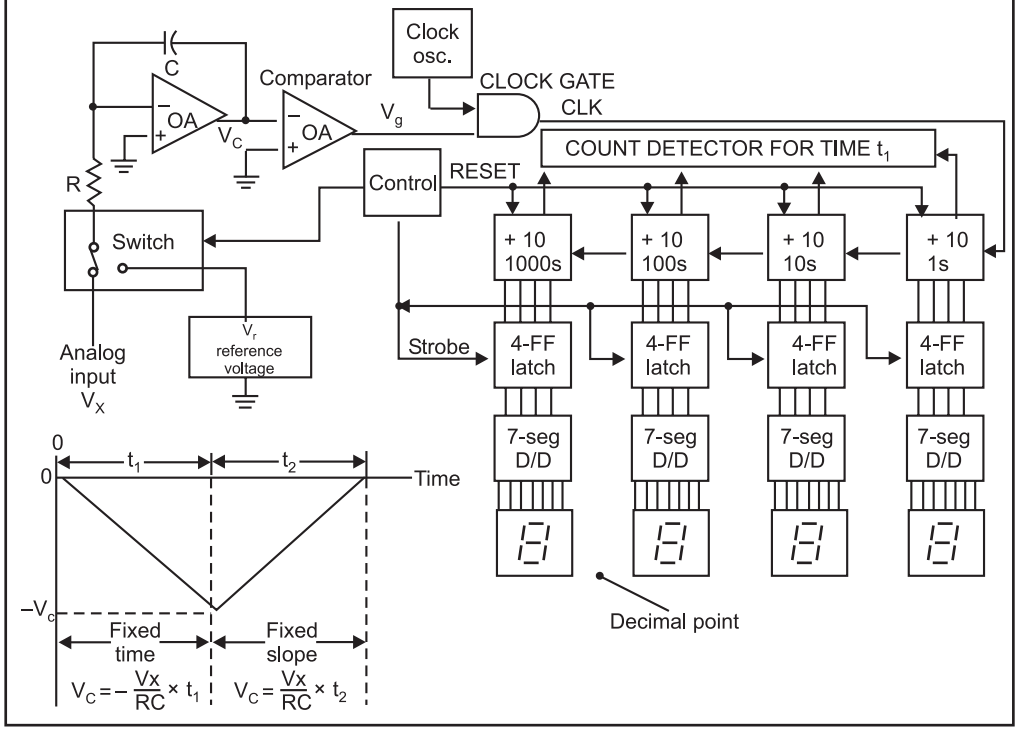
We begin with the assumptions that the clock is running, and that the input voltage V_x is positive. A conversion cycle begins with the decade counters cleared to all 0s, the ramp reset to 0.0 V, and the input switched to the unknown input voltage V_X. Since V_X is positive, the integrator output Vc will be a negative ramp. The comparator output V_g is thus positive and the clock is allowed to pass through the CLOCK GATE to the counters. We allow the ramp to proceed for a fixed time period t_1, determined by the count detector for time t,. The actual voltage V, at the end of the fixed time period t_1, will depend on the unknown input V_X, since we know that $V_c = -(V_X/RC) * t_1$ for an integrator.

When the counter reaches the fixed count at time ti, the CONTROL unit generates a pulse to clear the decade counters to all 0s and switch the integrator input to the negative reference voltage V_r. The integrator will now begin to generate a ramp beginning at - V_c and increasing steadily upward until it reaches 0.0 V. All this time, the counter is counting, and the conversion cycle ends when V_c = 0.0 V since the CLOCK GATE is now disabled. The equation for this positive ramp is $V_c = (Vr/RC) * t_2$. In this case, the slope of this ramp (V_r/RCI is constant, but the time period t_2 is variable.

In fact, since the integrator output voltage begins at 0.0 V, integrates down to $-V_c$ and then integrates back up to 0.0 V, we can equate the two equations given for V_c. That is:

$$(V_X * t_1)/RC = (V_r * t_2)/RC$$

Figure 12.34: Dual-slope A/D Converter



The value RC will cancel from both sides, leaving

$$V_X = V_r * (t_1/t_2)$$

Since V_r is a known reference voltage and t_1 is a predetermined time, clearly the unknown input voltage is directly proportional to the variable time period t_2. However, this time period is exactly the contents of the decade counters at the end of a conversion cycle! The obvious

Notes

advantage here is that the RC terms cancel from both sides of the equation above—in other words, this technique is free from the absolute values of either R or C and also from variations in either value.

