

Rita Lovassy

Digital Technics

Kandó Kálmán Faculty of Electrical Engineering
Óbuda University
Budapest, 2013

Preface

Digital circuits address the growing need for computer networking communications in the technological world today. Starting from the development of the integrated circuit in 1959 there has been a continuous interest in the understanding of new digital devices capable of performing complex functions.

It is the intent of this book to give an overview of the basic concepts and applications of digital technics, from Boolean algebra to microprocessors. The book highlights the distinction between combinational circuits and sequential circuits, deals with numeral systems, and gives a clear overview of main digital circuits, starting from gates through latches and flip-flops.

In the case of combinational circuits, further distinction between logic circuits and arithmetic circuits is provided. Furthermore, in the last two chapters, the book develops, in detail, the main memory structures, gives a basic microcomputer organization, and introduces a typical 8-bit microprocessor.

The material in this book is suitable for one or two semesters' course of the bachelors' degree program in electrical engineering or is also well-fitted for self study. The aim is to acquaint the future engineers with the fundamentals of digital technics, digital circuits, and with their characteristics and applications. The book includes, as additional features, comprehensive examples and figures. At the end of each chapter a series of problems are given, which intend to give a broader view of the applicability of the concepts.

This book could be readily used in a completely new approach, as follows:

1. Number systems (Chapter 2)
2. Combinational logic networks (subsection 1.5) with various Boolean logic gates followed by switches and implementation of Boolean logic gates using transistors (subsection 3.1. Then the topics related to Boolean algebra and combinational logic optimization, minimization (subsections 1.1-1.4)
3. Sequential logic networks (subsection 3.2) with synchronous and asynchronous circuits
4. Microprocessor basics (Chapter 5) including memory structures (Chapter 4)

Contents

1	Fundamental Principles of Digital Logic	1
1.1	Boolean Algebra and its Relation to Digital Circuits	3
1.2	Truth Table and Basic Boolean Functions	7
1.3	Boolean Expressions	9
1.4	Minimization of Logic Functions	13
1.5	Combinational Logic Networks	23
2	Number Systems	30
2.1	Positional Number Systems	30
2.2	Binary Arithmetic	34
2.3	Signed Binary Numbers	38
2.4	Binary Codes and Decimal Arithmetic	40
2.5	Functional Blocks	45
3	Logic circuits and Components	56
3.1	Digital Electronic Circuits	56
3.2	Sequential Logic Networks	60
3.3	Flip-Flops	62
3.4	Registers and Counters	70

4	Semiconductor Memories and Their Properties	79
4.1	Volatile Memories	79
4.2	Nonvolatile Memories	83
4.3	Memory Expansion	86
5	Microprocessors Basics	89
5.1	Basic Microcomputer Organization	89
5.2	General Purpose Microprocessor	91
5.3	Instruction Sets	92
	References	95

Chapter 1

Fundamental Principles of Digital Logic

In general a **signal** can be defined as a value or a change in the value of a physical quantity. The signal represents, transmits or stores the information. The two main types of signal encountered in practice are analog and digital.

The **analog signal** is a continuous signal, which represents the information directly with its value. The time evolution of the analog signal can be represented by a continuous function. It changes continuously in time and it can cover fully a given range, see Figure 1.1. In practice the analog signal usually refers to electrical signals: especially voltage, but current, field strength, frequency etc. of the signal may also represent information.

For example, in sound recording, the voice (acoustic vibrations) is transformed by microphone (electro-acoustic transducer) into an electrical signal (voltage). Its characteristics are frequency range, signal-to-noise ratio, distortion, etc.

Analog circuits are designed for handling and processing analog signals and their input and output quantities are continuous. The advantages of such circuits are their ability to define infinite amounts of data, and they also use less bandwidth.

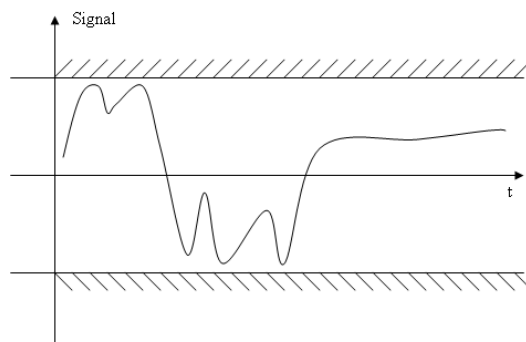


Figure 1.1 Analog signal

The analog signals have easy processing, which is made by analog circuits. The primary disadvantage of analog signals and analog circuits is their noise sensitivity. As the signal is transmitted from source to a distant destination the unwanted noise and disturbance introduced by each step in the signal path deteriorates the signal quality.

The **digital signal** contains the information in discrete symbols (e.g. numbers in coded form). It has discrete or quantized (the values of such a signal are restricted to belong to a finite set) values. The signal can be represented by integer numbers. One of the most common representations of a digital signal is the binary signal, which has a set of two elements: **0** and **1**.

The digital signal represents the information divided into elementary parts in a numeric form using appropriate encoding. Sampling is performed at given times, and the numbers are attached to it. The digital signal therefore represents coded information, see Figure 1. 2.

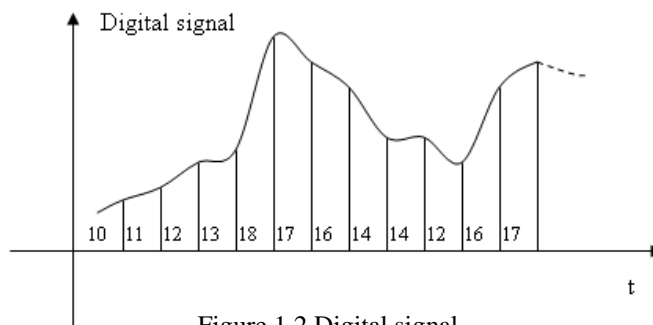


Figure 1.2 Digital signal

Digital systems manage discrete quantities of information; they are suitable for handling and processing digital signals. For example, a digital circuit is able to manipulate speech and music which are continuous (non-discrete) quantities of information. The signal is sampled at 8000 samples per second. Each sample is quantized and coded by a single byte. After these steps we have discrete quantity of information:

- The cost is 64 Kbit/s which is too much.
- Digital Signal Processing techniques allow us to bring this amount down to as low as 2.4 Kbit/s [1].

Digital systems are less expensive, with reliable operation, are easy to manipulate, are flexible, are immune to noise to a certain extent, etc.

Some disadvantages of digital circuits are the sampling of errors; digital communications require greater bandwidth than analog to transmit the same information.

Different data converters are the interfaces between analog devices and digital systems. In many applications it is need to convert an analog signal in a digital form suitable for processing by a digital system. An **analog-to-digital converter** (A/D) measures the analog signal at a certain rate and turns each sample into some bit values. The **digital-to-analog** (D/A) converters produce an analog output from a given digital input.

The next chapter introduces the basic principles of digital logic, and deals with the study of digital systems. The digital computer is the best known of such systems. Within a digital system the elementary units operate like switches, being either ON or OFF. The logic circuits can be built up from any basic unit that has two different states, one for the 1 input/output, and one for the 0 input/output. The complicated logic functions are the interconnection of a large number of switches called logic gates. The formal mathematical tool which can be used to describe the behaviour of logic networks is called Boolean algebra. In this chapter various types of Boolean algebra expressions will be introduced, and the description of logic connection and their implementation with various logic gates will be discussed.

1.1 Boolean Algebra and its Relation to Digital Circuits

The operation of almost all modern digital computers is based on two-valued or binary systems. Propositions may be TRUE or FALSE, and are stated as functions of other propositions which are connected by the three basic logical connectives: AND, OR, and NOT. [2].

Boolean algebra was introduced in 1854 by George Boole in his book: An Investigation of the Laws of Thought [3]. The connection between Boolean algebra and switching circuits was established by Claude Shannon [4]. He introduced the so called switching algebra as a main analytical tool to analyze and design logic circuits and networks. Typically, the units are in the form of switches that can be either ON or OFF (mapping to transistor-switches; high voltage means logic 1 and low voltage means logic 0).

The binary logic systems use the Boolean algebra, as a mathematical system, defined on a set of two-valued elements, in which the values of variable are 1 and 0. The binary variables are connected through logic operations. Special elements of the set are the unity (its value is always 1) and the zero (its value is always 0). The binary/logic variables are typically represented as letters: A,B,C,...,X,Y,Z or a,b,c,...,x,y,z.

Logic Variables

Logic variables are used to describe the occurrence of events. It can have two values i.e. TRUE or FALSE or YES/NO which refers to the occurrence of an event. Their meaning corresponds to the everyday meaning of the words in question. TRUE corresponds to logic-1 and FALSE corresponds to logic-0. Here 1 and 0 are not digits; they do not have any numeric value.

The levels represent binary integers or logic levels of 1 and 0. In active-high logic, HIGH represents binary 1, and LOW represents binary 0. The meaning of HIGH/LOW is connected with the usual electrical representation of logic values, they correspond to high(er) and low(er) potentials (voltage levels) e.g. (nominally) +5 V and 0 V, respectively.

Basic Boolean Operations

There are several Boolean operations. The most important are:

- AND (conjunction) – represented by operators “ \cdot ” or “ \wedge ”
- OR (disjunction) – represented by operators “ $+$ ” or “ \vee ”
- NOT (negation, inversion, complementation) - represented by operator “ \neg ” or denoted by overline (bar).

The AND and OR logic operations are two- or multi-variables, the logic negation is a one-variable operation.

The result of AND operation is TRUE if and only if both input operands are TRUE. In logic algebra the AND operation is also called binary/logic multiplication. The AND operation between two variables A and B is written as $A \cdot B$ or AB . The postulates for the AND operation are given in Table 1.1.

A	B	$A \cdot B$
0	0	0
0	1	0
1	0	0
1	1	1

Table 1.1 Definition of the AND operation

The result of OR operation is TRUE if any input operands are TRUE. In logic algebra the OR operation is also called binary/logic addition. The OR operation

between two variables A and B is written as A+B. The postulates for the OR operation are given in Table 1.2.

A	B	A+B
0	0	= 0
0	1	= 1
1	0	= 1
1	1	= 1

Table 1.2 Definition of the OR operation

Electrical implementation of AND and OR are series and parallel connection of switching elements (see Figure 1.3) like electromechanical relays or n-and p-channel FETs in CMOS circuitry (see Chapter 3).

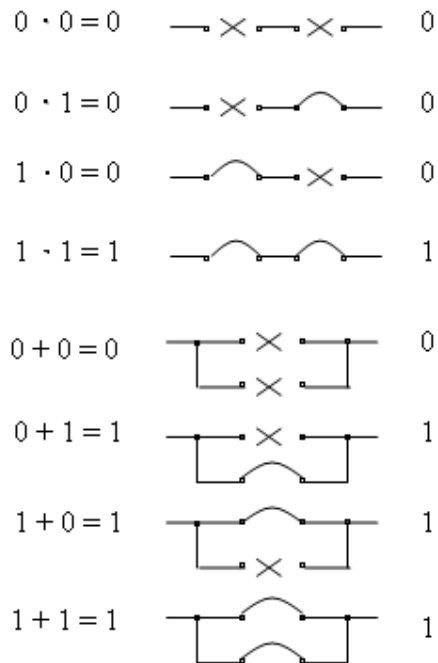


Figure 1.3 Electrical implementation of the AND and OR operations

The result of NOT operation is TRUE if the single input value is FALSE. In this case the complementation of A is written as \bar{A} .

If $A = 0$; $F = \bar{A} = 1$ and if $A = 1$; $F = \bar{A} = 0$

A	\bar{A}
0	1
1	0

Table 1.3 Definition of the NOT operation

Each element of the set has its complementary also belonging to the set. A two-valued Boolean algebra is defined as a mathematical system with the elements 0 and 1 and three operations, whose postulates are given in Tables 1.1 to 1.3.

Boolean Theorems

Basic identities of Boolean algebra are presented in pairs i.e. with both AND and OR operations.

Let A be a Boolean variable and 0, 1 constants

$$A + 0 = A; \text{ Zero Axiom;}$$

$$A + A = A; \text{ Idempotence}$$

$$A \cdot 1 = A; \text{ Unit Axiom}$$

$$A \cdot A = A; \text{ Idempotence}$$

$$A + 1 = 1; \text{ Unit Property}$$

$$A + \bar{A} = 1; \text{ Complement}$$

$$A \cdot 0 = 0; \text{ Zero Property}$$

$$A \cdot \bar{A} = 0; \text{ Complement}$$

$$\overline{\bar{A}} = A; \text{ Involution}$$

Let A, B and C Boolean variables

1. Commutativity: the order of operands can be reversed

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

$$A + B = B + A$$

2. Associativity: the operands can be regrouped

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

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

The order of operations is given by the parentheses.

3. Distributivity: the operands can be reordered

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

$A + (B \cdot C) = (A + B) \cdot (A + C)$
 Uniting theorem (absorption law)

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

These theorems are only valid in logic algebra, and they are not valid in the ordinary algebra! In the binary system is some kind of symmetry between the AND and OR operators which is called duality. Every equation has its dual pair which can be generate by replacing the AND operators with OR (and vice versa) and the constants 0 with 1s (and vice versa).

De Morgan's Laws

De Morgan's laws or theorems occupy an important place in Boolean algebra. De Morgan's theorems may be applied to the

- negation of a disjunction: $\overline{A + B} = \overline{A} \cdot \overline{B}$

Since two variables are false, it's also false that either of them is true.

- negation of a conjunction: $\overline{A \cdot B} = \overline{A} + \overline{B}$

Since it is false that two variables together are true, at least one of them should be false. The De Morgan's theorem is an important tool in the analysis and synthesis of digital and logic circuits. Its generalization to several variables is stated below:

$$\overline{A + B + C + \dots} = \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot \dots$$

$$\overline{A \cdot B \cdot C \cdot \dots} = \overline{A} + \overline{B} + \overline{C} + \dots$$

1.2 Truth Table and Basic Boolean Functions

In order to describe the behavior and structure of a logic network it is necessary to express its output F as a function of the input variables A, B, C,...

A Boolean function domain is a set of n-tuples of 0's and 1's, and the range is an element of the set {0, 1}. The values of the function are obtained by substituting logic-0 and logic-1 for the corresponding variables in the expression [4]. The **truth table** is a unique representation of a Boolean

function which shows the binary value of the function for all possible combinations of the independent variables. In case of N variables, the truth table has N + 1 columns, and 2^N rows, for all possible binary combinations for the variables. In general, a truth table consists of

- column for each input variable
- row for all possible input values
- column for resulting function value

For given N binary variable there exist 2^{2^N} different Boolean functions of these N variables.

One Variable Boolean Functions

In case of one variable, there exist four Boolean functions.

The names of these functions and the truth table (Table 1.4) are given below:

- $F_0^1 = 0$ function constant 0
- $F_1^1 = \overline{A}$ function inversion (NOT)
- $F_2^1 = A$ function identity
- $F_3^1 = 1$ function constant 1

A	F_0^1	F_1^1	F_2^1	F_3^1
0	0	1	0	1
1	0	0	1	1

Table 1.4 Truth table - one variable Boolean functions

Two Variable Boolean Functions

In the case of two variables the number of possible input combinations is $2^2 = 4$, therefore the number of possible two-variable functions is $2^4 = 16$. Each function describes a single or complex logic operation, see Table 1.5.

A	B	F_0^2	F_1^2	F_2^2	F_3^2	F_4^2	F_5^2	F_6^2	F_7^2	F_8^2	F_9^2	F_{10}^2	F_{11}^2	F_{12}^2	F_{13}^2	F_{14}^2	F_{15}^2
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Table 1.5 Truth table - two variables Boolean functions

$F_0^2 = 0$	function constant 0
$F_1^2 = A \cdot B$	function AND
$F_2^2 = A \cdot \overline{B}$	function inhibition
$F_3^2 = A$	function identity
$F_4^2 = \overline{A} \cdot B$	function inhibition
$F_5^2 = B$	function identity
$F_6^2 = \overline{A} \cdot B + A \cdot \overline{B} = A \oplus B$	function antivalency, exclusive-OR (XOR)
$F_7^2 = A + B$	function OR
$F_8^2 = \overline{A + B}$	function NOR
$F_9^2 = A \cdot B + \overline{A} \cdot \overline{B} = A \otimes B$	function equivalency, exclusive-NOR (XNOR)
$F_{10}^2 = \overline{B}$	function inversion
$F_{11}^2 = A + \overline{B}$	function implication
$F_{12}^2 = \overline{A}$	function inversion
$F_{13}^2 = \overline{A} + B$	function implication
$F_{14}^2 = \overline{A \cdot B}$	function NAND
$F_{15}^2 = 1$	function constant 1

A logic function can be specified in various ways:

1. Truth table
2. Boolean equation, algebraic form
3. Maps (see subsection 1.4)
4. Symbolic representation, logic gates (see subsection 1.5)

The conversion of one representation of a Boolean function into another is possible in a systematic way.

1.3 Boolean Expressions

Obtaining a Boolean expression from a truth table

In the next example a Boolean expression of three variables is obtained from a truth table, see Table 1.6.

The logic expression is a function (formula) consisting of Boolean constants and variables connected by AND, OR, and NOT operations. The expression is:

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

i	A (2 ²)	B (2 ¹)	C (2 ⁰)	F
0	0	0	0	0
1	0	0	1	1
2	0	1	0	1
3	0	1	1	0
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	0

Table 1.6 Truth table for the Boolean expression

Each term represents an input variable combination for which the function value is $F = 1$, consisting of all variables either in negated or in unnegated form.

Sum-of-Products (SOP) Form, the Minterm Canonical Form

The unique algebraic form readout from the truth table as AND connections of OR operations is called **Sum-of-Products (SOP) form, disjunctive canonical form, disjunctive normal form (DNF)** or **minterm canonical form**.

Minterm

The minterm is a term composed from the variables logic product, in which all the variables appear exactly once, either complemented or uncomplemented. The terms of the disjunctive canonic form are called minterm [5]. There are 2^N distinct minterms for N variables.

The generalized form is denoted by: m_i^N

where N is the number of independent variables, and i (minterm index) is the decimal value of the binary number corresponding to the given combination of the independent variables.

To illustrate the notation, consider the previous function expression

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

$$F = m_1^3 + m_2^3 + m_5^3 + m_6^3 = \sum^3(1, 2, 5, 6)$$

Product-of-Sums (POS) Form, Maxterm Canonical Form

The unique algebraic form readout from the truth table as OR connections of AND operations is called **Product-of-Sums (POS) form, conjunctive canonical form, conjunctive normal form (CNF) or maxterm canonical form.**

Maxterm

There is a dual entity called maxterm which is a product of sums expansion (conjunctive normal form). The maxterm is a term composed from the variables logic sum, in which all the variables appear exactly once, either complemented or uncomplemented. There are 2^N distinct maxterms for N variables [5].

The generalized form is denoted by: M_I^N

where N is the number of independent variables, and I (maxterm index) is the decimal value of the binary number corresponding to the given combination of the independent variables.

To find the conjunctive canonic form, we consider the negated function from Table 1.6 (see rows nr. 0, 3, 4, 7)

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

$$\overline{F} = m_0^3 + m_3^3 + m_4^3 + m_7^3$$

Based on De Morgan's law the conjunctive canonical form of function F can be obtained from the negated function expression by appropriate transformations resulting in a product-of-sums (POS) i.e. in a product of maxterms. The complemented function consists of those minterms, where the function value is $F = 0$.