As a concrete example, let us suppose that the clock in Figure 12.34 is 1.0 MHz, the reference voltage is –1.0 Vdc, the fixed time period t1 is 1000 μs, and the RC time constant of the integrator is set at RC = 1.0 ms. During the time period t_1 the integrator voltage V_c will ramp down to –1.0 Vdc if V_X = 1.0 V. Then, during, time t_2, V_c. will ramp all the way back up to 0.0 V, and this will require a time of 1000 μs, since the slope of this ramp is fixed at 1.0 V/ms. The output display will now read 1000, and with placement of a decimal as shown, this reads 1.000 V.

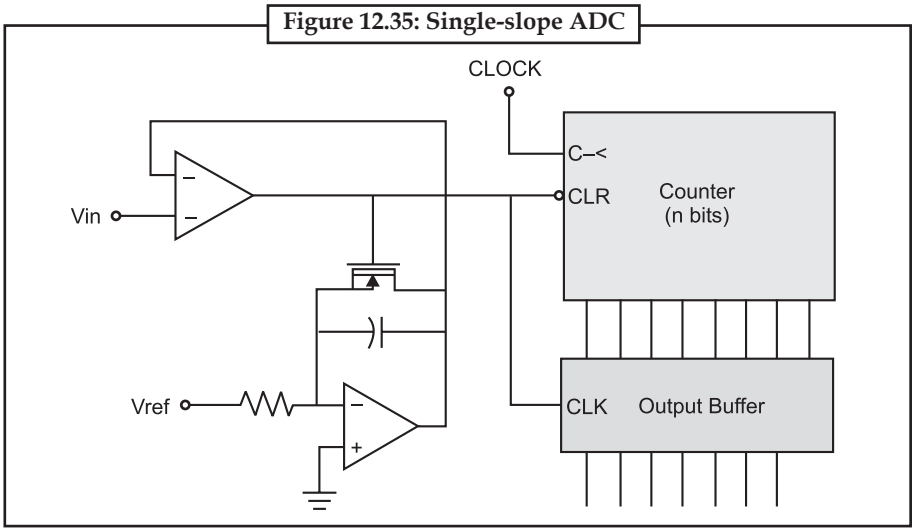
Another way of expressing the operation of this A/D converter is to solve the equation $V_X = V_r(t_2/t_1)$ for t_2, since t_2 is the digital readout. Thus $t_2 = (V_X/V_r)ti$. If the same values as given above are applied, an unknown input voltage V_X = 2.75 V will be digitized and the readout will be t_2 = (2.7511.0)1000 = 2750, or 2.75 V, using the decimal point on the display. Notice that we have used t_1 = 1000, the number of clock pulses that occur during the time period t_1. Likewise, t_2 is the number of clock pulses that occur during the time period t_2.

The analog signal is continuous in time and it is necessary to convert this to a flow of digital values.

12.10 Single-Slope A/D Converter

In Figure 12.35, you can see a single-slope ADC. If you pay close attention, you will see that it is very similar to a ramp counter ADC, as it uses a counter, but instead of using a DAC for generating the comparison voltage, it uses a circuit called integrator, which is basically formed by a capacitor, a resistor and an operational amplifier (op amp). The MOSFET transistor makes the necessary control circuit.

Figure 12.35: Single-slope ADC



The integrator produces a saw-tooth waveform on its output, from zero to the maximum possible analog voltage to be sampled, set by $-V_{ref}$. The minute the waveform is started, the counter starts counting from 0 to 2^n – 1, where n is the number of bits implemented by the ADC. When the voltage found at $_{Vin}$ (the analog signal) is equal to the voltage achieved by the triangle waveform generated by the integrator, the control circuit captures the last value produced by the counter (by trigging the output buffer clock pin), which will be the digital correspondent of the analog sample being converted. At the same time, it resets the counter and the integrator, starting the conversion of the next sample.

Notes

Like the successive approximation ADC, this circuit uses an output buffer, meaning that the last converted value can be read while the ADC is converting the current value.