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

$$F = M_7^3 \cdot M_4^3 \cdot M_3^3 \cdot M_0^3 = \prod^3(7, 4, 3, 0)$$

Minterm to Maxterm Conversion

Let's start from the original function disjunctive normal form

$$F = m_1^3 + m_2^3 + m_5^3 + m_6^3$$

The expression of the negated function, also in disjunctive form (index i) is

$$\bar{F} = m_0^3 + m_3^3 + m_4^3 + m_7^3$$

The function expression in conjunctive normal form, (index I = 2³ - 1 - i) is

$$F = M_7^3 \cdot M_4^3 \cdot M_3^3 \cdot M_0^3$$

The relationship between the minterm indexes **i** of the complemented function and the maxterm **I** of the uncomplemented function (written for the case of three variables) is $i + I = 7 = 2^3 - 1$

In general, we can write for a function with N variables

$$i + I = 2^N - 1$$

We can state that: all minterm is the complement of a maxterm and vice versa.

$$\overline{m_i^n} = M_{2^N-1-i}^n \quad \text{and} \quad \overline{M_i^n} = m_{2^N-1-i}^n$$

The sum of all the minterms of an N variable function is 1, the product of all the maxterms is 0.

$$\sum_{i=0}^{2^N-1} m_i^n = 1 \quad \text{and} \quad \prod_{i=0}^{2^N-1} M_{2^N-1-i}^n = 0$$

1.4 Minimization of Logic Functions

The logic functions are used to design digital logic circuits. The aim is to find an economic, small, fast and cheap implementation of the specified logic network. In many cases the optimization, simplification of the network means to reduce the number of electronic components, the number of gates level, the number of inputs-, interconnections etc. Since the expressions resulting from simplification are equivalent, the logic networks that they describe will be the same.

Boolean function simplification methods are:

- Algebraic minimization, using Boolean algebraic transformations
- Graphic minimization, using the Karnaugh-map
- Numeric (tabular) minimization, using Quine-McCluskey- method.
- Heuristic algorithms, (e.g. algorithms like ESPRESSO).

In the next the first two methods will be discussed in detail.

Algebraic minimization using Boolean theorems

The laws and identities of Boolean algebra allow us to simplify a minterm expression. A significant simplification of minterms yields to an equivalent new function expression with fewer Boolean operators and variables. Unfortunately, with this procedure it could be difficult to find the „simplest” expression because the Boolean expressions are not algorithmic. Hence, it is not always obvious which theorem to apply at each step. During algebraic minimization, systematically, the basic properties and theorems of Boolean algebra had to be applied. In this way, step by step the adjacent minterms (differing by ONLY one variable, which appears complemented in one term and uncomplemented in the other) are contacted, and the corresponding variables are eliminated.

For example, to simplify the function

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

we can proceed as follows:

$$\begin{aligned} F &= \overline{A}\overline{B}C + \overline{A}B\overline{C} + A\overline{B}\overline{C} + ABC = \overline{B}C(\overline{A} + A) + B\overline{C}(\overline{A} + A) \\ &= \overline{B}C + B\overline{C} \end{aligned}$$

A similar approach can be applied to the conjunctive form, adjacent maxterms are contacted, and the corresponding variables are eliminated. For example, to simplify the function

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

we can proceed as follows:

$$\begin{aligned} F &= (A + B + \bar{C})(A + \bar{B} + \bar{C}) = ((A + \bar{C}) + B)((A + \bar{C}) + \bar{B}) \\ &= (A + \bar{C}) + (B + \bar{B})(A + \bar{C}) + B\bar{B} \\ &= A + \bar{C} \end{aligned}$$

Graphic minimization, Karnaugh map (K-map)

Karnaugh maps were invented by Maurice Karnaugh, a telecommunications engineer. He developed them at Bell Labs in 1953 while studying the application of digital logic to the design of telephone circuits. This method is typically used on Boolean functions of two, three or four variables - past that, other techniques are frequently used. [4].

The Karnaugh map, known also as Veitch diagram, is a unique graphic representation of Boolean functions which provides a technique for the logic equation minimalization. The array of cells contains the truth table information. Mapping can be applied both to minterms and maxterms as well. The K-map of a Boolean function of N variables consists of 2^N cells and is built up from adjacent cells having terms which differ only in one bit (place) [6]. Adjacent terms are where only **one** logic variable appears in complemented and uncomplemented form, while all others remain the same. For example:

(011) and (010), also (000) and (100)

This arrangement allows a quick and easy simplification keeping some simple rules. In the case of 5 or more variables the adjacent cells scheme becomes much more complex.

Minimal Sums

One method of obtaining a Boolean expression from a K-map is to select only those minterms of the normal expression that have a logic-1 value.

Two Variable Karnaugh Maps

The two-variable K-map contains four cells, covering all possible combinations of the two variables as is shown in Figure 1.4. Each row of the truth table corresponds to exactly one cell. If the truth table row is **one** the respective cell contains a **one**. Usually the zeros are not indicated, and an empty cell is considered to contain a **zero**. Figure 1.4 a presents also the numbering of K-maps cells. Figure 1.4 c shows a two variable logic function truth table and the corresponding K-map. The logic function would be 1 if **A = 0 AND B = 1 OR A = 1 AND B = 1**.

If the row or column of the map in which the **1** appears is labeled by a **1**, the variable appears uncomplemented, otherwise the variable appears complemented. Read through the K-map cells content, the function expression is: $F = \overline{A}B + AB$

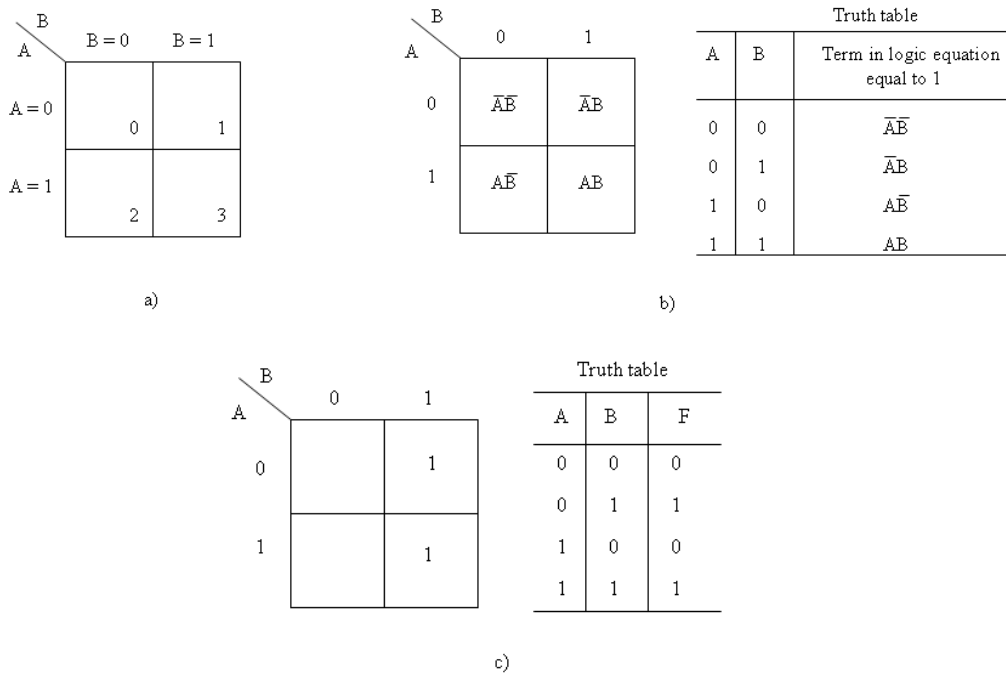


Figure 1.4 Karnaugh maps: a) two-variable map; b) correspondence with truth table; c) example

Using the algebraic minimization, the function can be rewritten as:

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

The K-map advantage is that the adjacent cells (which differ by ONLY one variable, appear complemented in one cell and uncomplemented in the other) can be grouped visually to eliminate redundant variables. Thus, grouping of the two cells, see Figure 1.5, immediately we obtain the simplified function form.

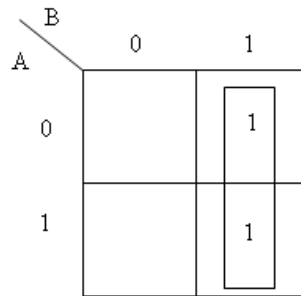


Figure 1.5 Karnaugh map; $F = B$

Three - Variable Karnaugh Maps

The K-maps edges are headed using a one-step (Gray) code.

The **Gray code** is a series of 2^N code words, each of N-bits, in such a sequence that any adjacent code words differ only in one bit, including the first and last words too (cyclic property).

For example, in case of $N = 2$ the sequence of code words is:

(00), (01), (11), (01)

The **Hamming distance** of two code words of equal length is the number of positions at which the corresponding codes are different. For example, 10110 and 01101 differ in 4 positions, the distance between them is 4. The Hamming distance between any two adjacent code words of the Gray code is one (see Table 2.4).

When K-maps involve three variables; the cells represent the minterms of all variables, as is shown in Figure 1.6 a. The numbering of K-maps cells is presented in Figure 1.6 b. In Figure 1.6 c those columns and row are signed where the corresponding variables have logic-1 value.

Top cells are adjacent to bottom cells. Left-edge cells are adjacent to right-edge cells. Rows and columns on the opposite sides are also adjacent.

In the process of contraction and minimization the following steps and rules are necessary:

- introduce **ones** in each cell of the K-map for which the corresponding minterm in the function is equal to **one**.

- group adjacent cells which contain **ones**. The number of cells in a group must be a power of 2.
- the process is continued until no more variables can be eliminated by further contractions. The groups cover all cells containing **ones**.

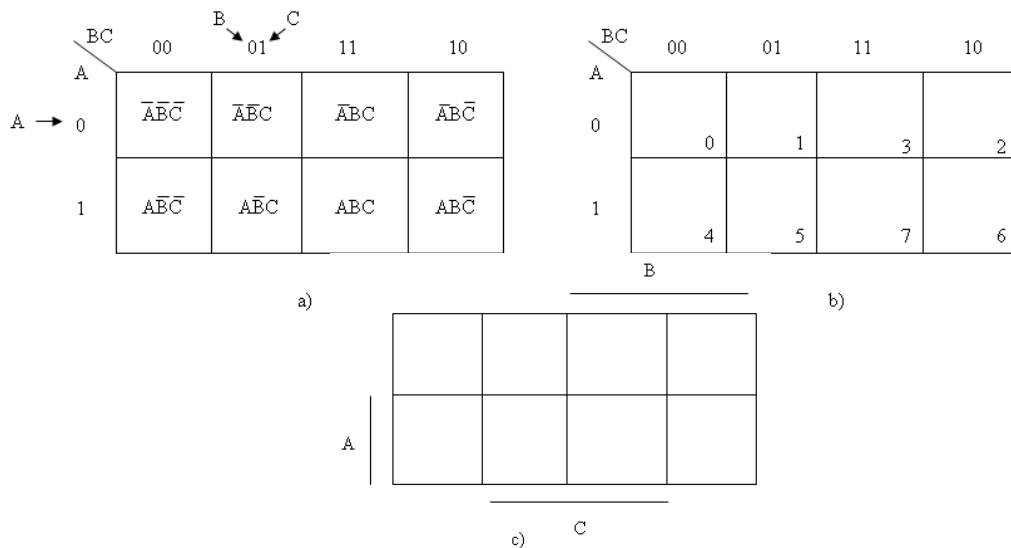


Figure 1.6 Three-variable Karnaugh maps

In the case of three variables K-map:

- If a group of four adjacent cells (in-line or square) is contracted, the result yields in a single variable.
- If a group of two adjacent cells is contracted, the result yields in a two-variable product term.
- A single cell which cannot be combined represents a three-variable term.

Example 1.1 Use the K-Map to simplify the following expression:

a) $F = \overline{A}\overline{B}\overline{C} + \overline{A}\overline{B}C + \overline{A}BC + A\overline{B}\overline{C}$; (Truth table, see Table 1.6)

The map is shown in Figure 1.7, and the indicated grouping leads to the simplified expression

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

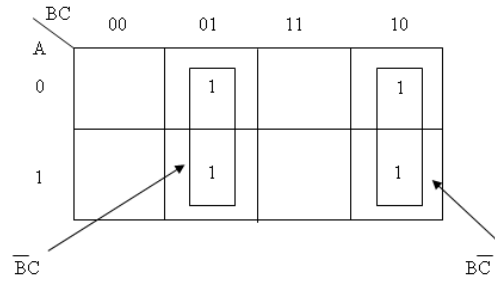


Figure 1.7 Karnaugh map for Example 1.1 a

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

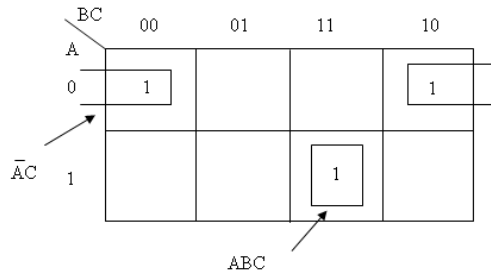
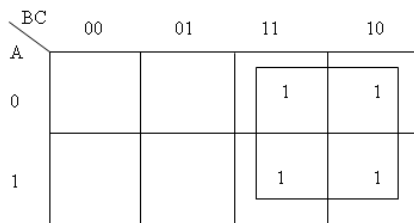


Figure 1.8 Karnaugh map for Example 1.1 b

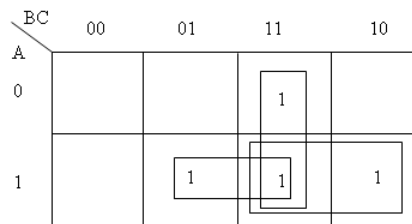
Figure 1.8 shows a split rectangular grouping and a single cell which cannot be combined (represents a three-variable term). The simplified expression is:

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

In Figure 1.9 the optimal grouping of 1-cells are shown. The minimal sums for K maps are given.



$$F = B$$



$$F = AC + BC + AB$$

Figure 1.9 Grouping on three-variable Karnaugh maps

Four-Variable Karnaugh Maps

A four-variable map has 16 cells, as shown in Figure 1.10. The grouping rules are the same as for three variables K map. The goal is to find the smallest number of the largest possible sub cubes that cover the 1-cells.

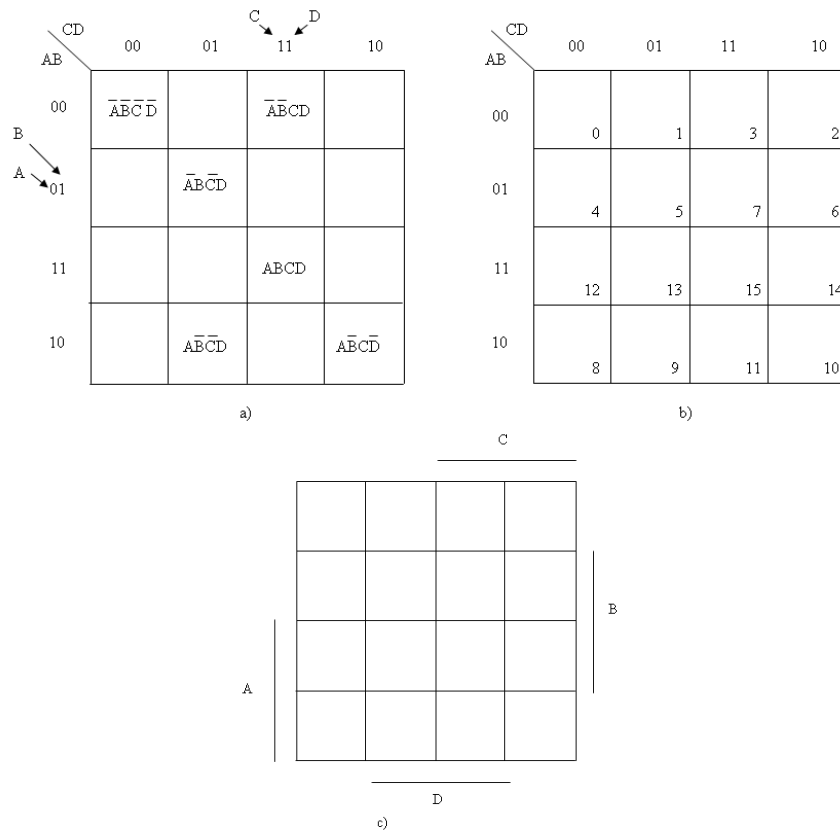


Figure 1.10 Four-variable Karnaugh maps

Example 1.2 Use the K-map to simplify the logic function given by the minterms

$$F = \sum (0, 2, 5, 8, 9, 10, 11, 12, 13, 14, 15)$$

The corresponding Karnaugh map is given in Figure 1.11.

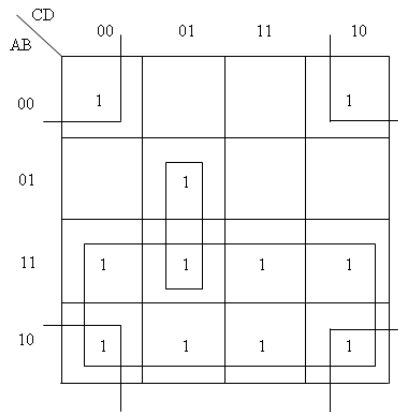


Figure 1.11 Karnaugh map for Example 1.2

The simplified expression is

$$F = A + \overline{BCD} + \overline{BD}$$

The obtained terms are called the function **prime implicants**. If a minterm of a function is included in only one prime implicant, then this prime implicant is an **essential prime implicant** of the function [7].

In this way the K-maps permit the rapid identification and elimination of potential race hazards. The simplification result may not be unique.

Example 1.3 Simplify the Karnaugh-map shown in Figure 1.12.

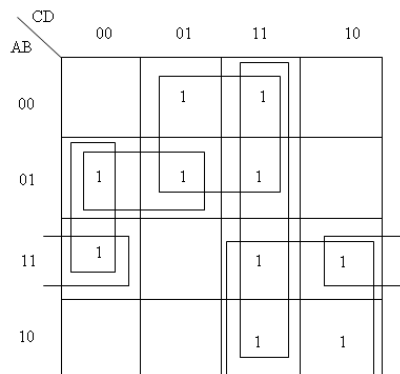


Figure 1.12 Karnaugh map for Example 1.3

The simplification result consists of 6 prime implicants:

$$F = \overline{A}D + CD + AC + AB\overline{D} + \overline{B}C\overline{D} + \overline{A}B\overline{C}$$

Essential prime implicants for minimum cover

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

Karnaugh Map for Minimalization of a Four Variables Logic Function Expressed in Product-Of-Sums

Sometimes the Product-Of-Sums form of a function is simpler than the Sum-of-Product form. In a very similar way as in the previous subsection (Four-Variable Karnaugh Maps), two adjacent maxterms can be contracted to a single sum. The involution identity is applied for the logic function: $\overline{\overline{F}} = F$. In the minterm K-map the simplifications are made using zeros, obtaining the minimalization of \overline{F} .

Example 1.4 K-map method: contracting zeros (second “optimum” solution for the function given in Example 1.2)

Use the - map to simplify the logic function given by the minterms

$$F = \sum_{i=0}^4 (0, 2, 5, 8 - 15)$$

The corresponding Karnaugh-map is given in Figure1.13.

In the next the following rule is used: Replace F by \overline{F} , and 0's become 1's and vice versa.

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

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

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

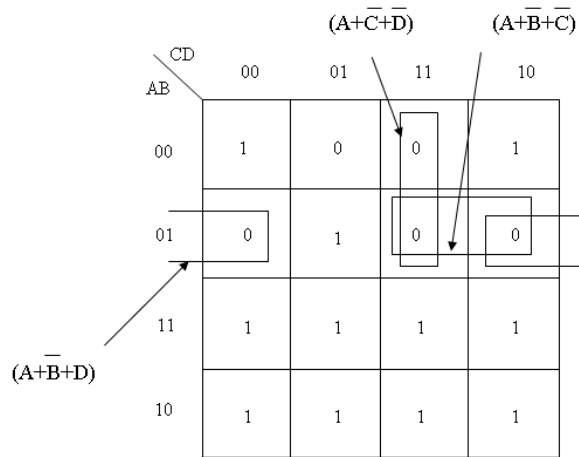


Figure 1.13 Karnaugh map for Example 1.4

Incompletely Specified Logic Function

In the expression of incompletely specified logic function are such input combinations to which the Boolean function is not specified. The function value for these combinations is called **don't care** and the combination is called don't care condition.

In an implementation the don't care terms value may be arbitrarily 0 or 1. The selection point of view is if they are able to generate prime implicants in order to obtain the most advantageous solution.

Don't care conditions are indicated on the K- maps by dash entries. Figure 1.14 shows a Karnaugh map involving don't care conditions.

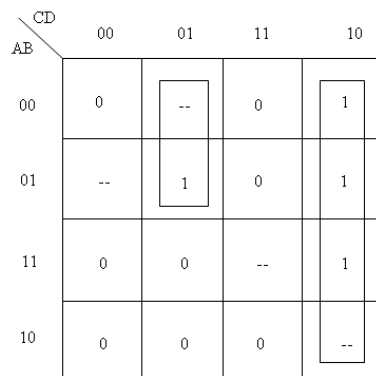


Figure 1.14 Karnaugh map with don't care conditions

The map of Figure 1.14 can be used to obtain a minimal sum:

$$F = \overline{\overline{A}CD} + C\overline{D}$$

1.5 Combinational Logic Networks

Logic networks are implemented with digital circuits, and in reverse, digital circuits can be described and modeled with logic networks. For the analysis and synthesis of logic network the Boolean algebra is used.

The logic network (logic circuit) processes the actual values of the input variables (A, B, C, ...) and produces accordingly the output logic functions (F₁, F₂, ...).

Logic networks described by truth tables or Boolean expressions can be classified into two main groups:

1. Combinational networks: the output values depend only on the present input variable values;
2. Sequential networks: the output values depend on both the inputs to the operation and the result of the previous operation. Networks having the memory property will be studied in subsection 3.2.

The combinational logic network is the simplest logic network. The logic operations on the input variables are performed "instantaneously" and the result will be available on the output at the same time, (except for the time delay due to the internal operation of the circuits). The output variables can be represented as logic functions of the input variables. Figure 1.15 shows a combinational logic network as a black box.

$$F_i = f(A, B, C, \dots, N) \quad i = 1, 2, \dots, M$$

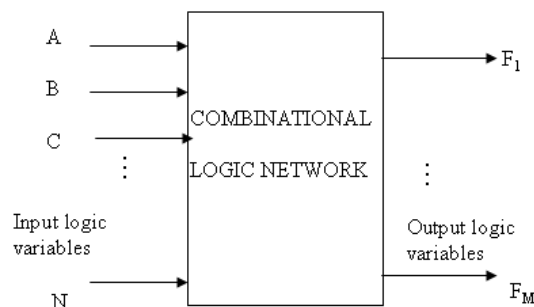


Figure 1.15 Combinational logic network

The combinational circuit maps an input (signal) combination to an output (signal) combination. The same input combination always implies the same output combination (except transients). The reverse is not true. For a given output combination different input combinations can belong.

Combinational circuit examples: binary arithmetic circuits (half-adder, full-adder, etc.) (see subsection 2.2), binary-coded-decimal code (BCD) – seven segment display (see subsection 2.4), various encoders and decoders, multiplexers and demultiplexers, comparators (see subsection 2.5), etc.

The combinational circuits are the interconnections of a large number of switches called logic gates.

Logic Gates

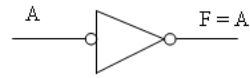
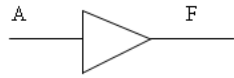
Logic gates as electronic circuit components are elementary building blocks of logic circuits which implement basic Boolean functions of one or more variables.

Figure 1.16 lists the symbols of the main logic gates with two inputs. The NOT gate implements the function inversion, having one input. A simple triangle symbol denotes a buffer amplifier, which serves the IDENTITY function. The small circle on the input or output of a gate means the NOT operation. Based on De Morgan's identities:

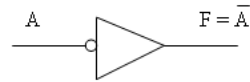
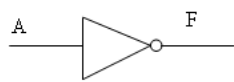
$$F = AB = \overline{\overline{A + B}} \quad \text{and} \quad F = A + B = \overline{\overline{A} \overline{B}}$$

While the NOT, AND, OR functions have been designed as individual circuits in many circuit families, by far the most common functions realized as individual circuits are the AND-NOT (NAND) and OR-NOT (NOR) circuits. A NAND can be described as equivalent to an AND element driving a NOT element. Similarly, a NOR is equivalent to an OR element driving a NOT element. The reason for this strong bias favoring inverting outputs is that the transistors which preceded it are by nature inverters or NOT-type devices when used as signal amplifiers.

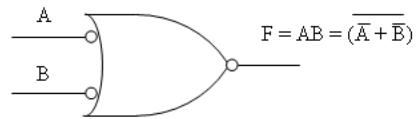
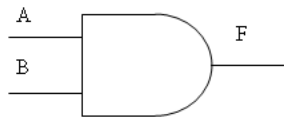
IDENTITY



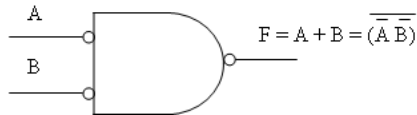
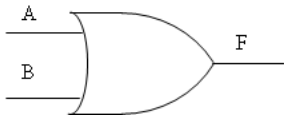
NOT



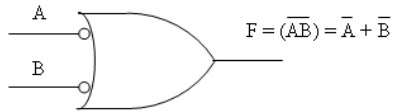
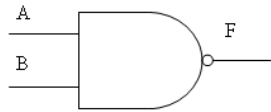
AND



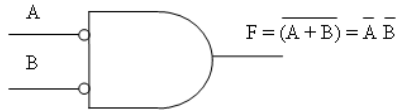
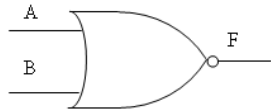
OR



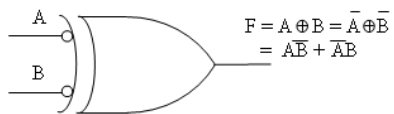
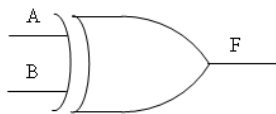
NAND



NOR



EXCLUSIVE OR



NOT EXCLUSIVE OR (EQUIVALENCE)

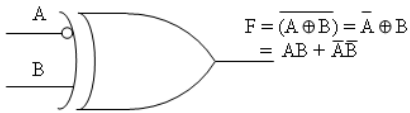
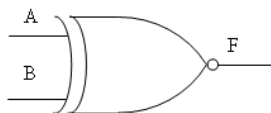


Figure 1.16 Main gate symbols

The EXCLUSIVE OR and NOT EXCLUSIVE OR Functions

The final two gates symbols introduced in Figure 1.16 are the EXCLUSIVE OR gate and the NOT EXCLUSIVE OR (Equality) gate. The EXCLUSIVE OR, (XOR) called the modulo-2-sum or “antivalency” operation is denoted by the symbol \oplus . The EXCLUSIVE OR function forms are:

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

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

By definition the value of $A \oplus B$ is logic-1 if and only if A and B variables have different values. The complement of the EXCLUSIVE OR operation is the NOT EXCLUSIVE OR (called EXCLUSIVE NOR, XNOR or “equivalency”) operation. Their expressions are:

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

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

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

By definition the value of $\overline{A \oplus B}$ is logic-1 if and only if A and B variables have the same values. The EXCLUSIVE OR and NOT EXCLUSIVE OR gates are typically available with only two inputs.

The commercial gates (exception NOT) are often designed for multiple inputs. Generalized symbol shown in Figure 1.17 is frequently used when a single gate has several inputs.

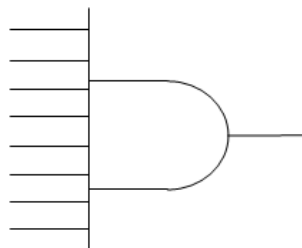


Figure 1.17 AND gate with 8 inputs

The IEEE standard specifies two different types of symbols for logic gates [8]: the distinctive- and rectangular-shape symbols. Figure 1.16 shows the main distinctive-shape symbols and Figure 1.18 compares the AND and NAND gate symbols. Both of them are frequently used and the standard says that it has no preference between them. Since most digital designers and computer-aided design (CAD) systems prefer the distinctive-shape symbols, these symbols are used in this book.

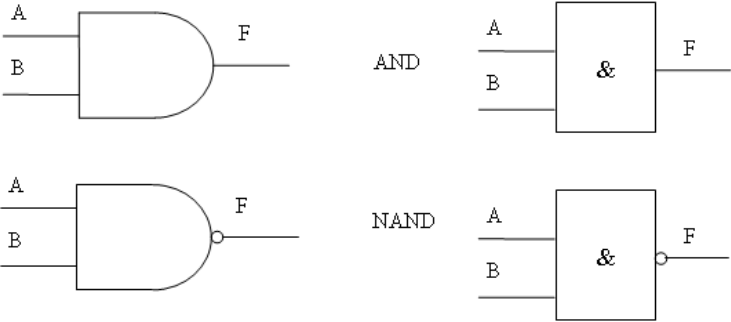


Figure 1.18 Example for distinctive- and rectangular shape logic symbols

Standard Forms for Logic Functions, Synthesis Using Standard Expressions and the Corresponding Circuits with NAND or NOR Gates

All logic functions can be specified using AND, OR and NOT operations. Both canonical forms: Sum of Products (SOP) and Product of Sums (POS) can be specified and implemented by two-level AND-OR or OR-AND gate networks, respectively. Because the AND, OR and NOT operations can be implemented using either only NAND or only NOR gates, then based on the respective canonical forms all logic functions can be implemented with homogeneous two-level NAND or NOR gate networks. Consider the sum-of-products expression:

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

The two-level AND-OR circuit consists of a number of AND gates equal to the number of terms followed by a single OR gate. The logic circuit is shown in Figure 1.19 a. Next we apply De Morgan’s theorem to the above function

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

and therefore $F = \overline{\overline{\overline{AB}} \overline{\overline{AC}}} = \overline{\overline{AB} \overline{\overline{AC}}}$. The corresponding circuit with NAND gates is shown in Figure 1.19 b.

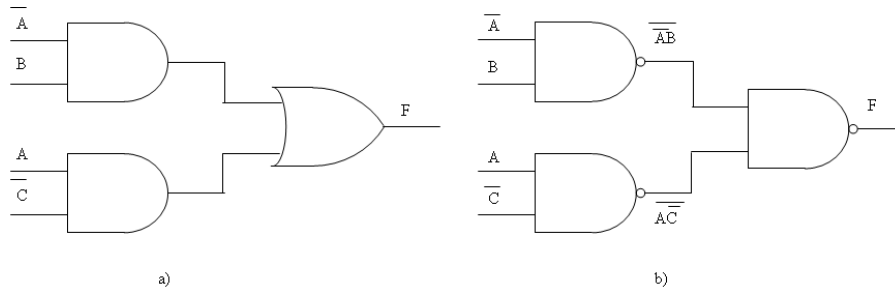


Figure 1.19 a) AND-OR circuit; b) corresponding circuit with NAND only

Although, the expression $F = \overline{\overline{\overline{AB}} \overline{\overline{AC}}}$ looks more complicated than $F = \overline{AB} + \overline{AC}$, the circuit built up from NAND gates (Figure 1.19 b) has the advantage that it is built up from the same gate types and consists of less transistors. If we start with a product-of-sums expression, the resulting circuit will be a two-level OR-AND structure. If the POS expression is:

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

Using De Morgan's theorems, we can transform the expression as follows:

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

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

The POS function two-level OR-AND structure and the corresponding NOR circuit is shown in Figure 1.20.

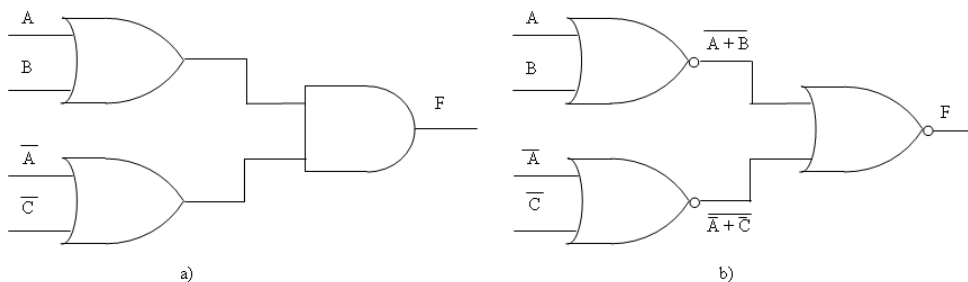


Figure 1.20 a) OR-AND circuit; b) corresponding circuit with NOR only

Problems

1.1 Simplify the following expressions as far as possible:

a) $F_1 = \overline{A}BCD + A\overline{B}CD + A\overline{B}C\overline{D} + A\overline{B}C\overline{D}$

b) $F_2 = AB + A\overline{B} + ABC\overline{D}$

c) $F_3 = \overline{ABC} + \overline{A+B+C}$

1.2 For three logic variables prove the identity:

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

1.3 Evaluate the following expressions for $A = B = C = 1$; $D = 0$; $E = 1$

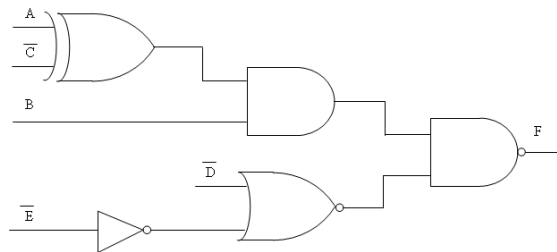
a) $(\overline{A}B + A\overline{B}) + (\overline{B} + C)D\overline{E}$

b) $\overline{A}(B + E) + ABC\overline{D}$

c) $AB(\overline{C} + E) + \overline{B}(D + \overline{E})$

1.4 Convert the EXCLUSIVE OR function into NAND form and show the corresponding circuit.

1.5 Write a Boolean expression for the logic diagram shown below:



1.6 Find the simplest expressions in the following Karnaugh maps:

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

	CD	00	01	11	10
AB					
00					
01		1	1	1	1
11		1	1		
10		1	1		

Chapter 2

Number Systems

In digital computers the information is represented in a string of ON and OFF states of logic variables, a series of logic-1s and logic-0s. This chapter covers the positional number systems (decimal, binary, octal, and hexadecimal), the number conversions, representations of integer and real numbers and arithmetic operations. Various codes and the code conversion are studied. Finally, encoders, decoders, multiplexers, demultiplexers and various comparators are discussed.

2.1 Positional Number Systems

In the positional number system, or so called radix-weighted positional number system the value of a number is a weighted sum of its digits. The value associated with a digit is dependent on its position. In general, the numbers, in the base r (radix) system are of the form:

$$N = a_j r^j + a_{j-1} r^{j-1} + \dots + a_1 r^1 + a_0 r^0 + a_{-1} r^{-1} + \dots + a_{-k} r^{-k}$$

Where: r is the base of the number system, $j, j-1, \dots, -(k-1), -k$ scalars, $a_j, a_{j-1}, \dots, a_1, a_0, a_{-1}, \dots, a_{-k}$ natural numbers between 0 and $r - 1$, inclusive.

Decimal Number System

The well known decimal system is just one in the class of number systems which belongs to the weighted positional number system. The decimal system is the most commonly used in our daily arithmetic. The numbers in combination of 10 symbols (digits) are called the decimal number system. This is a grouping system based on the repetition of symbols to note the number of each power of the base, in this case 10. The distinct digits (0 – 9) are multiplied by the power of 10, and it is significant that the position occupied by each digit. “.” is called the radix point. In the case of decimal number 356.21:

$$N = 356.21 = (3 \times 10^2) + (5 \times 10^1) + (6 \times 10^0) + (2 \times 10^{-1}) + (1 \times 10^{-2})$$

Binary Number System

In the binary system, the base is 2 ($r = 2$) and the symbols are 0 and 1. These numbers in positional code are expressed as power series of 2, and are called bits (binary digit).

In the expression $N = a_j r^j + a_{j-1} r^{j-1} + \dots + a_1 r^1 + a_0 r^0 + a_{-1} r^{-1} + \dots + a_{-k} r^{-k}$

a_j denotes the most significant bit (MSB);

a_{-k} denotes the least significant bit (LSB)

Conversion from binary to decimal

For example, consider the binary number 1101.01. This is expanded as:

$$\begin{aligned} 1101.01_2 &= (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) + (0 \times 2^{-1}) + (1 \times 2^{-2}) \\ &= 8 + 4 + 1 + 0.25 = 13.25_{10} \end{aligned}$$

Conversion from decimal to binary

Several methods exist to convert from decimal to binary and vice versa. In the next two methods are presented:

Method 1 Descending Powers of Two and Subtraction

For numbers less than thousands, this method offers a very rapid and easy technique. The steps are:

- Find the greatest power of 2 that is close and less than the decimal number, and after that calculate the difference between them.
- Choose a next (lower) power of 2 which is close and less than the subtraction result.
- Repeat the above mentioned operations until the sum of the powers of 2 will give the decimal number. The binary answer is composed from 1s in the positions where the power of 2 fit into the decimal number and 0s otherwise.

For example, find the 156 decimal number binary equivalents. We list the power of 2 s and we “build” the corresponding number.

2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
256	128	64	32	16	8	4	2	1
0	1	0	0	1	1	1	0	0


Method 2 Division by Two with Remainder

If the number has a radix point, as a first step, it is important to separate the number into an integer and a fraction part, because the two parts have to be converted differently.

The conversion of a decimal integer part to a binary number is done by dividing the integer part to 2 and then writing the remainders (0 or 1). Proceed with all successive quotients the same way and accumulate the remainders.

For example, find the 56 decimal number binary equivalent:

Number divided by 2	Result	Remainder	
56/2	28	0	LSB
28/2	14	0	
14/2	7	0	
7/2	3	1	
3/2	1	1	
1/2	0	1	MSB




$$56_{10} = 111000_2$$

The conversion of a decimal fraction part to a binary number is done by multiplying the fractional parts by 2 and accumulating integers.

For example, find the 0.6875 decimal number binary equivalent:

Number multiplying by 2	Integer	Result
0.6875 x 2 =	1	0.3750
0.3750 x 2 =	0	0.7500
0.7500 x 2 =	1	0.5000
0.5000 x 2 =	1	0.0000



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

$$\text{And } 56.6875_{10} = 111000.1011_2$$

Octal and Hexadecimal Numbers

Positional number systems with base 8 (octal) and base 16 (hexadecimal) are used in digital computers.

In octal system the eight required digits are 0 to 7, and the radix is 8.

The hexadecimal system (base 16) uses sixteen distinct symbols: numbers from 0 to 9, and letters A, B, C, D, E, F to represent values between ten to fifteen.

Conversion from octal or hexadecimal to decimal

The conversion is similar to the previous subsection Method 2. This approach is called: Division by Eight or Sixteen with Remainder. Here the numbers in the positional code are expressed as power series of 8 and 16.

For example:

$$327_8 = 3 \times 8^2 + 2 \times 8^1 + 7 \times 8^0 = 215_{10}$$

$$A2D.C_{16} = 10 \times 16^2 + 2 \times 16^1 + 13 \times 16^0 + 12 \times 16^{-1} = 2605.75_{10}$$

Conversion from octal and hexadecimal to binary

For example, to convert from binary to octal, the binary number three bit groups were separated, that could be converted directly:

011	010	111	.	001	binary number
3	2	7	.	1	octal equivalent

Vice versa:

4	5	6	.	2
100	101	110	.	010

To convert from hexadecimal to binary and reverse, the binary number four bit groups were separated, that could be converted directly:

$$1010 \ 0010 \ 1101_2 = A2D_{16}$$

$$7F3_{16} = 0111 \ 1111 \ 0011_2$$

The **non-positional number systems** uses for example Roman numerals (I = 1, V = 5, X = 10, L = 50, C = 100, D = 500 and M = 1000).