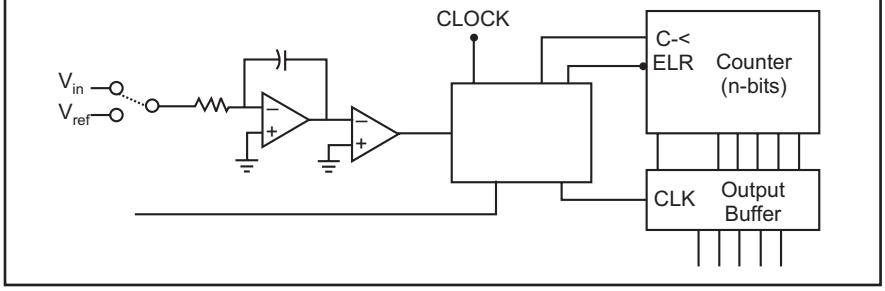
Even though its design is simpler than ramp counter design, it is still based on a counter, and thus suffering from the same basic problem found on ramp counter design: speed. It requires up to 2^n - 1 clock cycles to convert each sample. For an eight-bit ADC, it would take up to 255 clock cycles to convert a single sample. For a 16-bit ADC it would take up to 65,535 clock cycles to convert one sample.

12.11 Dual-slope A/D Converter

Another popular design based on this one is called dual-slope ADC, which solves an inherent single-slop problem called calibration drift, which leads to inaccuracy over time because the integrator is not linked to the clock signal (i.e. the saw-tooth waveform is not synchronized with the counter clock).

A classic dual-slope ADC can be seen in Figure 12.36.

Figure 12.36: Dual-slope ADC

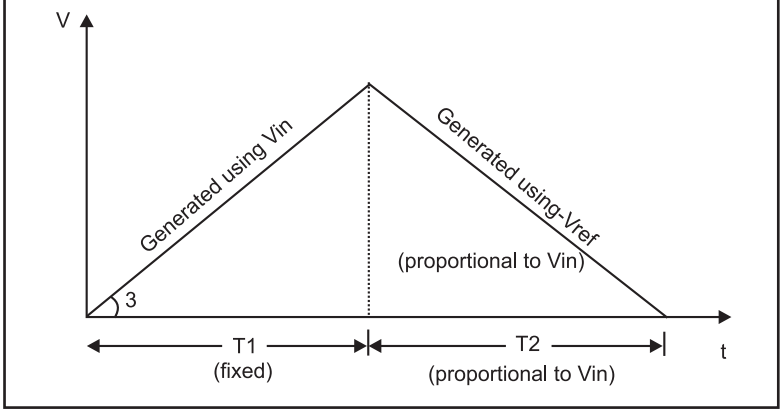


The analog switch first connects V_{in} to the integrator. With that, the integrator starts generating the saw-tooth waveform, and the switch position will remain set at V_{in} during a fixed number of clock cycles. When this number of clock cycles is reached, the analog switch moves its position to allow V_{ref} to enter the integrator. Since V_{ref} is a negative voltage, the saw-tooth waveform goes towards zero, using a number of clock cycles proportional of the V_{in} value.

For a better understanding, see Figure 12.36, where we show the waveform at the integrator output. So, T1 is fixed, while T2 duration is proportional to the value of V_{in}. V_{in} sets the slope angle: the higher V_{in} is, the higher the angle will be.

$$T2 = T1 * V_{in}/V_{ref}.$$

Figure 12.37: Waveform Found at the Integrator Output

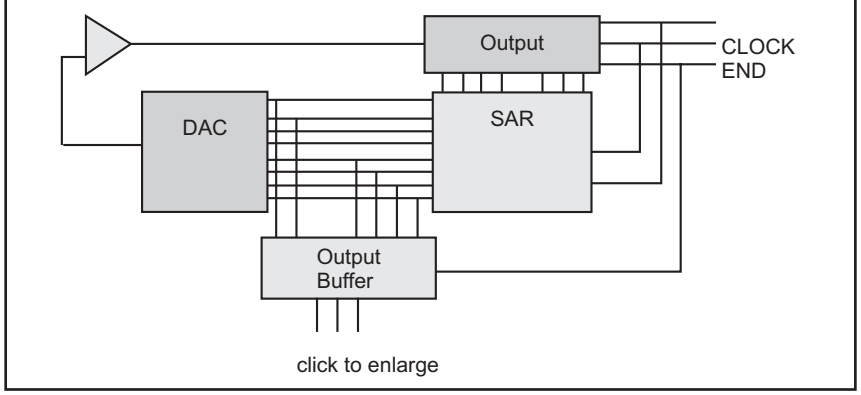


Notes

12.12 Successive Approximation A/D Converter

The second classical ADC circuit using DAC design is called successive approximation, which is the most used one, shown in Figure 12.38. Vin is the analog input and Dn through D0 are the digital outputs. As you can see, it uses a buffer, so the digital data is still available while the converter is processing the next sample. SAR stands for Successive Approximation Register. It has the same control signals as the ramp counter ADC: START, which commands the ADC to start the conversion, CLOCK and END, which tells us that the conversion of that particular sample has finished.