2.2 Binary Arithmetic

Arithmetic operations with numbers in base r follow the same rules as for decimal numbers. The addition, subtraction, multiplication and division can be done in any radix-weighted positional number system.

In digital computers arithmetic operations are performed with the binary number system (radix = 2). The binary system has several mathematical advantages: i.e. easy to perform arithmetic operations and simple to make logical decisions. The same symbols (0 and 1) are used of arithmetic and logic. Now we will review the four basic arithmetic operations, then the handling of the case of negative (signed) numbers.

Binary Addition

The algorithm of binary addition is similar to that of decimal numbers: aligning the numbers with the same radix, starting the addition with the pair of least significant digits. The **half adder** adds two single binary digits A and B. It has two outputs, sum (S) and carry (C). The rules for two-digit binary addition are the following:

A	B	Sum	Carry
0	+	0	0
1	+	1	0
0	+	1	0
1	+	0	1 (to the next more significant bit)

The half adder logic diagram is shown in Figure 2.1.

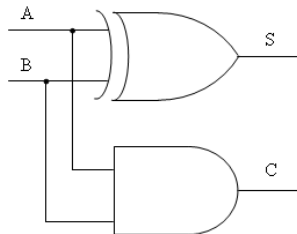


Figure 2.1 Half adder logic diagram

The rules of binary addition (without carries) are the same as the truths of the EXCLUSIVE OR (XOR) gate [9]. Circuit implementation requires 2 outputs; one to indicate the sum and another to indicate the carry. Two digits (bits) at the actual position and the carry from the previous position should be added. The process is then repeated. The **full adder** is a fundamental building block in many arithmetic circuits, which adds three one-bit binary numbers (C, A, B) having two one-bit binary output numbers, a sum (S) and a carry (C1).

C	A	B	S	C1
0	1	0	= 1	0
0	1	1	= 0	1 (to the next more significant bit)
1	1	0	= 0	1 (to the next more significant bit)
1	1	1	= 1	1 (to the next more significant bit)

The full adder logic diagram is shown in Figure 2.2.

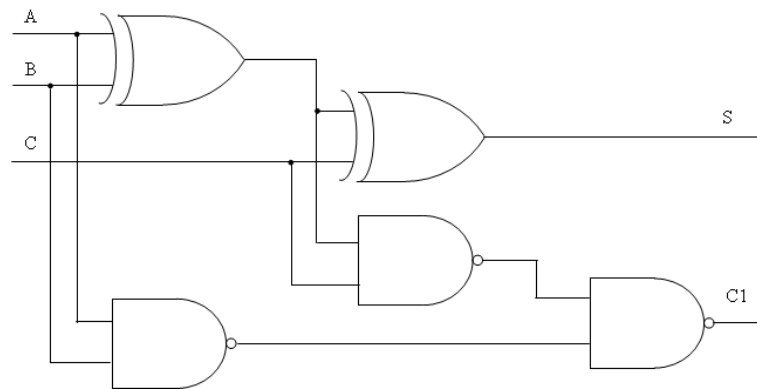


Figure 2.2 Full adder logic diagram

The following is an example of binary addition:

$$\begin{array}{r}
 1111 \quad \text{Carries} \\
 1001.011_2 \quad \text{Augend} \\
 + 1100.111_2 \quad \text{Addend} \\
 \hline
 10110.010_2 \quad \text{Sum}
 \end{array}$$

Binary Subtraction

In many cases binary subtraction is done in a special way by binary addition. One simple building block called adder can be implemented and used for both binary addition and subtraction. Using the borrow method; the basic rules are summarized in the next:

Borrow	A	B	Difference
0	0	0	= 0
0	1	0	= 1
0	1	1	= 0
0	0	1	= 1 and borrow 1 from the next more significant bit
1	0	0	= 1 and borrow 1 from the next more significant bit
1	0	1	= 0 and borrow 1 from the next more significant bit
1	1	0	= 0
1	1	1	= 1 and borrow 1 from the next more significant bit

When a larger digit is to be subtracted from a smaller digit it is necessary to “borrow” from the next-higher-order digit position. The following example illustrates binary subtraction:

$$\begin{array}{r}
 010 \quad \text{Modification of minuend resulting from borrowing} \\
 \cancel{1}0\cancel{1}0.11_2 \quad \text{Minuend} \\
 - 111.01_2 \quad \text{Subtrahend} \\
 \hline
 11.10_2 \quad \text{Difference}
 \end{array}$$

Binary Multiplication

The rules for two-digit binary multiplication are the next:

A	B	Product
0	x 0	= 0
1	x 0	= 0
0	x 1	= 0
1	x 1	= 1 and no carry or borrow bits

The rules of binary multiplication are the same as the truths of the AND gate [9]. In a very similar way to the decimal multiplication an array of partial products are formed and binary added. The following is an example of binary multiplication:

1 1.1 ₂	Multiplicand
x 1 0 1 ₂	Multiplier
1 1 1	Array of partial product (1 x 111)
+ 0 0 0	Array of partial product (00 x 111)
0 1 1 1	
+ 1 1 1	Array of partial product (100 x 111)
1 0 0 1.1 ₂	Product

Binary Division

Binary division is the repeated process of subtraction, just as in decimal division [9]. A trial quotient digit is selected and multiplied by the divisor. The product is subtracted from the dividend to determine whether the trial quotient is correct. The principle of binary division is seen by the next example:

	110101 ₂	Quotient
Divisor	11 ₂) 10100000 ₂	Dividend
	$\begin{array}{r} \underline{-11} \downarrow \\ 100 \\ \underline{-11} \downarrow \\ 10 \\ \underline{-00} \downarrow \\ 100 \\ \underline{-11} \downarrow \\ 10 \\ \underline{-00} \downarrow \\ 100 \\ \underline{-11} \downarrow \\ 1 \\ \hline 1_2 \end{array}$	
	1 ₂	Remainder

2.3 Signed Binary Numbers

The positive integers and the number zero can be represented as unsigned binary numbers using an n-bit word. When working with any kind of digital electronics in which numbers are being represented, it is important to distinguish both positive and negative binary numbers. The three representations of signed binary numbers will be discussed in the next. These approaches involve using one of the digits of the binary number to represent the sign of the number [10].

Sign-Magnitude Representation

To mark the positive and negative quantities instead of using plus and minus sign we will use two additional symbols 0 and 1. In this approach the information's left bit (the most-significant bit, MSB) is the sign bit, where 0 denotes positive and 1 denotes negative value. The rest of the bits represent the number magnitude. This method is simple to implement, and is useful for floating point representation. The disadvantage of sign-magnitude representation is that the sign bit is independent of magnitude, and mathematical operations are more difficult. It is very important to not confuse this representation with unsigned numbers! Table 2.1 illustrates this concept, including all three representations of signed binary numbers using 4 bits. Here, the MSB bit (sign bit) is separated from the remaining 3 bits which denote the magnitude in the binary number system. It can be noted that 0 has two different representations, and can be both + 0 and - 0!

1's-Complement Representation

The simplest of these methods is called 1's complement, which can be derived by just inverting all the bits in the number. Reversing the digits, by changing all the bits that are 1 to 0 and all the bits that are 0 to 1 is called complementing a number. The positive numbers 1's complement representation is the same as in the sign-magnitude approach and the 0 has again two different representations. In this approach the MSB bit also shows the sign of the number (all of the negative values begin with a 1, see Table 2.1).

2's-Complement Representation

The 2's complement number representation is most commonly used for signed numbers on modern computers. An easier way to compute the 2's complement of a binary integer is to consider the 1's complement of the number plus 1. The 8 bit representations of the integer number -9 are:

```
Signed-Magnitude representation: 1|0001001
1's complement:                 1|1110110
2's complement:                 1|1110110
                                +      1
                                1|1110111
```

For further examples using 4 bits see Table 2.1.

Signed decimal equivalent	Sign-magnitude representation	1's complement representation	2's complement representation
+7	0 111	0 111	0 111
+6	0 110	0 110	0 110
+5	0 101	0 101	0 101
+4	0 100	0 100	0 100
+3	0 011	0 011	0 011
+2	0 010	0 010	0 010
+1	0 001	0 001	0 001
0	0 000	0 000	0 000
-1	1 000	1 111	1 111
-2	1 001	1 110	1 110
-3	1 010	1 101	1 110
-4	1 011	1 100	1 101
-5	1 100	1 011	1 100
-6	1 101	1 010	1 011
-7	1 110	1 001	1 010
-8	1 111	1 000	1 001
	---	---	1 000

Table 2.1 Three representations of signed binary numbers using 4 bits

Addition with 2's Complement Representation

If the complement representation of signed numbers is used there is no need for both adder and subtractor unit in a computer.

Let's assume two n-bit signed numbers M and N represented in signed 2's complement format. The sum $M + N$ can be obtained including their sign bits to get the correct sum. A carry out of the sign bit position is discarded. In the next, addition in the 2's complement representation examples are given (using 5 bits).

$\begin{array}{r} (+7) \quad 0 \ 0111 \\ + (+5) \quad 0 \ 0101 \\ \hline (+12) \quad 0 \ 1100 \end{array}$	$\begin{array}{r} (+7) \quad 0 \ 0111 \\ + (-5) \quad 1 \ 1011 \\ \hline 1 \ 0 \ 0010 \\ \leftarrow \text{Carry discard} \end{array}$
$\begin{array}{r} (-7) \quad 1 \ 1001 \\ + (+5) \quad 0 \ 0101 \\ \hline (-2) \quad 1 \ 1110 \\ \swarrow \\ (-2) \text{ 2's complement} \end{array}$	$\begin{array}{r} (-7) \quad 1 \ 1001 \\ + (-5) \quad 1 \ 1011 \\ \hline 1 \ 1 \ 1 \ 0100 \\ \swarrow \quad \nwarrow \\ \text{Carry discard} \quad (-12) \text{ 2's complement} \end{array}$

If the sum of two n-bit numbers results in an n + 1 number an overflow appears. The first step in the detection of such an error is the examination of the sign of the result. The overflow detection can be implemented using either hardware or software, and depends on the signed or unsigned number system used.

2.4 Binary Codes and Decimal Arithmetic

The binary number system, handling only two digit symbols, is the simplest system for a digital computer. From the user point of view it is easy to compute and operate with decimal numbers. A combination of binary and decimal approaches, keeping their advantages, result in a system in which the digits of the decimal system are coded by groups of binary digits. The basic concept is to convert decimal numbers to binary, to perform all arithmetic calculations in binary, and then convert the binary result back to decimal.

The best known scheme to code the decimal digits is the 8421 binary-coded-decimal (8421 BCD) scheme. In this **weighted code** 10 decimal digits are represented by at least 4 binary digits. In 8421 BCD code each bit is weighted by 8, 4, 2 and 1 respectively. For example the 8421 BCD representation of the decimal number 3581 is 0011 0101 1000 0001.

Beside 8421 BCD code other weighted codes have been used. These codes have fixed weights for different binary positions. It has been shown in [11] that exist 17 different set of weights possible for a positively weighted code: (3,3,3,1), (4,2,2,1), (4,3,1,1), (5,2,1,1), (4,3,2,1), (4,4,2,1), (5,2,2,1), (5,3,1,1), (5,3,2,1), (5,4,2,1), (6,2,2,1), (6,3,1,1), (6,3,2,1), (6,4,2,1), (7,3,2,1), (7,4,2,1), (8,4,2,1). It is also possible to have a weighted code in which some of the weights are negative, as in the 8, 4, -2, -1 code shown in Table 2.2.

Decimal digit	8421 binary code	Excess-3 code	2-out-of-5 code
0	0000	0011	11000
1	0001	0100	00011
2	0010	0101	00101
3	0011	0110	00110
4	0100	0111	01001
5	0101	1000	01010
6	0110	1001	01100
7	0111	1010	10001
8	1000	1011	10010
9	1001	1100	10100

Table 2.2 Binary codes for the decimal digits

This code has the useful property of being self-complementing: if a code word is formed by complementing each bit individually (changing 1's to 0's and vice versa), then this new code word represents the 9's complement of the digit to which the original code word corresponds [12].

The non-weighted codes don't have fixed weights for different binary positions. For example the excess-3 code is derived by adding $0011_2 = 3_{10}$ to the 8421 BCD representation of each decimal digit. The 2-out-of-5 code shown in Table 2.2 has the property that each code word has exactly two 1's.

Decimal Addition Using 8421 BCD Code

The 8421 BCD code is widely used and it is simplified as **BCD code**. Because of the popularity of this code in the next the addition operation is presented.

Addition is performed by individually adding the corresponding digits of the decimal numbers expressed in 4-bit binary groups starting from right to left. [13]. If the result of any addition exceeds nine (1001) then the number six (0110) must be added to the sum to account for the six invalid BCD codes that are available with a 4-bit number.

Perform the following decimal additions (24 + 15) in BCD code.

$$\begin{array}{r}
 24 \quad 0010 \ 0100 \ (24 \text{ in BCD}) \\
 + 15 \quad + 0001 \ 0101 \ (15 \text{ in BCD}) \\
 \hline
 39 \quad 0011 \ 1001 \ (\text{No carry, no illegal code})
 \end{array}$$

When considering the two decimal numbers 26 and 37, it can be observed that the sum $6 + 7 = 13 > 9$ and a correction is necessary to skip over the six illegal combinations (by adding a correction factor of $6_{10} = 0110_2$). Thus, we have

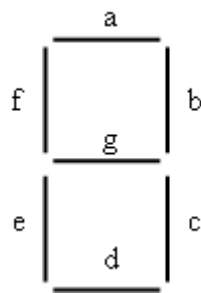
$$\begin{array}{r}
 26 \quad 0010 \ 0110 \\
 + 37 \quad + 0011 \ 0111 \\
 \hline
 \text{Correction} \quad 0101 \ 1101 \\
 \text{Sum} \quad 63 \quad + 0000 \ 0110 \\
 \hline
 \quad \quad 0110 \ 0011
 \end{array}$$

Consider the following addition: $28 + 59$. When the sum of the LSB digits of the two numbers ($8 + 9$) is greater than 15 it is necessary to introduce a correction. In this approach a correct code group results but with an incorrect sum.

$$\begin{array}{r}
 \text{carry} \\
 \leftarrow \\
 28 \quad 0010 \ 1000 \\
 + 59 \quad + 0101 \ 1001 \\
 \hline
 \text{Correction} \quad 1000 \ 0001 \\
 \text{Sum} \quad 87 \quad + 0000 \ 0110 \\
 \hline
 \quad \quad 1000 \ 0111
 \end{array}$$

7-Segment Code

A very useful decimal code is the **7-segment code** which is able to show numeric info on seven-segment displays. The 7-segment display (see Figure 2.3) consist of 7 LEDs (light emitting diodes), each one controlled by an input where 1 means “on”, 0 means “off”. The decimal digit and the corresponding 7-segment code are shown in Table 2.3.



Decimal Digit	7-Segment Code						
	a	b	c	d	e	f	g
0	1	1	1	1	1	1	0
1	0	1	1	0	0	0	0
2	1	1	0	1	1	0	1
3	1	1	1	1	0	0	1
4	0	1	1	0	0	1	1
5	1	0	1	1	0	1	1
6	1	0	1	1	1	1	1
7	1	1	1	0	0	0	0
8	1	1	1	1	1	1	1
9	1	1	1	1	0	1	1

Figure 2.3 7-segment display

Table 2.3 Decimal digit and the corresponding 7-segment code

Gray Code

The most useful **unit distance code** is the **Gray code** which is shown in Table 2.4. This unweighted code has such a sequence that any adjacent code words differ only in one bit (see subsection 1.4 Three - Variable Karnaugh Maps). The attractive feature of this code is the simplicity of the algorithm for translating from the binary number system into the Gray code. [12]

Decimal	Binary			Gray		
	b2	b1	b0	g2	g1	g0
0	0	0	0	0	0	0
1	0	0	1	0	0	1
2	0	1	0	0	1	1
3	0	1	1	0	1	0
4	1	0	0	1	1	0
5	1	0	1	1	1	1
6	1	1	0	1	0	1
7	1	1	1	1	0	0

Table 2.4 The 3 bit Gray code

This algorithm is described by the expressions:

$$g0 = b0 \oplus b1$$

$$g1 = b1 \oplus b2$$

$$g2 = b2$$

Unit-distance codes, which could minimize errors, are used in devices for converting analog or continuous signals such as voltages or shift rotations into binary numbers which represent the magnitude of the signal. Such a device is called an analog-digital converter [12].

Error Detection

In general data transfer between various parts of a computer system, the transmission over communication channels or their storage in memory is not completely error free. For the purpose of increasing system reliability, special features are included in many digital systems, i.e. to introduce some redundancy in encoding the information handled in the system. For example, the error detecting properties of the 2-out-of-5 code is based on its feature to have exactly two 1's within a code group. Not all codes have error detecting capability.

A simple error detecting method is the calculus of a **parity bit**, which is then appended to original data. The parity type could be: even or odd. The parity bit is added to each code word so as to make the total number of 1's in the resultant string even or odd. [14].

For example, when the parity type is even, the result is an even number of 1's

100 0100 → **0** 100 0100

110 0100 → **1** 100 0100

In data transmission, the sender adds the parity bit (message bit) to the existing data bits before forwarding it which is compared to the expected parity (check bit) calculated from the receiver.

Generating even parity bit is just an XOR function. In a similar way, generating odd parity bit is just an XNOR function.

To minimize the disadvantages of this **single** error detection method (cannot determine which bit position has a problem) the following rules have to be observed. The necessary and sufficient conditions for any set of binary words to be a single-error-correcting code is that the minimum distance between any pair of words be three [12].

In general, if the Hamming distance is D (see subsection 1.4), Hamming Distance is equal to the number of bit positions in which 2 code words differ), [14] to detect k -single bit error, minimum Hamming distance is

$$D(\min) = k + 1$$

The Hamming Code is a type of Error Correcting Code (ECC) which adopts parity concept, having more than one parity bit, providing error detection and correction mechanism. To correct k errors $D(\min) = 2k + 1$ is required.

Alphanumeric Codes

Several codes have been proposed to represent numeric information and various characters. The nonnumeric ones are called **alphanumeric codes**. The characters are for example: alphabet letters, special symbols, punctuation marks, special control operations. The commonly used alphanumeric code is the American Standard Code for Information Interchange (ASCII). The 7-bit version of this code is frequently completed with an eighth bit, the parity bit.

Another encoding, **Unicode** is a computing industry standard for the consistent encoding [15]. It can be implemented by so called UTF-8, UTF-16 character encodings. For example UTF-8 uses one byte for any ASCII characters, and up to four bytes for other characters.

2.5 Functional Blocks

The traditional process of logic synthesis is based on the application of logic gates. Its more modern variant makes the use of a composition of smaller, simpler circuits and programmable logic devices. However in many cases it is more advantageous to use a logic synthesis procedure based on the application of logic functional blocks.

In this subsection we will give an overview of some important and useful basic combinational functional blocks. In order to design new circuits, design hierarchy, the so called Top-Down, Bottom-Up, Meet in the Middle Design Approaches or Computer-Aided Design (CAD) tools could be used [16].

Code Converter

A code converter is an important application of combinational networks (see subsection 1.5). Such a digital system is able to transform information from one code to another. For example a BCD-to-Excess-3 code converter is useful

in digital arithmetic [16]. To understand the „machine language” a set of code conversions has to be applied. Figure 2.4 shows a possible application, where a signal in Gray code transmitted by a position sensor is received by a Gray-Binary converter (a typical application for Gray code is in absolute position sensing) and the result is converted in normal (8421) BCD code. At the end, the display unit applies BCD to 7-segment code conversion. In this way the output can be easy evaluated.