Figure 12.38: Successive Approximation ADC



While the ramp counters ADC does the analog-to-digital conversion counting from 0 to the maximum possible value ($2^n - 1$) until it "finds" the correct digital value for V_{in}, the successive approximation ADC starts first setting the MSB (most significant bit, on an eight-bit ADC it would be D7). In order to facilitate the explanations below, consider an eight-bit ADC.

The comparison between V_{in} and the DAC output will tell the control unit if this bit should remain set at 1 or should be set at 0, as the op amp will tell right away the control unit if the sample value is greater or lower than 128 (2^7). Then D6 is set to one, and from the comparison done by the op amp, the control unit will know if this bit should remain set or not. And so on.

The good thing about the successive approximation ADC is its speed. At the worst case it will find the correct digital value for the sample at n clock cycles, where n is the number of bits used. For an eight-bit ADC, the digital value for each sample can be found in up to eight clock cycles (compare to 255 on the ramp counter), and for a 16-bit ADC the digital value for each sample can be found in up to 16 clock cycles (compare to 65,535 on the previous circuit).

And, as we mentioned, another great advantage of this circuit is the use of an output buffer, which allows the circuit that is fed by the ADC to read the digital data while the ADC is already working on the next sample.

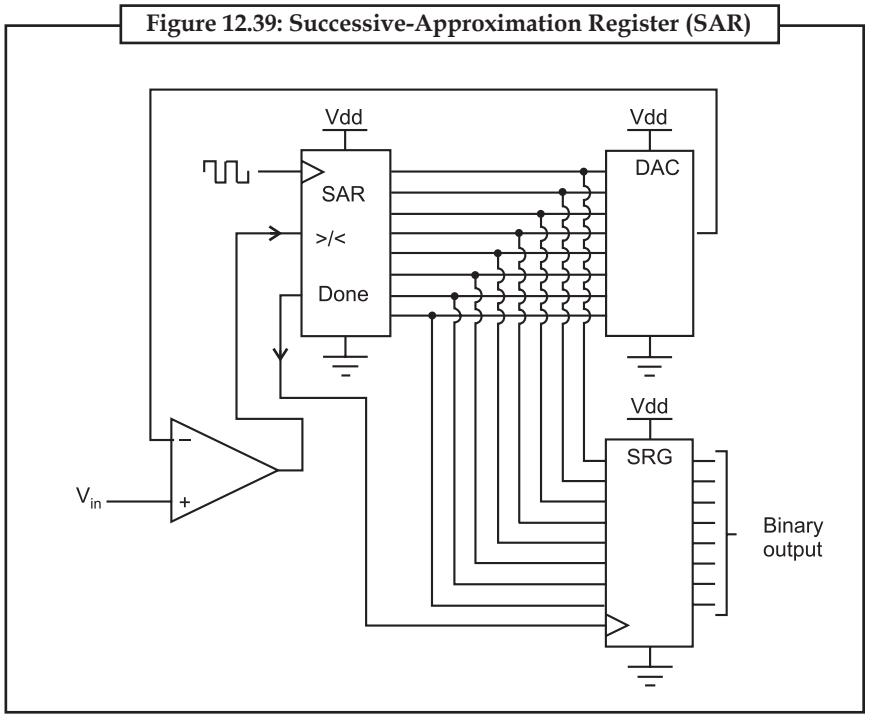
One method of addressing the digital ramp ADC's shortcomings is the so-called successive-approximation ADC. The only change in this design is a very special counter circuit known as a successive-approximation register. Instead of counting up in binary sequence, this register counts by trying all values of bits starting with the most-significant bit and finishing at the least-significant bit. Throughout the count process, the register monitors the comparator's output to see if the binary

Notes

count is less than or greater than the analog signal input, adjusting the bit values accordingly. The way the register counts is identical to the "trial-and-fit" method of decimal-to-binary conversion, whereby different values of bits are tried from MSB to LSB to get a binary number that equals the original decimal number. The advantage to this counting strategy is much faster results: the DAC output converges on the analog signal input in much larger steps than with the 0-to-full count sequence of a regular counter.