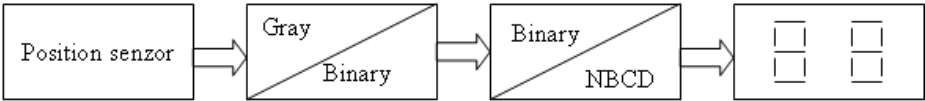


Figure 2.4 Code conversions

Code converters are typically multiple input-multiple output combinational circuits. They can be realized by appropriate gate networks or using Read only Memories.

Binary Decoders

A combinational circuit that converts binary information from n coded inputs to a maximum 2ⁿ coded outputs is called n-to-2ⁿ decoder, more generally n-to-m decoder, m ≤ 2ⁿ [17]. Figure 2.5 shows a binary decoder as a black box.

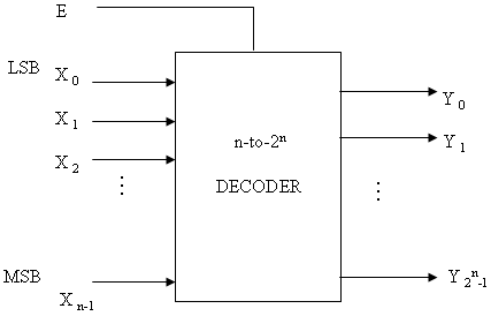


Figure 2.5 Binary decoder as a black box

Enable input (E): it must be on (active) for the decoder to function, otherwise its outputs assume a single ”disabled” output code word.

2-to-4 Decoder

In a 2-to 4 decoder, 2 inputs, A0, A1 are decoded into $2^2 = 4$ outputs, D0 through D3. Each output represents one of the minterms of the 2 input variables. The truth table and the logic circuit without Enable input are given in Figure 2.6.

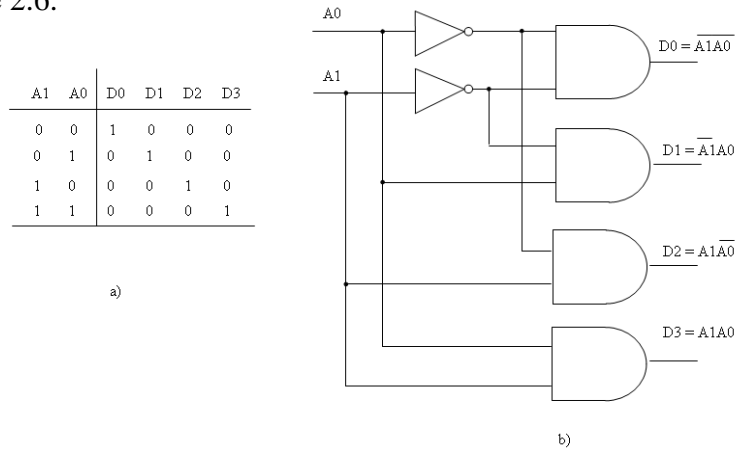


Figure 2.6 2-to-4 line decoder without Enable; a) truth table, b) gate level logic diagram

Decoder output lines implement minterm functions. Any combinational circuit can be constructed using decoders and OR gates [18]. The 2-to-4 decoder truth table and the logic circuit with Enable input are given in Figure 2.7. In this case the additional gate level produces time delay, which can be avoided by using 3 input AND gates instead of 2 input AND gates in Figure 2.6 b.

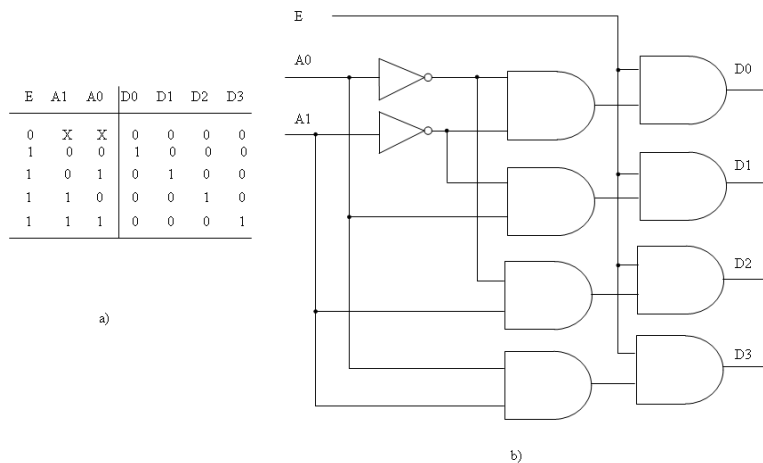


Figure 2.7 2-to-4 line decoder with Enable a) truth table; b) gate level logic diagram

Decoder expansion means to construct larger decoders from small ones. The next example (see Figure 2.8) shows the interconnection of two 2-to-4 decoders in order to have the required 3-to-8 decoder size. If $A_2 = 0$: enables top decoder, when $A_2 = 1$: enables bottom decoder.

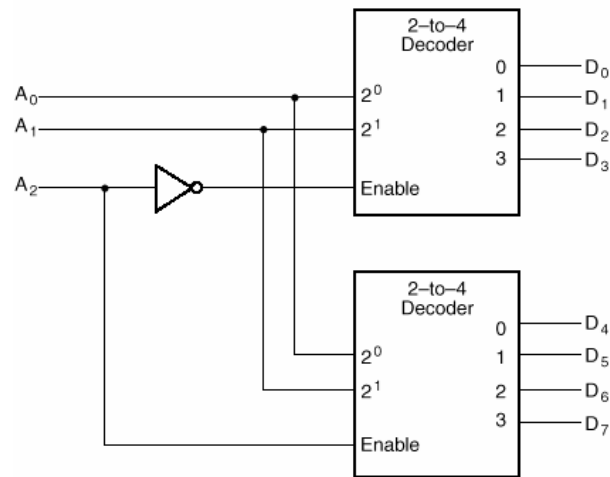


Figure 2.8 3-to-8 decoder from 2-to-4 decoders

Binary Encoders

An encoder is a multi-input combinational logic circuit that executes the inverse operation of a decoder. In general, it has 2^n input lines and n output lines. The output lines generate the binary equivalent of the input line whose value is 1 [19]. Figure 2.9 shows a binary encoder as a black box.

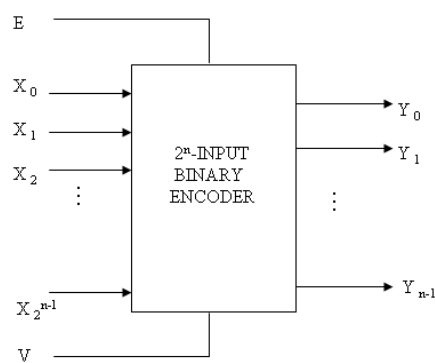


Figure 2.9 Binary encoder as a black box

If the enable signal $E = 0$ then all outputs are 0 else $Y_j = F(X_0, X_1, \dots, X_{2^j-1})$, $j = 0 \dots n-1$. If the valid signal is equal to 1 ($V = 1$) the valid code is present at the outputs; otherwise $V = 0$.

4-to-2 Encoder

Using a 4-to-2 encoder, 4 inputs, $D_0 - D_3$ are encoded into 2 outputs, A_0 and A_1 . The truth table and the logic circuit without Enable input are given in Figure 2.10.

$$A_1 = D_3 + D_2$$

$$A_0 = D_3 + D_1$$

$$V = \overline{D_3 D_2 D_1 D_0} + \overline{D_3 D_2 D_1 D_0} + \overline{D_3 D_2 D_1 D_0} + \overline{D_3 D_2 D_1 D_0}$$

D3	D2	D1	D0	A1	A0	V
0	0	0	1	0	0	1
0	0	1	0	0	1	1
0	1	0	0	1	0	1
1	0	0	0	1	1	1
x	x	x	x	x	x	0

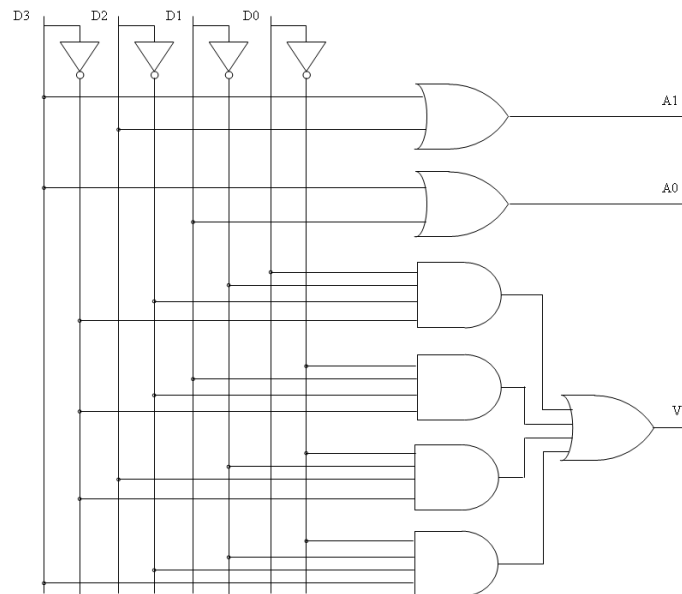


Figure 2.10 4-to-2 line encoder without Enable truth table and gate level logic diagram

Priority Encoders

Multiple asserted inputs are permitted; and one has priority over all others. A **valid output indicator**, designated by V, is set to 1 only when one or more inputs are equal to 1. $V = D3 + D2 + D1 + D0$ by inspection. Figure 2.11 shows the truth table and the corresponding logic circuit [16] of a 4-to-2 priority encoder

Inputs				Outputs		
D3	D2	D1	D0	A1	A0	V
0	0	0	0	X	X	0
0	0	0	0	X	X	0
0	0	0	0	X	X	0
0	0	0	0	X	X	0
0	0	0	0	X	X	0

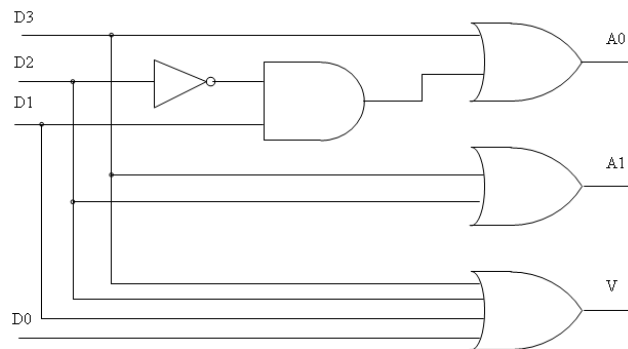


Figure 2.11 Priority encoder truth table and logic circuit

Multiplexer (MUX)

Multiplexers work as selectors, which choose one input to pass through to the output. A 2 x 1 multiplexer has two data inputs (I_0 and I_1), one select input S and one data output (D). Figure 2.12 a) shows the internal structure of a two input MUX. At the output of the simple AND - OR combinational circuit if $S = 0$, appears the I_0 value, and if $S = 1$, I_1 's input value passes through.

In general, an $M \times 1$ multiplexer has M data inputs, $\log_2(M)$ select inputs, and one output. Using another notation, if $M = 2^n$, the 2^n -to-1 multiplexer works with 2^n data inputs, n select inputs and one output.

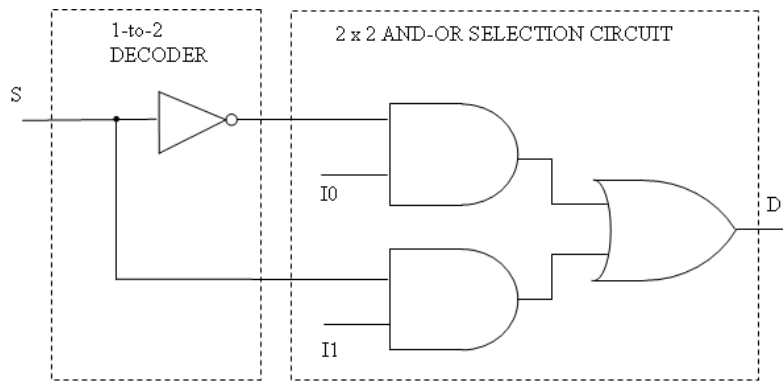


Figure 2.12 2-to-1 multiplexer

N-bit $M \times 1$ Multiplexer

In many applications multiplexers are often used to pass through N -bit data items [20]. For example, if the inputs A and B consist of four-four bits (a_3, a_2, a_1, a_0 , and b_3, b_2, b_1, b_0 respectively) we need four 2 x 1 MUXes (called 4-bit 2 x 1 MUX) to multiplex the inputs to four-bit output C (denoted by c_3, c_2, c_1, c_0). Figure 2.13 shows the internal design using four 2 x 1 MUXes and the corresponding block diagram. The 2 x 1 multiplexers are connected to the same select input.

Any Boolean function of n variables can be implemented using a 2^n -to-1 multiplexer [16].

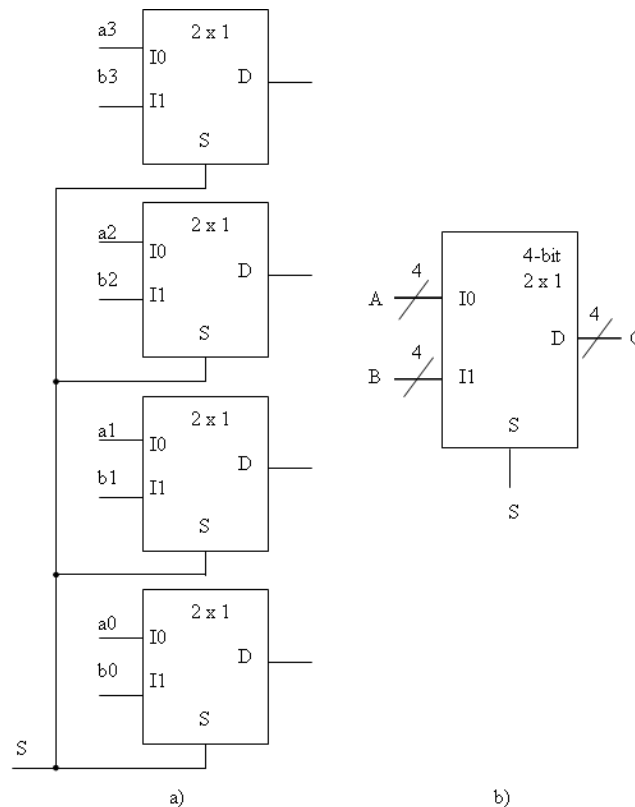


Figure 2.13 4-bit 2 x 1 MUX a) internal design; b) block diagram

Demultiplexer

The demultiplexer is basically a decoder, which performs the inverse of a multiplexing operation. In general, a 1-to- 2^n demultiplexer has one data input, which is transmitted to one of the 2^n possible output lines. The selection of the proper output depends on the n select lines.

Comparators

A comparator is used to compare binary numbers in order to indicate if they are equal or if one is greater than the other. A comparator is used in applications where some varying signal level is compared to a fixed level (usually a voltage reference) [21].

Equality (Identity) Comparator

In general, a combinational circuit able to compare two n -bit inputs generating a 1 or a 0 at its output, depending on whether the inputs are the same or not, is called an equality comparator. Comparator design first step is to write the combinational circuit truth table from which those situations (lines) are selected where all the input bits are equal. The next step is the comparator output function minimalization which is complicated for more than 4-bit input binary numbers. A simpler design can be obtained by recalling the XNOR gate (see subsection 1.5) property whose output is set to 1 if the gate's two input bits are equal. For example, a 4-bit equality comparator is composed from four XNOR gates, and each unit detects if the corresponding bits are equal. Denoting the two 4-bit inputs by $A = a_3a_2a_1a_0$ and $B = b_3b_2b_1b_0$, the comparator output will indicate equality (logic-1 value) if $A = B$ ($a_3 = b_3, a_2 = b_2, a_1 = b_1, a_0 = b_0$). Figure 2.14 shows a 4-bit equality comparator internal design, and its block symbol [20].

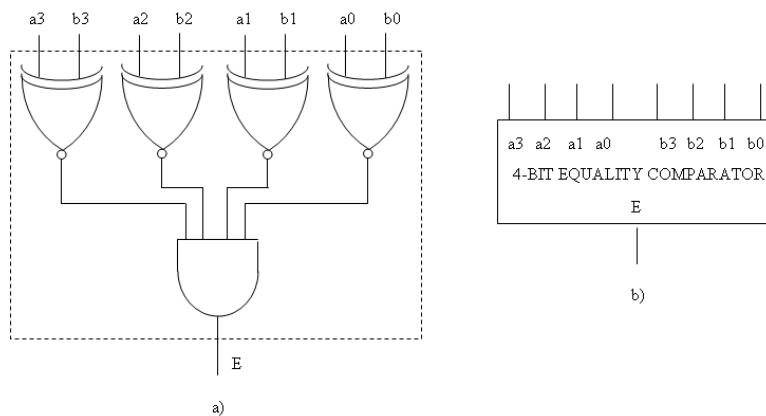


Figure 2.14 4-bit equality comparator internal design, and its block symbol

Magnitude Comparator- Carry-Ripple Style

An N -bit magnitude comparator is a combinational circuit able to compare two N -bit binary numbers A and B , and indicates if $A > B$, $A = B$, or $A < B$. In general, the comparison of two binary numbers starts from checking their MSB bit (most significant bit, section 2.1) values followed by comparing the remaining bits down to the LSB (least significant bit) bit pairs.

As long as bit pairs are equal we need to compare the next lower bit pair [20]. The bit pair is different if $a_i = 1$ and $b_i = 0$ (case $A > B$) or $a_i = 0$ and $b_i = 1$

(case $A < B$). Figure 2.15 illustrates a 4-bit magnitude comparator with identical units in each stage and its block symbol, using the notations:
G (is 1 when $A > B$)
E (is 1 if two numbers are equal) and
L (is 1 when $A < B$).

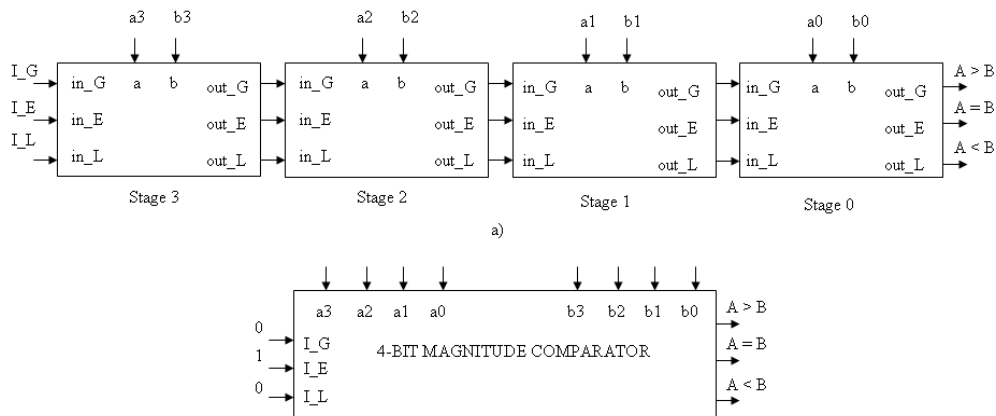


Figure 2.15 4-bit magnitude comparator a) internal design; b) block symbol

Problems

2.1 Convert the following binary numbers to decimal, octal and hexadecimal:

a) 11001100

b) 100000011

c) 11100011

2.2 Perform the following operations in the binary number system

a) $110011.11 + 1101.1$

b) $110011.11 - 1101.1$

c) 101.1×111

d) $100100 : 110$

2.3 Perform the following addition and subtraction of signed binary numbers in the 2's complement number representation

a) $10001111 + 101101$

b) $11101111 - 101000$

2.4 Express the following signed decimal numbers as signed 8-bit binary numbers in sign-magnitude, 1's-complement, and 2's complement representations

a) +33

b) +125

c) -48

d) -113

- 2.5 Give the coded representation of the decimal numbers 346 and 418 in 8421 BCD code and perform the $346 + 418$ operation in BCD code.
- 2.6 Calculate the odd parity bit of the following strings
a) 1110001 b) 101010 c) 011100 d) 1111100
- 2.7 Design a 4 x 2 encoder using AND, OR, and NOT gate
- 2.8 Design a 8 x 3 encoder using AND, OR, and NOT gate
- 2.9 Design a 4 x 2 priority encoder using AND, OR, and NOT gate
- 2.10 Design a 3 x 8 decoder with and without enable using AND, OR, and NOT gates
- 2.11 Design a 4 x 16 decoder using AND, OR, and NOT gate
- 2.12 Design a 8 x 1 multiplexer using AND, OR, and NOT gate
- 2.13 Design a 16 x 1 multiplexer using AND, OR, and NOT
- 2.14 Design a 4-bit 4 x 1 multiplexer using 4 x 1 multiplexers
- 2.15 Design a 1 x 4 demultiplexer using AND, OR, and NOT gate
- 2.16 Design a 1 x 8 demultiplexer using AND, OR, and NOT gate
- 2.17 Design a BCD to 7-segment code converter
- 2.18 Design a decimal to Excess-3 code converter
- 2.19 Design a decimal to 2-out-of-5 code converter
- 2.20 Design a BCD to Excess-3 code converter

Chapter 3

Logic Circuits and Components

3.1 Digital Electronic Circuits

Logic circuits can be implemented using different digital electronic logic circuit families, each of which has its own advantages and disadvantages. Usually a system is built with circuits which belong from a selected single family, and the logic gates in this family are used to realize all the logical operations. The combination of various logic circuit families results in a hybrid approach. In this approach it is very important to take note of the systems compatibility, and interfacing circuits may be required. The choice of family to be used depends on availability of different logic functions, switching speed, power drain, signal voltage level, cost, noise immunity, power dissipation, circuits' density, flexibility, and other characteristics. In many cases the main goal is to design a circuit for the highest speed possible in order to minimize calculation time.