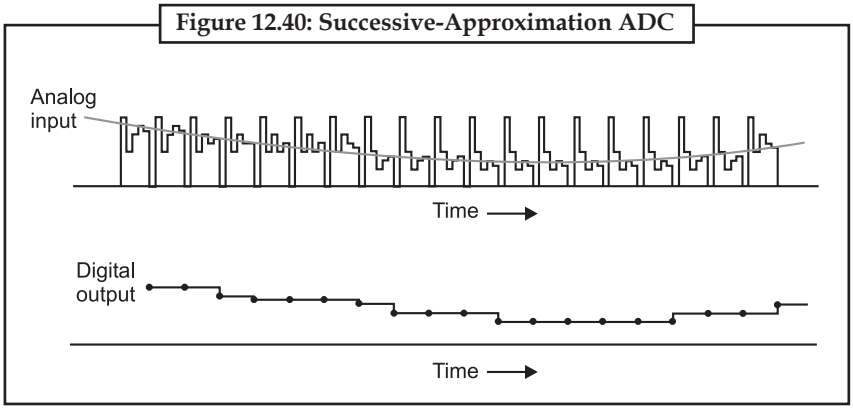
Without showing the inner workings of the successive-approximation register (SAR), the circuit looks like this:

Figure 12.39: Successive-Approximation Register (SAR)



It should be noted that the SAR is generally capable of outputting the binary number in serial (one bit at a time) format, thus eliminating the need for a shift register. Plotted over time, the operation of a successive-approximation ADC looks like this:

Figure 12.40: Successive-Approximation ADC



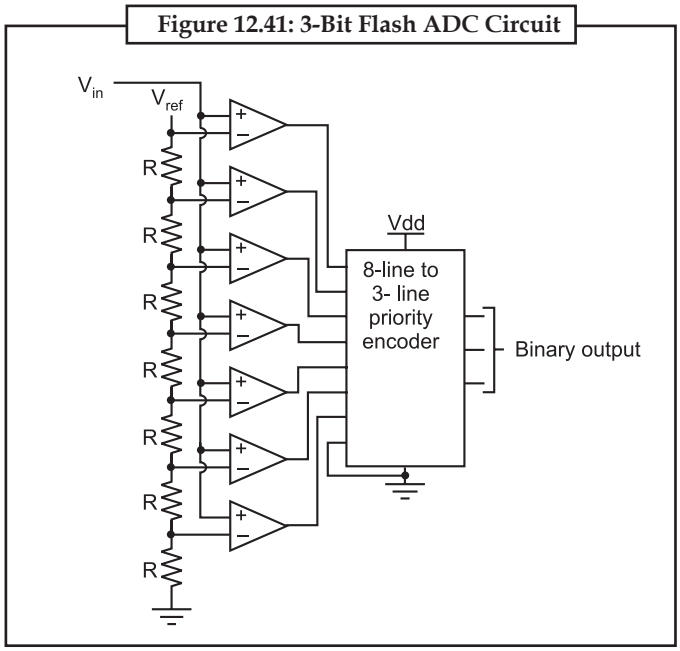
Note how the updates for this ADC occur at regular intervals, unlike the digital ramp ADC circuit.

Notes

12.13 Flash Converters

Also called the parallel A/D converter, this circuit is the simplest to understand. It is formed of a series of comparators, each one comparing the input signal to a unique reference voltage. The comparator outputs connect to the inputs of a priority encoder circuit, which then produces a binary output. The following illustration shows a 3-bit flash ADC circuit:

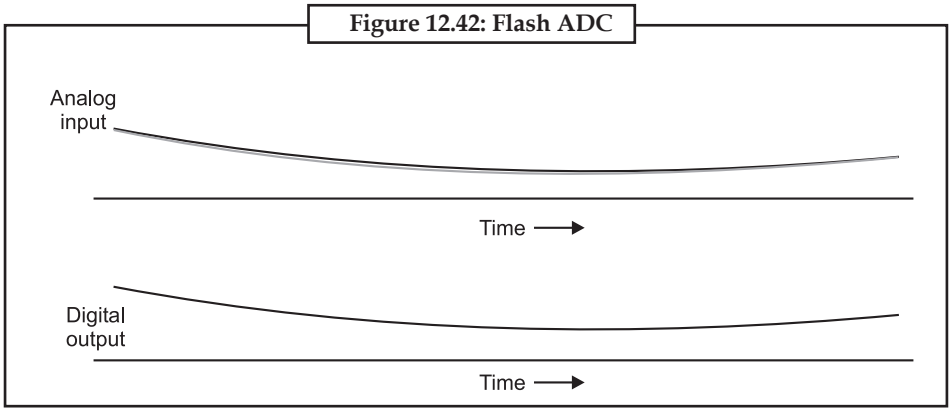
Figure 12.41: 3-Bit Flash ADC Circuit



V_{ref} is a stable reference voltage provided by a precision voltage regulator as part of the converter circuit, not shown in the schematic. As the analog input voltage exceeds the reference voltage at each comparator, the comparator outputs will sequentially saturate to a high state. The priority encoder generates a binary number based on the highest-order active input, ignoring all other active inputs.

When operated, the flash ADC produces an output that looks something like this:

Figure 12.42: Flash ADC

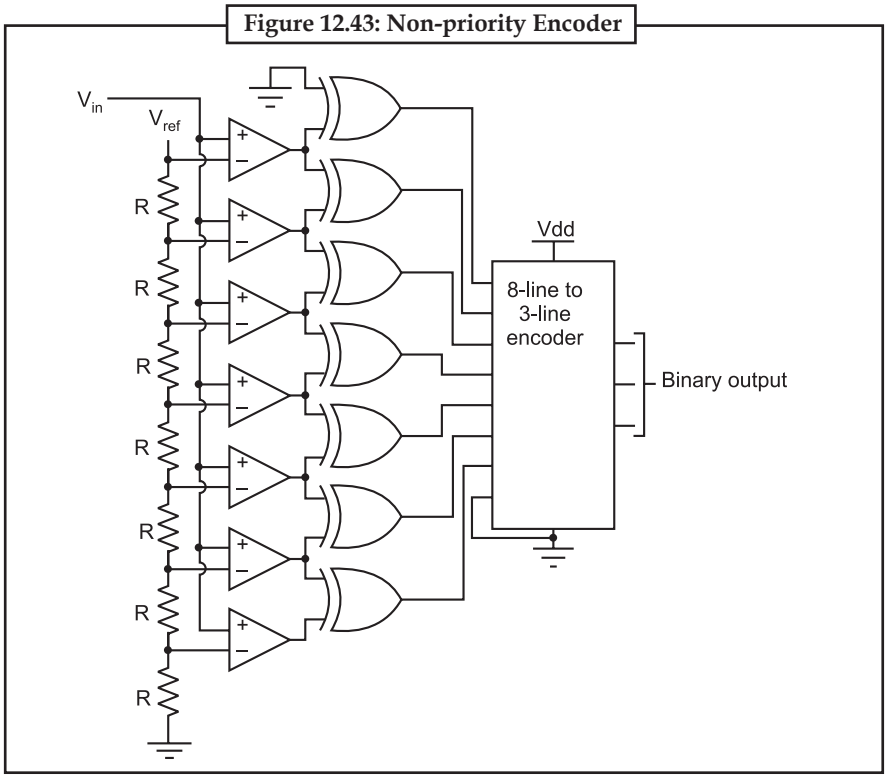


For this particular application, a regular priority encoder with all its inherent complexity is not necessary. Due to the nature of the sequential comparator output states (each comparator

Notes

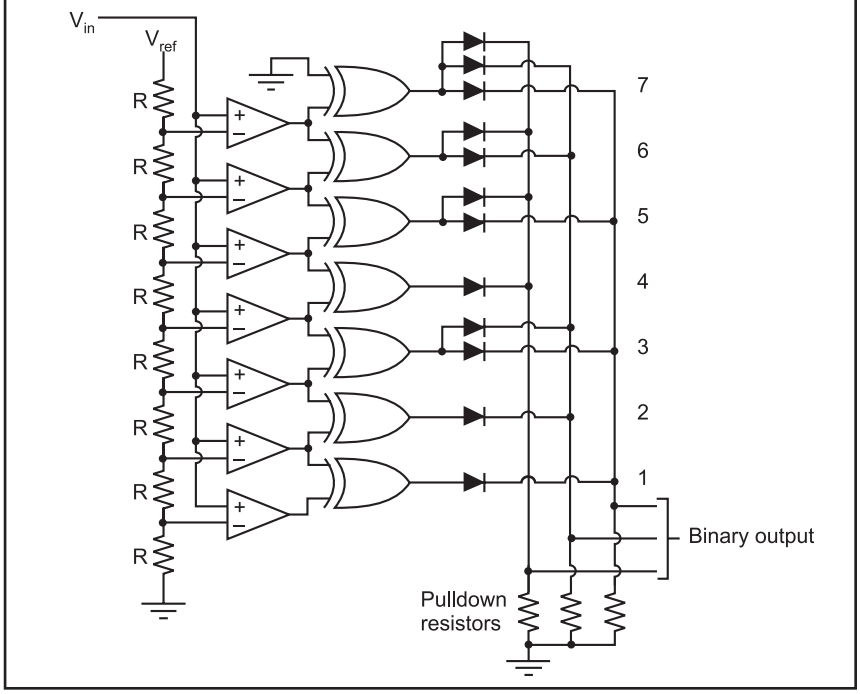
saturating "high" in sequence from lowest to highest), the same "highest-order-input selection" effect may be realized through a set of Exclusive-OR gates, allowing the use of a simpler, non-priority encoder:

Figure 12.43: Non-priority Encoder



And, of course, the encoder circuit itself can be made from a matrix of diodes, demonstrating just how simply this converter design may be constructed:

Figure 12.44: Matrix of Diodes



Notes

Not only is the flash converter the simplest in terms of operational theory, but it is the most efficient of the ADC technologies in terms of speed, being limited only in comparator and gate propagation delays. Unfortunately, it is the most component-intensive for any given number of output bits. This three-bit flash ADC requires seven comparators. A four-bit version would require 15 comparators. With each additional output bit, the number of required comparators doubles. Considering that eight bits is generally considered the minimum necessary for any practical ADC (255 comparators needed!), the flash methodology quickly shows its weakness.

An additional advantage of the flash converter, often overlooked, is the ability for it to produce a non-linear output. With equal-value resistors in the reference voltage divider network, each successive binary count represents the same amount of analog signal increase, providing a proportional response. For special applications, however, the resistor values in the divider network may be made non-equal. This gives the ADC a custom, nonlinear response to the analog input signal. No other ADC design is able to grant this signal-conditioning behaviour with just a few component value changes.

Case Study

TRX Systems

Global Positioning System (GPS) technology has had a transformative impact on our navigation and tracking capabilities, but there are many environments – indoors, underground, urban centres – where GPS is unavailable or degraded. In these environments, teams responding to emergency situations cannot rely on GPS location and status monitoring to help ensure the safety and security of first responders.

TRX Systems' Sentrix® Tracking System overcomes this challenge, enabling 3D infrastructure-free personnel tracking in office buildings, warehouses, caves, mines, 'urban canyons' and other GPS-denied environments. Utilizing advanced sensor and map fusion technology, motion classification algorithms, and data analysis engines, the Sentrix Tracking System affords first responders the ability to navigate unfamiliar terrain and track personnel with motion-capture precision in even the harshest conditions.

TRX Sentrix Command Station

Within each body-mounted Sentrix tracking unit, an ADI iSensor® ADIS16365 inertial sensor provides the high-performance gyroscope and accelerometer sensing functionality required to compute user motion and orientation in real-time. ADIS16365 inertial sensors utilize ADI's advanced iMEMS® technology to enable dynamic, multiaxis sensing precision in conjunction with TRX's proprietary Inertial Engine algorithms, providing the ability to track a user's movement and posture, and classify user motions such as walking, running, side stepping and crawling.

Contd...

Notes

Sentrix tracking units are typically worn under users' uniforms, and are ruggedized to withstand high body heat and environmental temperature. Where the performance of competing sensors is compromised under these conditions, ADI's ADIS16365 inertial sensors are designed to sustain consistent operation in temperatures up to 105 degrees.

ADI's integration of multiple axis of accelerometer and gyroscope in a single package enabled TRX's design team to conserve valuable board space and achieve a highly compact form factor so as not to impede users' agility in the field. ADIS16365 inertial sensors also help to minimize power consumption, ensuring that users can work a full 8+ hour shift between systems re-charges.

Questions:

1. Explain the working of Sentrix tracking system.
2. What do you understand by Global Positioning System (GPS) technology?

Self Assessment

Choose the correct answer:

6. Process of converting an analog voltage into an equivalent digital signal is known as.....
   (*a*) analog-to-digital (A/D) (*b*) digital-to-analog (D/A)
   (*c*) Both (*d*) None of these
7. The simultaneous method of A/D conversion is based on the use of a number of .....
   (*a*) single circuit (*b*) complete circuit
   (*c*) integrator circuit (*d*) comparator circuit
8. If you pay close attention, you will see that it is very similar to a ramp counter ADC.
   (*a*) True (*b*) False
9. Flash converter is called the .....
   (*a*) single-slope (*b*) parallel A/D converter
   (*c*) A/D converter (*d*) dual-slope
10. Signals in the real world are .....
    (*a*) serial (*b*) parallel
    (*c*) digital (*d*) analog

12.14 Summary

- Resistive divider that performs these functions is resistors R0, R1 and R2 form the divider network resistance RL represents the load to which the dividers connected and is considered to be large enough that it does not load the divider network.
- The total resistance looking out toward the 20 input is also 2R; these two resistors can be combined to form an equivalent resistor of value R.
- The steady-state accuracy test involves setting a known digital number in the input register, measuring the analog output with an accurate meter, and comparing with the theoretical value
- D/A converter is primarily a function of the accuracy of the precision resistors used in the ladder and the precision of the reference voltage supply used. Accuracy is a measure of how close the actual output voltage is to the theoretical output value.

Notes

- Higher-resolution A/D converter using only one comparator could be constructed if a variable reference voltage were available.

12.15 Keywords

***Accuracy test*:** Converter output voltage when the switch is opened, the capacitor holds the voltage level until the next sampling time the operational amplifier provides a large input impedance.

***ADC0804*:** It is an inexpensive and very popular A/D converter which is available from a number of different manufacturers, including National Semiconductor.

***Analog to Digital converter* (*A/D*):** It is used to change the analog output signals from transducers (measuring temperature, pressure, vibration, etc.) into equivalent digital signals.

***Digital-to-analog conversion* (*D/A*):** It is a straightforward process and is considerably easier than A/D conversion. In fact, a D/A converter is usually an integral part of any A/D converter.

***Dual-Slope A/D converter*:** It is a popular design based on this one is called dual-slope ADC, which solves an inherent single-slop problem called calibration drift, which leads to inaccuracy over time because the integrator is not linked to the clock signal.

***Flash converter*:** It remembers that three comparators were necessary for defining four ranges in general, it can be said that $2^n - 1$ comparators are required to convert to a digital signal that has n bits.

***Free-running mode*:** All the logic blocks inside the dashed line or some equivalent arrangement are frequently constructed on a single MSI chip; this chip is called a successive approximation register (SAR).

***Ladder*:** It is a resistive network whose output voltage is a properly weighted sum of the digital inputs. Such a ladder, designed for 4 bits, is constructed of resistors that have only two values and thus overcomes one of the objections to the resistive divider previously discussed.

***Monotonicity test*:** This is so as not to discharge the capacitor appreciably and at the same time offers gain to drive external circuits.

***Steady-state*:** Sample-and-hold amplifier when the switch is closed, the capacitor charges to the D/A.

***Successive-approx Imation register*:** When dealing with conversion times this short, it is usually necessary to take into account the other delays in the system.

Lab Exercise

1. Draw a truth table for successive approximation converter.

2. Write the Millman's theorem and solve it.

12.16 Review Questions

1. Define variables-resistor networks.
2. Describe binary ladders. Draw the circuit diagram.
3. Describe in brief D/A converters.
4. What are A/D converters and techniques? Explain in brief.
5. Explain dual-slope A/D conversion. Draw the circuit diagram.
6. Describe A/D accuracy and resolution.

Notes

7. Write a short notes on:
   - Section counters
   - ADC0804
   - Successive approximation
   - Single-ramp A/D converter
8. Describe the following:
   - Dual-slope A/D converter
   - Ladder
   - Free-running mode
   - Accuracy test
9. Explain in brief single-slope A/D converter.
10. What is dual-slope A/D converter? Explain.

Answers to Self Assessment

1. (*b*) 2. (*a*) 3. (*a*) 4. (*a*) 5. (*a*)
6. (*a*) 7. (*d*) 8. (*a*) 9. (*b*) 10. (*d*)

12.17 Further Readings

Books

Smart AD and DA Conversion, by Pieter Harpe, Arthur H. M. Roermund, Hans Hegt, Arthur Van Roermund.

PC Recording Studios For Dummies, by Jeff Strong.

Online link

http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=1312

LOVELY PROFESSIONAL UNIVERSITY
Jalandhar-Delhi G.T. Road (NH-1)
Phagwara, Punjab (India)-144411
For Enquiry: +91-1824-521360
Fax.: +91-1824-506111
Email: odl@lpu.co.in