Digital systems are built up from digital circuits. Digital logic circuits are implemented using transistors and interconnections in complex semiconductor devices called integrated circuits (IC). An IC is a silicon semiconductor crystal (*chip*) that contains a network of transistors. The number of transistors determines the integration level [16]:

- **Small-scale Integration (SSI)**; several transistors (< 40) per chip
- **Medium-scale Integration (MSI)**; between 40 - 400 transistors per chip
Perform basic digital functions, e.g., 4-bit addition, multiplication, etc.
- **Large-scale Integration (LSI)**; between 400 and a few thousands of transistors per chip. Implement digital systems, e.g. small (micro-) processors and memories. For example, Intel i4004 has ca. 2300 transistors.
- **Very Large-scale Integration (VLSI)**; Several thousands to over 1 billion transistors per chip, implements complex digital systems, e.g., complex microprocessors, multi-processor systems on-chip, etc. For example, Intel i7-4770k has ca. 1.5 billion transistors.

In the next we will give an overview of the most popular logic families, transistor-transistor logic (TTL) and complementary-MOS (CMOS). The CMOS logic family is the slowest of the three logic families but also dissipates significantly less power than the medium-speed family, TTL, or the high-speed family ECL [22]. Nowadays, High-Speed CMOS and Advanced CMOS Logic (AHC and AHCT) products, with low-power consumption [23] are also used.

Transistor-Transistor Logic (TTL)

Transistor-Transistor Logic (TTL) is a very prominent logic family. Its name refers to the use of bipolar transistors throughout the circuit.

Inverter (NOT Gate)

The inverter is available in each of the logic families. A possible implementation is shown in Figure 3.1, where the bipolar transistor, connected in the common-emitter configuration serves as an inverter.

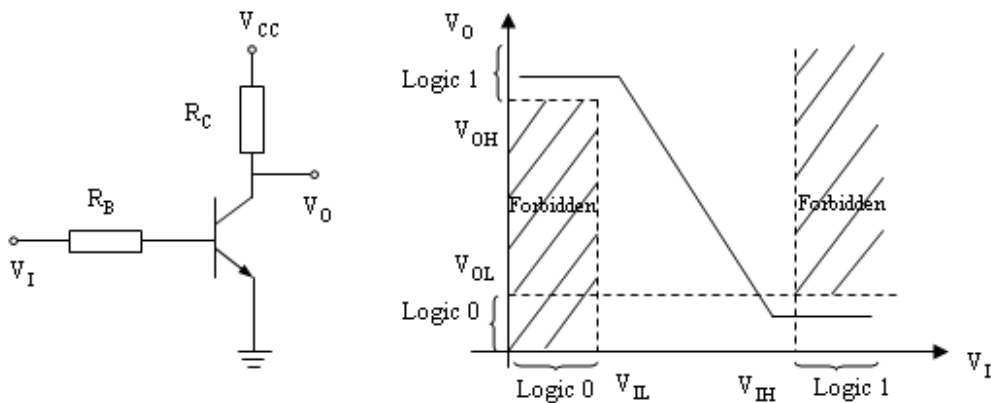


Figure 3.1 The transistor inverter and the input-output characteristics

The inverter specifications typically include the voltage levels [24]:

V_{IH} : minimum gate input voltage which will reliably be recognized as logic 1

V_{IL} : maximum gate input voltage which will reliably be recognized as logic 0

V_{OH} : minimum voltage at gate output when output is at logic 1 (HIGH)

V_{OL} : maximum voltage at gate output when output is at logic 0 (LOW)

TTL NAND gate

A possible design for a TTL NAND gate is shown in Figure 3.2 [24]. The logic values 0 and 1 are represented by the nominal voltage 0 V and + 5 V. The multi-emitter transistor input numbers can be increased.

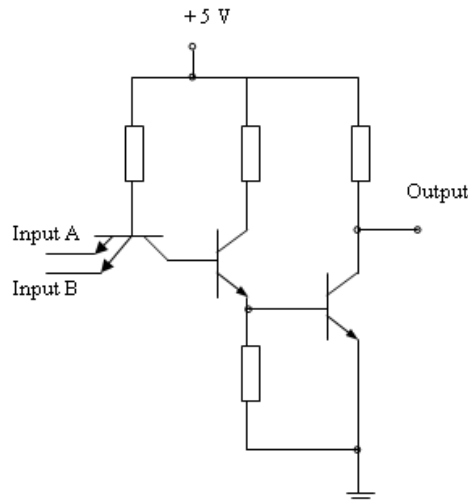


Figure 3.2 TTL NAND gate with resistor pull-up

CMOS Logic

The basic CMOS gates are considerably slower than the TTL gates but because of the simplicity of their geometry and very small physical size, they can be packed densely on a silicon chip [22].

Basic CMOS Gates

CMOS technology implements physically digital logic circuits using NAND, NOR, and NOT gates [22]. The CMOS inverter built up from an interconnection of pMOS and nMOS transistors may be considered as a basic switch circuit. Figure 3.3 presents the general structure of a CMOS circuit where the pMOS transistors have to be connected to +V_{cc} and the nMOS transistors to GND.

The circuit of the two-input NOR and NAND gates is also shown. For example, the NAND gate consists of two series-connected n-channel driver transistors and two parallel-connected p-channel load transistors.

In manufacturer's specifications the basic technology parameters characterizing digital logic gates are listed [22]:

- Fan-in: indicating the number of gate inputs (usually up to 4 or 5).

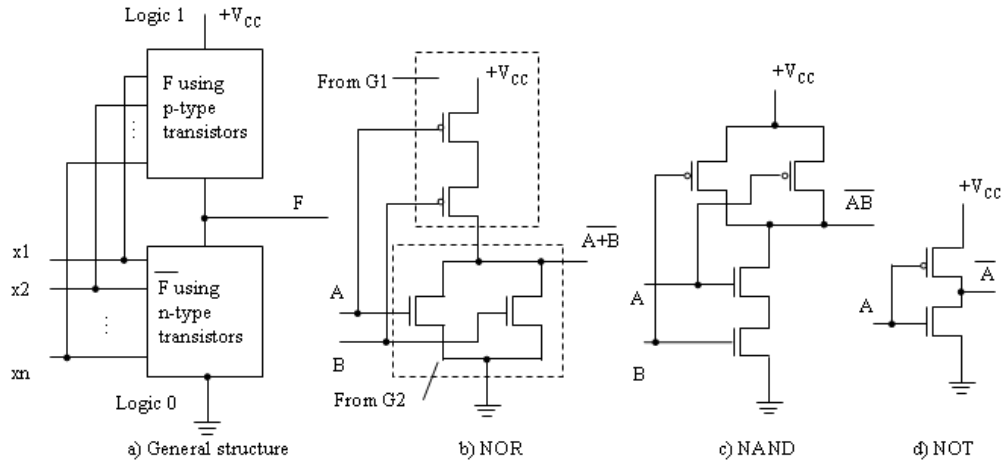


Figure 3.3 CMOS general structure and basic CMOS gates

- Fan-out: indicating the number of standard loads a gate's output can drive without reducing gate performance.
- Size: area on the silicon crystal occupied by the layout cell for the circuit corresponding to the gate. The area is proportional to the size of a transistor.
- Noise margin: absolute worst-case condition, constitutes the guaranteed margins against signal undershoot and power of thermal disturbances
- Power dissipation: power consumed by the gate (dissipated as heat)
- Logic voltage levels: important because the TTL and CMOS families input and output levels differ
- Propagation delay: time required for an input digital signal to be noticed at an output line.

]

Comparison of Logic Families

Transistor-transistor logic (TTL) based on bipolar transistors is one of the most widely used families for small- and medium-scale devices, and rarely used for VLSI. This family typically operates from 5V supply and has very good noise immunity. It is quite fast, especially in the Schottky version.

Emitter-coupled logic (ECL) based on bipolar transistors, removes storage time problems by keeping the transistors from saturation. ECL is by far, the fastest logic, but with low noise immunity.

Complementary metal oxide semiconductor (CMOS) is the most widely used family for large-scale devices. It combines high speed with low power consumption. Usually operates from a single supply of 5 - 15 V, has excellent noise immunity and can be connected to a large number of gates [25].

3.2 Sequential Logic Networks

In many applications the use of combinational circuits is not sufficient. Logic networks whose outputs depend not only on the actual input signal combination, but on the actual state of the network established previously is called sequential logic network. The network input variables are called **primary variables**, and the output variables which are fed back; **secondary variables**. The sequential circuits, in contrast to the combinational ones, have "memory". A sequential logic network, as a black box, is shown in Figure 3.4.

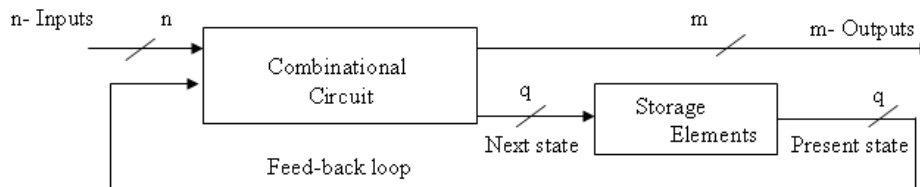


Figure 3.4 Sequential logic network

Sequential circuits can be classified into two groups: 1. Asynchronous sequential circuits (no clock signal), 2. Synchronous sequential circuits (operating with synchronizing/clock signal).

In the **asynchronous sequential circuits** the inherent time delay in the feedback loop will ensure the „memory” property necessary to generate the secondary variables. In this case the logic state transitions occur at different times, i.e. asynchronously.

In **synchronous sequential circuits** the operations are synchronized. This is the function of clock, which provides a series of pulses with precise pulse width and repetition rate. A synchronous sequential circuit, a clocked system, uses a clock to decide when to update the state of the circuit. Most sequential circuits are edge triggered: they change their state on either the rising or falling edge of the pulse. A transition from one state to the other occurs only at fixed

time intervals dictated by the clock pulse, giving synchronous operation. An idealized clock waveform is shown in Figure 3.5.

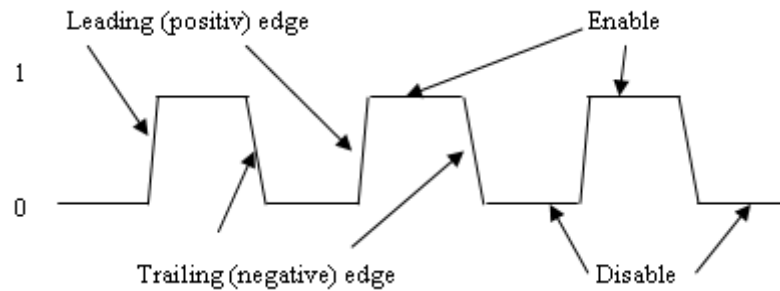


Figure 3.5 Clock waveform

One-bit storage element is able to hold a single bit, 0 or 1, to read the saved bit and to change its value. In sequential circuits the storage elements are: flip-flops and latches.

The **flip-flop**, as the basic memory element in sequential circuits, is itself a sequential circuit having two states. The one-bit storage device has several inputs (X), an output (Q), and a specific trigger input (clock - CLK). The output value depends on the response of a pulse at the trigger input CLK (on the *rising or falling edge* of the pulse). When a pulse is absent at input CLK the output remains unchanged (storage mode). The flip-flop two states correspond to logic-0 or logic-1 stored in the flip-flop. The flip-flop is **set** when the output has logic-1 value, and the flip-flop is **reset** when the output has logic-0 value.

A **latch** is a one-bit storage device with several inputs (X) and an output (Q). The output value is a function of the inputs $Q = f(X)$ only when specific combinations occur at the inputs X ; otherwise the output remains unchanged (storage mode). A latch can change state if there is an *active level* on the CLK input. Their content changes immediately when their inputs change.

Nowadays, most of the circuits are synchronous sequential circuits because the applications need predictable simple design and analysis. An asynchronous circuit is preferred over a synchronous circuit when high speed of operation is required. Asynchronous sequential circuits respond immediately whenever there is change in any input variable without having to wait for a clock pulse. The asynchronous sequential circuits cost less in terms of the number of gates than the synchronous circuits, and therefore, for economical reasons, they could find useful applications [16].

In the next we will give an overview of the most important flip-flop and latch types, and their implementation in the various logic families. The behaviour of

a particular flip-flop type will be described by truth/characteristic table and characteristic equation, which gives the next output in terms of the input control signals and the current output.

3.3 Flip-Flops

The SR Latch and Flip-Flop

The SR type is one of the simplest storage elements (bistable multivibrator) with two inputs S and R, which force the unit to become set and reset, and two outputs Q and \bar{Q} . The value of the next state Q (t + 1) depends on the values of the two present inputs S (set) and R (reset), and furthermore from the value of the present state Q (t). Figure 3.6 shows the symbols of SR latch and flip-flop used in logic circuits. The black boxes clocked symbols clearly distinguish the above mentioned two types, indicating that the latch is level-triggered (a, b) and the flip-flop is edge-triggered (c, d). Without the small triangle, the circuit is a latch. As in the case of logic gates, a bubble indicates the negation of a logic value. For latches a bubble in the clock line indicates that the clocked unit is enabled when the clock line is at logic-0, instead of logic-1.

A flip-flop can change state only during a transition of the trigger input Clk. For example a rising-edge triggered flip-flop can change its state only during 0-to-1 transition on clock pulse [26].

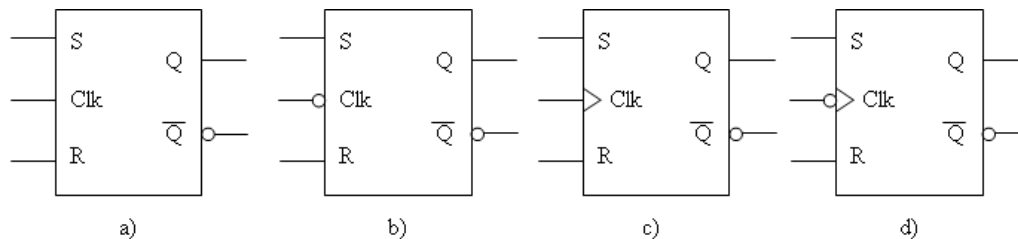


Figure 3.6 Symbols for SR latches and flip-flops

Figure 3.6 a) shows the standard symbol for logic-1 active level SR latch (latch can change state if Clk = logic-1). The standard symbol for logic-0 active level SR latch is illustrated in Figure 3.6 b). The symbol used for rising-edge triggered SR flip-flop (flip-flop can change state only during 0-to-1 transition on Clk) is shown in Figure 3.6 c). Finally, Figure 3.6 d) illustrates a falling-edge triggered SR flip-flop (flip-flop can change state only during 1-to-0

transition on Clk). The SR latch function and characteristic table, a relationship which exists between the inputs, outputs, present states and next states is described by Table 3.1. If the clock line is set to logic-1, the state of the SR latch becomes logic-0 when logic-0 is placed on R but not S, and the state of the SR latch becomes logic-1 when logic-1 is placed on S but not R. No change in state occurs when $S = R = 0$, and the SR latch behaviour is not defined for the $S = R = 1$ values.

Function table					Characteristic table				
Clk	S(t)	R(t)	Q(t)	Q(t+1)	Operation	S(t)	R(t)	Q(t)	Q(t+1)
1	0	0	0	0	No change	0	0	0	0
1	0	0	1	1		0	0	1	1
1	0	1	0	0	Reset	0	1	0	0
1	0	1	1	0		0	1	1	0
1	1	0	0	1	Set	1	0	0	1
1	1	0	1	1		1	0	1	1
1	1	1	0	X	Undefined	1	1	0	X
1	1	1	1	X		1	1	1	X
0	X	X	X	Q(t)	No change				

Table 3.1 Behaviour of an SR latch

In the SR flip-flop function table instead of $\text{Clk} = 1$ appears the rising clock edge (\uparrow), and $\text{Clk} = 0$ is substituted by the falling clock edge (\downarrow). The characteristic table is the same. The SR latch and flip-flop characteristic equation is derived from the characteristic table. Their expression is obtained by using a three variable Karnaugh-map (notations correspond to Figure 1.6 c) for the inputs $S(t)$, $R(t)$ and $Q(t)$, see Figure 3.7.

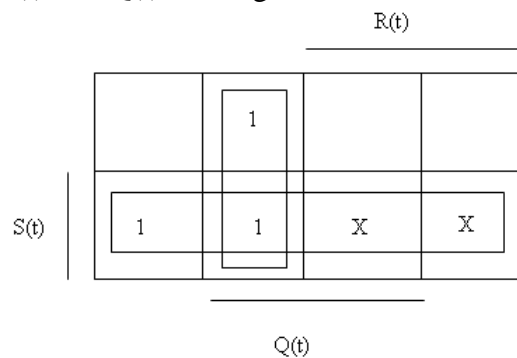


Figure 3.7 $S(t)$, $R(t)$ and $Q(t)$, variable Karnaugh-map

From the above Karnaugh-map we can deduce the characteristic equation of a SR latch or flip-flop: $Q(t+1) = S(t) + \overline{R(t)} Q(t)$

The so called excitation table is similar to the truth table, showing the input variable states that are necessary to generate a particular next state when the current state is known. SR latch and flip-flop has the same excitation table, see Table 3.2.

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

Table 3.2 SR latch and flip-flop excitation table

The SR latch and flip-flop can also be represented graphically by a state diagram, derived from the excitation table. In the state diagram a state is represented by a circle, and the transition between them is indicated by arrows, representing the path between different circles. SR latch and flip-flop has the same state diagram, see Figure 3.8.

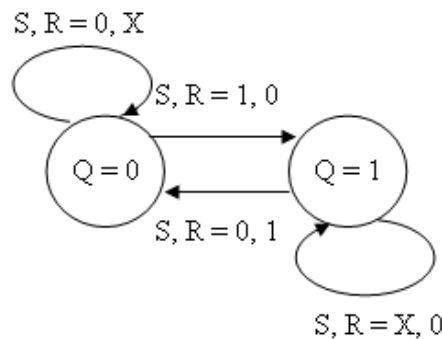


Figure 3.8 SR latch and flip-flop state diagram

Starting from the characteristic equation $Q(t+1) = S(t) + \overline{R(t)} Q(t)$, from a simple AND-OR gate implementation, making the transformations according to DeMorgan's law and rearranging/redrawing the circuit (see Figures 3.9 a and b), we deduce the corresponding two level NAND structure (see Figure 3.9 c).

This is the so called $\bar{S} \bar{R}$ latch design. By using NOR gates instead of NANDs we get the S R latch structure (Figure 3.9 d).

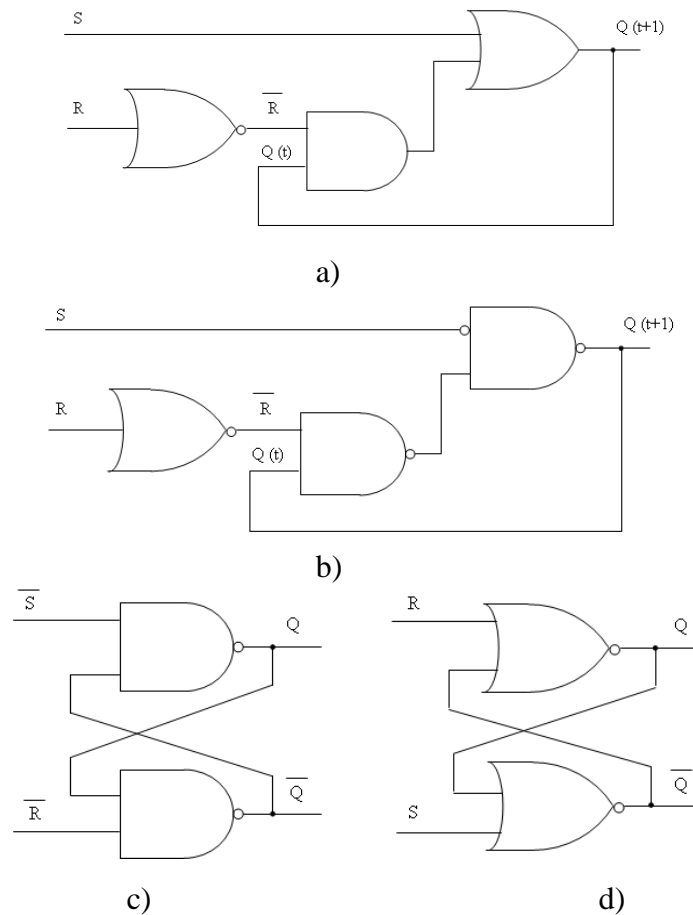


Figure 3.9 $\bar{S} \bar{R}$ and S R latch design using logic gates

In this approach SR latch is a simple *asynchronous sequential circuit*, built up from gates with feedback loops. This type can be use as a basic unit in every synchronous and asynchronous sequential circuit.

Clocked Latch and Flip-Flop

The latches and flip-flops often used for implementing *synchronous sequential circuits* have a clock input. A clocked SR latch or flip-flop can be performed from a simple structure NAND (Figure 3.8 c) by adding two more NAND gates as shown in Figure 3.10. The additional gates generate the \bar{S} and \bar{R} signals, based on inputs S and R and Clk. In case of SR latch the Clk input works as a control input which acts just like an enable.

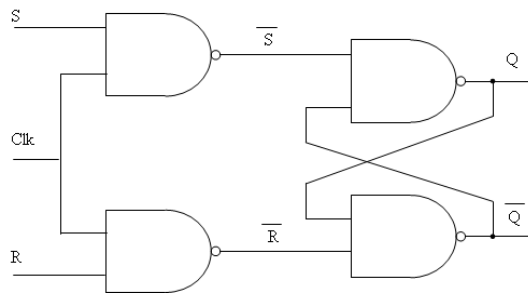


Figure 3.10 Clocked SR Latch

The latch is level triggered and is called “transparent” because any change on the inputs is seen at the outputs immediately. This property causes synchronization problems and this unit is not recommended for use in synchronous design. In the applications various flip-flops are used which are created using latches. The flip-flop output respond to the inputs on specific times, the rising or falling edge of a clock signal.

JK, D and T Flip-Flops

In this section we will study the JK (Jack Kilby), D (data) and T (toggle) flip-flops (omitting the JK and T latches), emphasizing the popular D flip-flop design using latches. In general, the latches and flip-flops have the same characteristic table, characteristic equation, excitation table, and state diagram. The rising-edge triggered flip-flops standard symbols are presented in Figure 3.11. The JK flip-flop is a generalized form of the SR flip-flop. The J input has the set function, and the K the reset one. When $J = K = 1$ at the Q output appears the complemented value of the present state.

The D flip-flop has the simplest relationship between its next state and the input line. Its function is to copy the data on the D input to the Q output line at the next clock pulse. The signal is delayed by exactly one clock period.

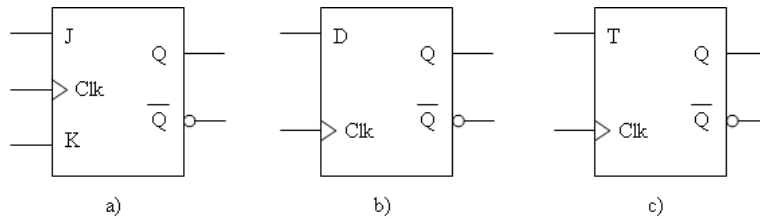


Figure 3.11 Standard symbols for JK, D and T flip-flops

If T flip-flop input line has logic-1 value, the value of the next state becomes the complement of the previous state; otherwise its state remains unchanged. Table 3.3 summarizes the above mentioned three flip-flops truth/characteristic tables.

J(t)	K(t)	Q(t)	Q(t+1)
0	0	0	0 No change
0	0	1	1
0	1	0	0 Reset
0	1	1	0
1	0	0	1 Set
1	0	1	1
1	1	0	1 Complement
1	1	1	0

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

T(t)	Q(t)	Q(t+1)
0	0	0 No change
0	1	1
1	0	1 Complement
1	1	0

a) b) c)

Table 3.3 JK, D and T flip-flops truth table

The various flip-flops characteristic equation can be derived from the corresponding three and two-variable Karnaugh maps (notations correspond to Figure 1.6 c) shown in Figure 3.12.. The map cells follow the flip-flop truth tables.

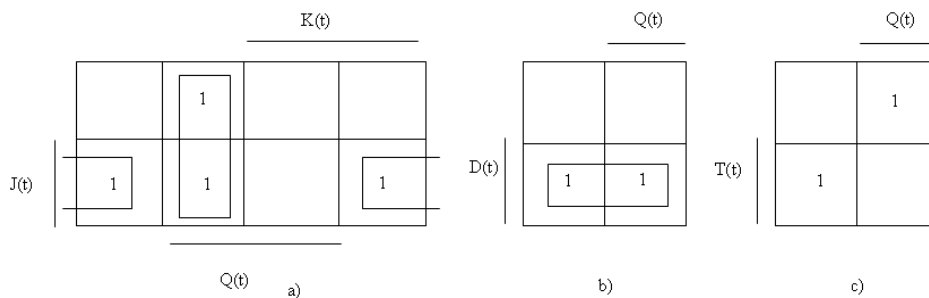


Figure 3.12 JK, D and T flip-flops Karnaugh maps

The JK, D and T flip-flops characteristic equations are:

$$Q_{JK}(t+1) = J(t) \overline{Q(t)} + \overline{K(t)} Q(t)$$

$$Q_D(t+1) = D(t)$$

$$Q_T(t+1) = T(t) \oplus Q(t)$$

From the JK, D and T flip-flops truth table we can deduce their excitation tables and state diagrams, as shown in Figure 3.13.

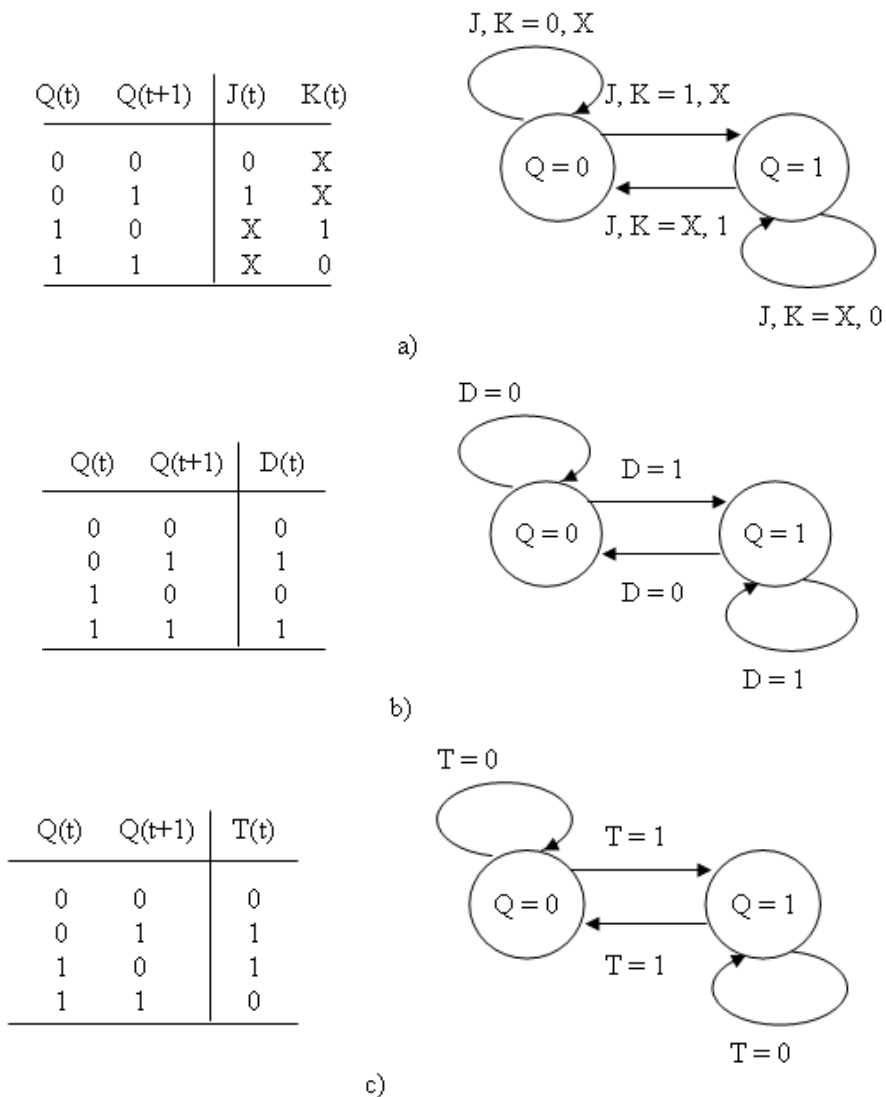


Figure 3.13 JK, D and T flip-flops excitation table and state diagram

Asynchronous Set/Reset Flip-Flops

In many applications there is a need for asynchronously SR, JK, D or T flip-flops. These units have two additional inputs as:

- asynchronous set (S) is called direct set or Preset
- asynchronous reset (R) is called direct reset or Clear

The clock pulse controls all inputs excepting S and R, which have logic-0 active level. The standard graphics symbols are shown in Figure 3.14.

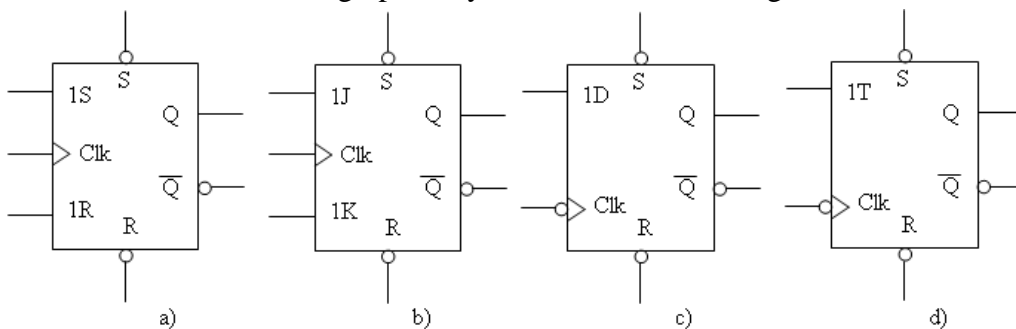


Figure 3.14 Examples of standard graphics symbols

The characteristic table, characteristic equation, excitation table, and state diagram are the same as for the normal flip-flops.

SR Flip-Flop Design using Latches, Master-Slave Flip-Flop

In many applications it is necessary to observe the flip-flop state while in parallel a new state is entered. In this case the new flip-flop state may be logically dependent on itself. To avoid the possibility of an oscillation which could be caused by the flip-flop continuously state-changing during the period in which the clock is equal to logic-1, it is useful to introduce an intermediate network. As an example, consider an SR flip-flop using latches as shown in Figure 3.15. The new structure, called Master-Slave is composed of two main units, called master, which is an SR latch, and slave, another SR latch. Because of the particular interconnection the SR flip-flop's clock input enables either the first or the second SR latch, but not both. For example when $\text{Clk} = 0$, the master SR latch is enabled. The structure presented above is a rising edge-triggered flip-flop. The output Q depends on the flip-flop input value that was present right at the rising edge of the clock signal. Like SR latches, SR flip-flops are useful in control applications.

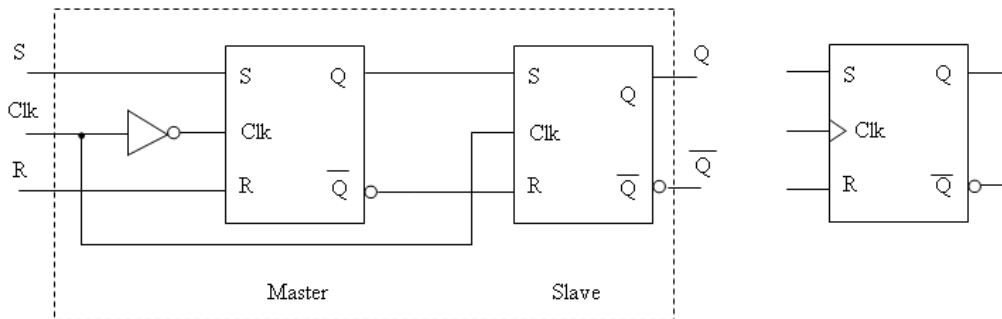
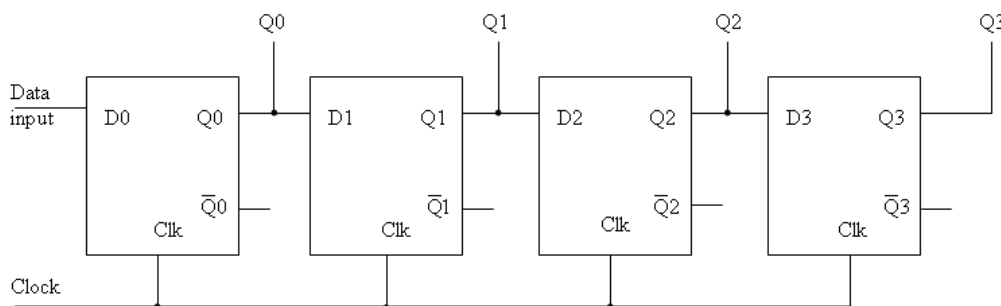


Figure 3.15 An SR master-slave flip-flop

3.4 Registers and Counters

One of the most important sequential circuit used in digital system is the **register**. Its main role is to store and/or shift the input data (bits). A register is built up from flip-flops connected in cascade.

This unit allows serial and/or parallel shift into or out of the register, and right or left data movement. An N-bit register requires N flip-flops. The registers are typically built up from D flip-flops. Figure 3.16 shows a basic 4-bit serial-in, parallel-out, right shift register logic diagram and its waveforms. In order to illustrate its operation we consider the data word 1011 as the input (waveform is illustrated in Figure 3.16 b) [22]. The register is initially cleared, at $t = 0$ $Q_0 - Q_3$ outputs are all 0. At $t = 1$, on the positive edge of the clock pulse (CP1) the first bit appears at the output Q_0 . At the second clock pulse, the bit at Q_0 is transferred to Q_1 while the next data bit appears at Q_0 , based on D flip-flop truth table (see subsection 3.3). Next clock pulse causes the bit at Q_1 to appear at Q_2 , the bit at Q_0 to appear at Q_1 and so on.



a)

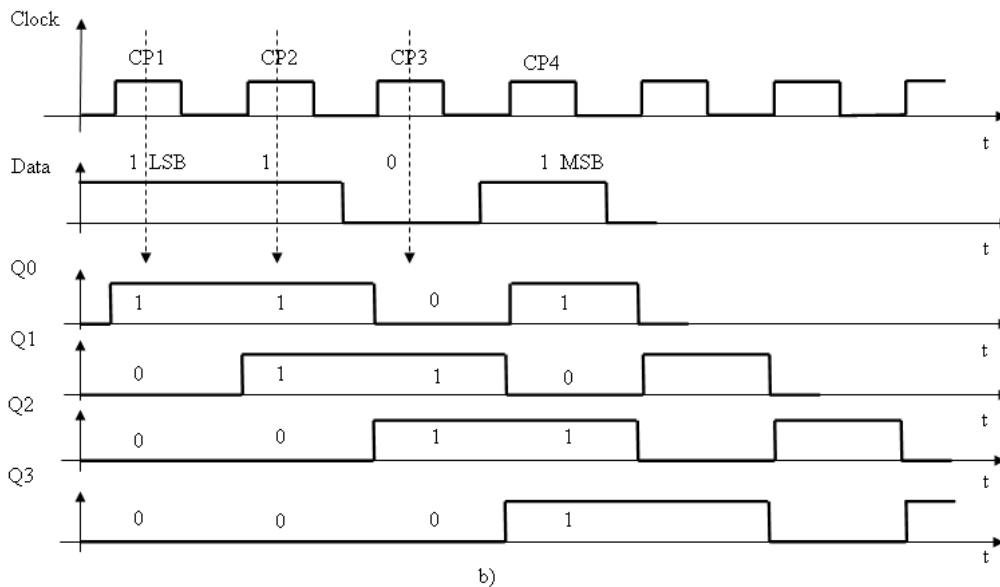


Figure 3.16 4-bit shift register a) logic diagram; b) waveforms

Counters

An N-bit **counter** is an extended N-bit register able to increment or decrement its own value on each clock cycle (when the count is enabled) [20]. A counter that can increment (means to add 1) is called up-counter. The down-counter can decrement (means to subtract 1). The up/down counter can increment and decrement. The counters having a repeated state sequence are called **modulus counters**. The modulus refers to the number of different states that make up the counting sequence. For example, a binary coded decimal counter is used to count from 0 to 9 (10 different states) is a mod-10 counter. A counter with N flip-flops has a maximum of 2^N states (maximum modulus is 2^N). The counters are typically built up from JK, T or D flip-flops.

Counters are classified into two categories [27]:

- Asynchronous Counters (Ripple counters)
- Synchronous Counters

Asynchronous Counters

The Ripple Counter

A three-stage (modulo - 2^3) up-counter using JK flip-flops is shown in Figure 3.17. The flip-flops are connected to toggle, and change state during 1-to-0 transition on clock. The input signal, whose pulses are to be counted, is applied to the clock input of the first flip-flop, and the output of each flip-flop is connected directly to the clock input of the next. The flip-flop clear inputs are connected together, before counting starts; a pulse can clear all the flip-flops. During counting a delay is caused by the **rippling**, which results in a limitation of the maximum frequency of the input signal. The ripple counters could be down-counters as well.

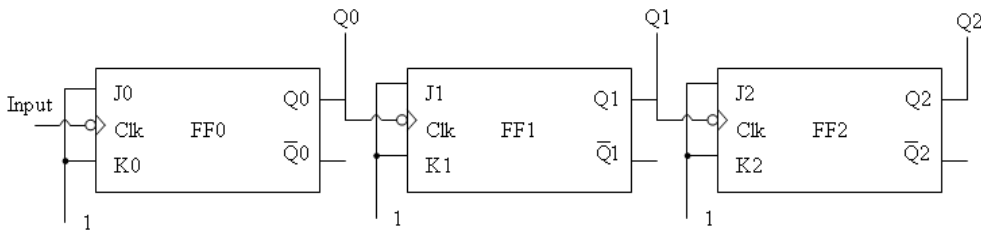


Figure 3.17 Mod-8 ripple counter

Divide-by-N Counter

A mod-N counter may also be described as a divide-by-N counter [27]. The circuit shown in Figure 3.18 acts as frequency divider. In general, in an N flip-flop circuit, the input frequency is divided by 2^N in steps of 2. The negated output of the first flip-flop is directly connected to the clock input of the second one, which is the most significant flip-flop. The Q0, Q1 output values depend on the response of a pulse at the trigger input clock (on the rising edge of the pulse). The FF0 output frequency is equal to a half of the main clock frequency, and the FF1 output's is quarter of it. Thus, this is an example of a divide-by-4 counter.

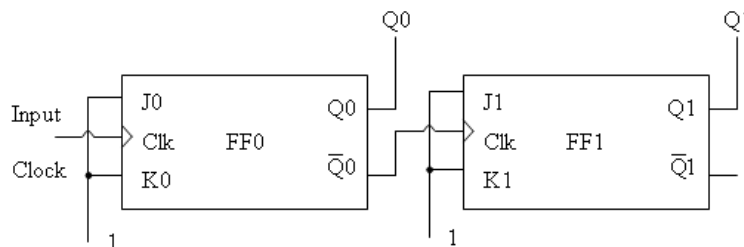


Figure 3.18 Divide-by-4 counter

Synchronous Counters

Ring Counter

In synchronous counters, the clock input is connected to all of the flip-flops so that they are clocked simultaneously. A 4-bit ring counter using D flip-flops shown in Figure 3.19 is very similar to the 4-bit shift register (Figure 3.16). The **ring counter** Q3 output is fed back to D0 input. If $Q0 = 1$ while $Q1 = Q2 = Q3 = 0$, each clock pulse shifts the 1, first to Q1, then to Q2, Q3, and finally back to Q0, as a ring. Since the ring counter is composed from four flip-flops, and has four distinct states, is called a modulo-4 counter. Each flip-flop output frequency is equal to a quarter ($1/4$) of the main clock frequency.

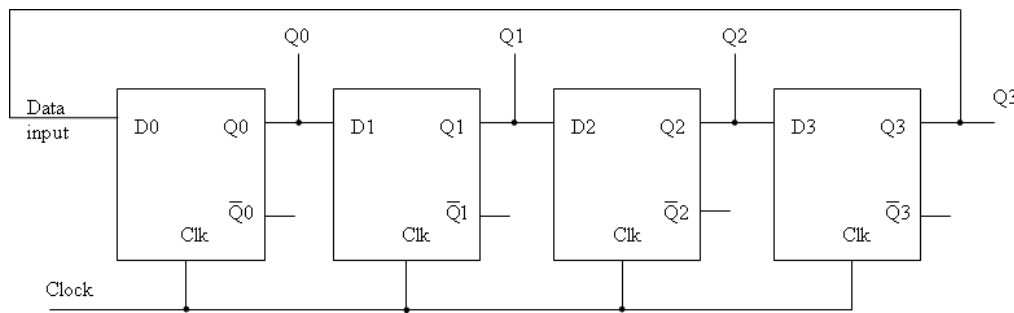


Figure 3.19 Ring counter

A ring counter has many applications, as: frequency divider, code generator, counter, period and sequence generator etc. [28].

Johnson Counter

The **Johnson counter** is very similar to the 4-bit ring counter shown in Figure 3.19, having in the feed back loop an inverter, or the inverted output Q3 ($\bar{Q3}$) of the last flip-flop is connected to the input of the first flip-flop. An N-stage Johnson counter has $2N$ different states, and it is considered as a modulo- $2N$ counter. The Johnson counter is economical, it can be implemented with only half the number of flip-flops.

Case Study 1: Synchronous Modulo-3 Counter

A **modulo-3 counter** [22] has only three different states, and to cover them we need only two flip-flops (generating 2^2 different states). Table 3.4 lists the counter states (the **1 1** state is omitted), the two JK flip-flop circuit and its waveforms is shown in Figure 3.20.

Counter state	Q1 (2^1)	Q0 (2^0)
0	0	0
1	0	1
2	1	0
0	0	0
1	0	1
2	1	0
0	0	0

Table 3.4 State table

The flip-flops are initially reset ($Q_1Q_0 = 00$) [22]. Then $\overline{Q_1} = 1$ and $J_0K_0 = 11$, FF0 is set by the first clock pulse. Now $Q_1Q_0 = 01$.

Since $J_0K_0 = 11$, the next clock pulse will reset FF0, making $Q_0 = 0$ once more. Since $Q_0 = 1$ before the second clock pulse arrived, FF1 is set by the second pulse and Q_1 becomes 1.

With $Q_1Q_0 = 10$, $J_0 = 0$ while $K_0 = 1$. After the third clock pulse, $Q_1Q_0 = 00$, which was the initial state.

In general, the design of a synchronous counter (sequential logic circuit) requires the design of a combinational logic circuit and a memory unit, composed from flip-flops.

The counter design starts with develop of the state diagram and the next-state table for a specific counter sequence. Based on the flip-flop transition (excitation) table the Karnaugh-maps are used to derive the logic expressions for flip-flop inputs. Finally, the counter implementation is given [29].

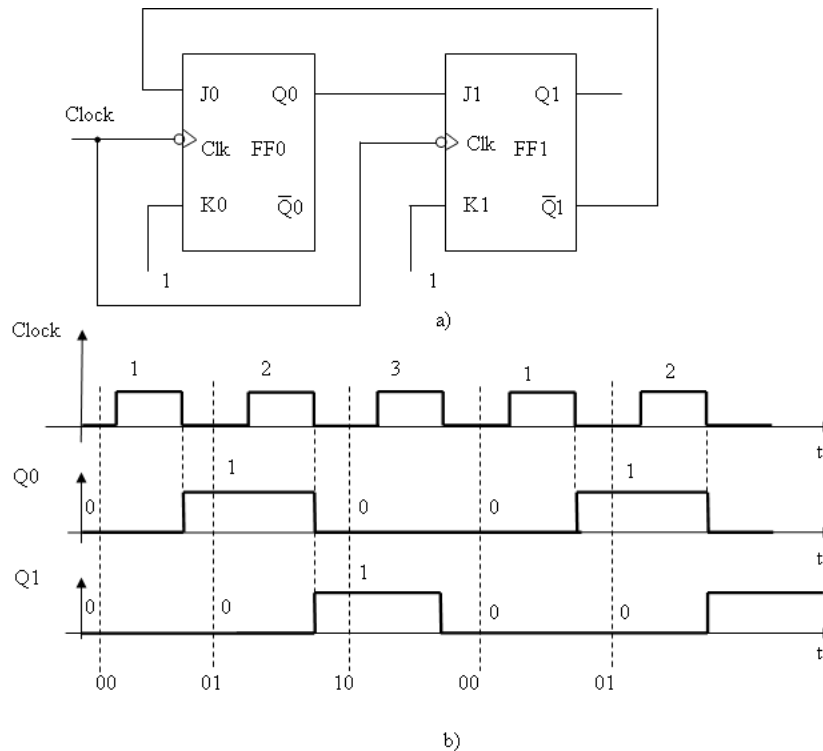


Figure 3.20 Mod-3 counter a) circuit; b) waveform

Case Study 2: Design the 3-bit Gray code counter using JK flip-flops

The Gray code is a binary representation for positive integers having a sequence with a special property (see Table 2.4). In the next the design of the 3-bit Gray code counter is presented. Figure 3.21 shows its state diagram and next state table.

Based on JK flip-flop excitation table (see Figure 3.12 a) Figure 3.22 shows the Karnaugh maps for present-state J and K inputs and the logic expressions for the flip-flop inputs [29].

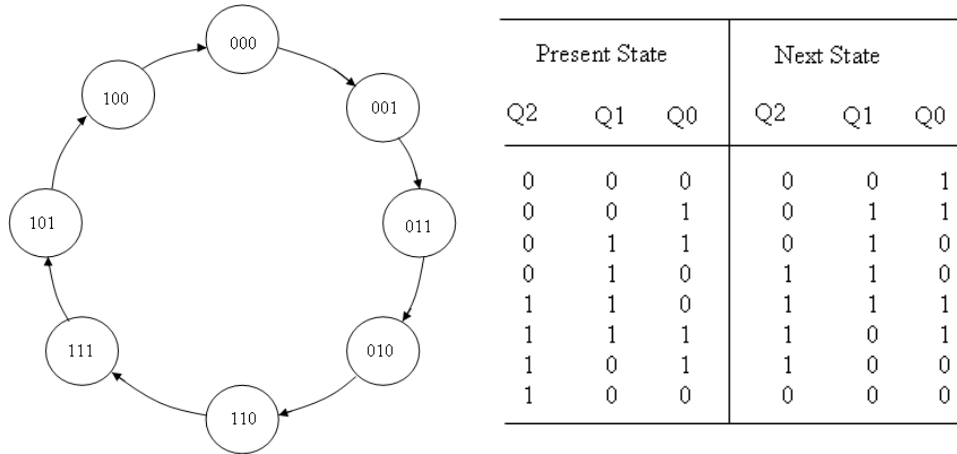


Figure 3.21 State diagram and next state table for a 3-bit Gray code counter

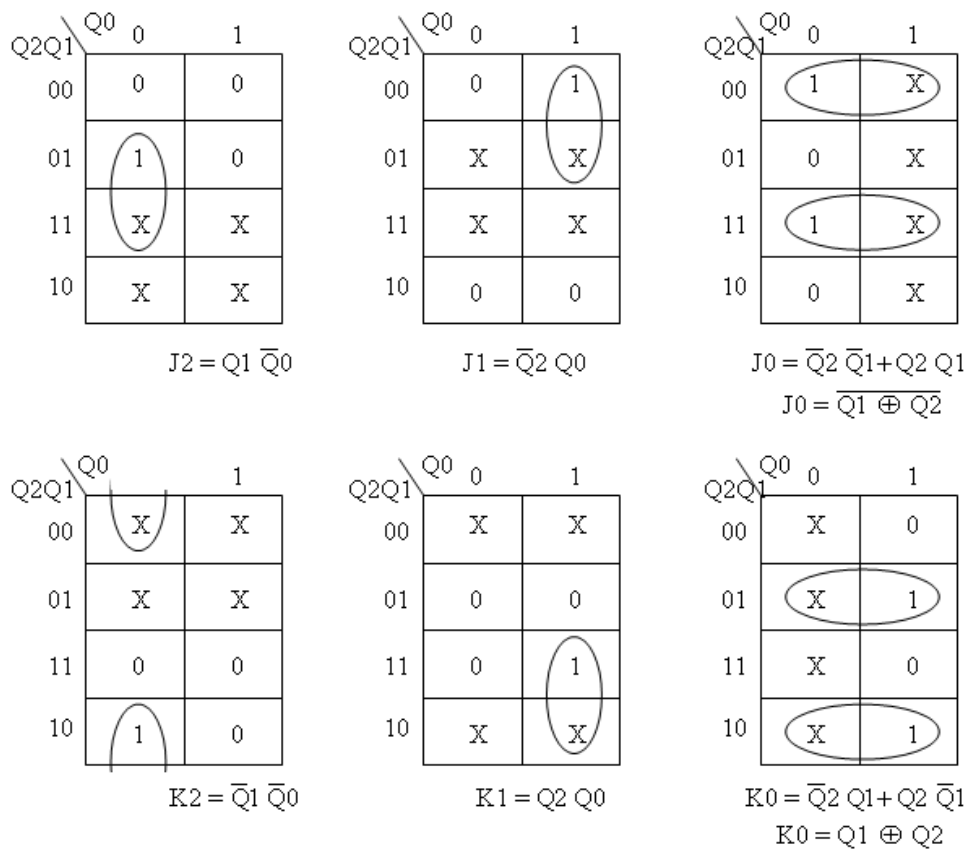


Figure 3.22 Karnaugh maps and the next-state J and K outputs expressions

Finally, Figure 3.23 shows the 3-bit Gray code counter implementation. Digital counters are very useful in many applications. They can be easily found in digital clocks and parallel-to-serial data conversion (multiplexing) [30]. The asynchronous counters logic circuit is very simple, while the design of a synchronous counter involves complex logic circuit. The asynchronous has low speed in comparison with the synchronous design, where all flip-flops are clocked simultaneously.

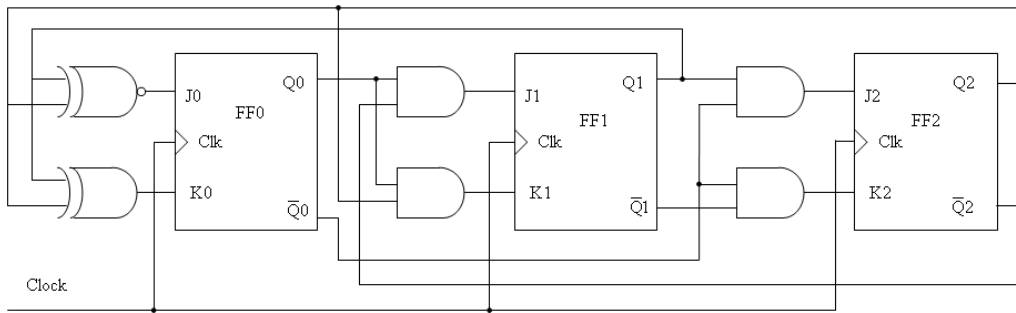
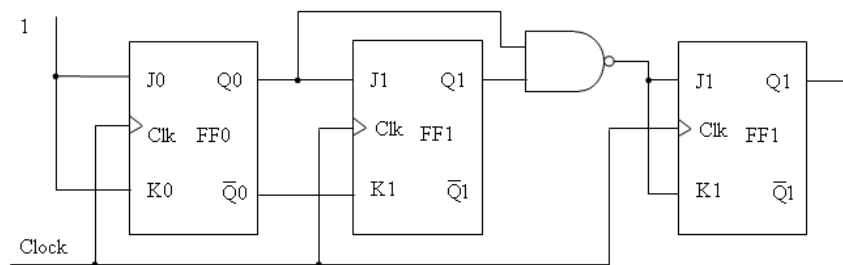


Figure 3.23 The hardware diagram of the 3-bit Gray code counter

Problems

- 3.1 Draw the schematic of a four-input CMOS NOR and NAND gate
- 3.2 Draw the schematic of the logic function $F = A + B \overline{(F + G)}$ from CMOS gates
- 3.3 In the next circuit the sketch Q1 and Q2 (time delay negligible) waveforms



- 3.4 Using four D flip-flops, design a 4-bit register which can be used for parallel-in parallel-out data transfer. Show the four input and four output waveforms when the data word 1110 is transferred
- 3.5 How many flip-flops are required to count to
a) 8 b) 28 c) 67 d) 124?
- 3.6 What is the maximum modulus for a counter which contains
a) 3 b) 6 c) 8 d) 10 flip-flops?
- 3.7 Design a four stage counter that has six states: 0000; 0110; 0111; 1100; 0001 and 1101

Chapter 4

Semiconductor Memories and Their Properties

All sequential circuits have a memory property which is due to the use of basic memory units, such as flip-flops. In this chapter we will overview the main memory types, how they are built, their main features, and how they work.

From the memory data-holding point of view the memories are classified in two main groups: as **volatile** and **nonvolatile** types. Volatile memory is also called **RAM** (random access memory), while nonvolatile is called **ROM** (read only memory). The RAM can be written to and read from, and the ROM can only be read from. We speak about a $M \times N$ memory capacity if the memory component is able to store M data items of N bit each. A **word** represents each data item in a memory. Words can be read one by one and written using address inputs.

4.1 Volatile Memories

In this subsection we will discuss RAM memory in general presenting the SRAM, DRAM, and CAM memory types. The volatile memories criterion is that they store their information as long as the power supply is on. A RAM is a kind of memory whose contents can be easily modified. Writing (storing) data into a RAM chip is as fast as reading data. Figure 3.1 shows a block diagram for a 1024×32 RAM ($M = 1024$, $N = 32$). **Data** is a 32-bit wide set of data lines used as input lines during writes or as output during reads. **ADRS** is a 10-bit ($2^{10} = 1024$) input serves as address line during reads or writes. **RW** is a control input which value is 0 for read operation and $RW = 1$ for write. **EN** is a 1-bit control input that enables the RAM for reading or writing [20].

Figure 4.2 shows the logical internal structure of an $M \times N$ RAM which is an array of bit storage blocks, known as **cells**. A collection of N cells forms a word, and there are M words. The address inputs arrive into a decoder selecting all the cells in one word corresponding to the present address values. The **EN** and **RW** inputs role are the same as above. The data lines are connected through one word's cell to the next word's cell.

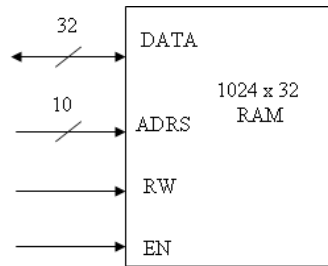


Figure 4.1 1024 x 32 RAM black box

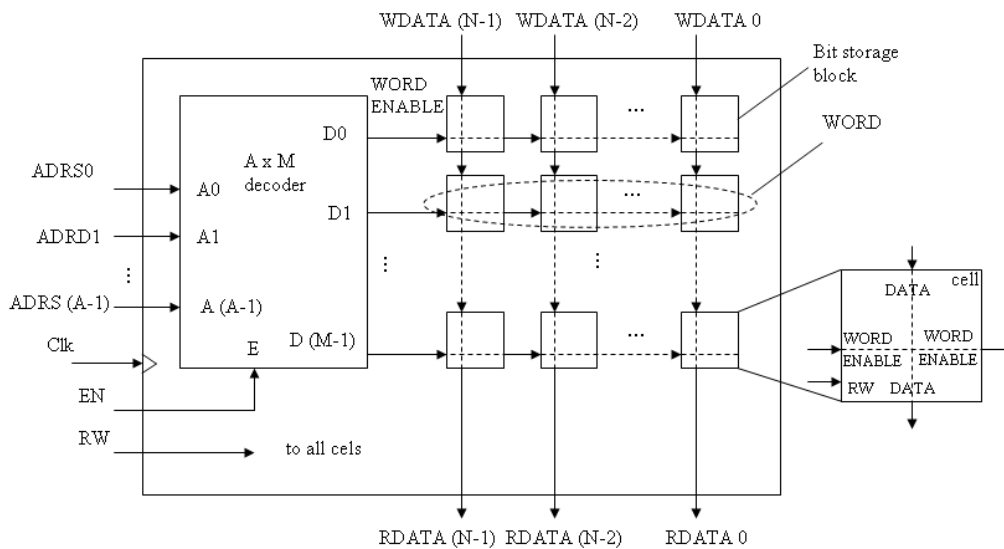


Figure 4.2 Logical internal structure of a RAM

Static Random Access Memory (SRAM)

SRAM is one of the most known temporary memory types. A static RAM could be modeled as two inverters connected in a loop as is shown in Figure 4.3 [20].

A D bit storing procedure is simple; the bit will go through the bottom inverter resulting in a \bar{D} value, then back through the top inverter to become D again.

To **write** a logic-1 in a RAM cell it is required to set DATA line to 1, and $\overline{\text{DATA}} = 0$ value. To **read** fast a stored bit it is recommended to set both

DATA and $\overline{\text{DATA}}$ lines to logic-1 (so called precharging), and then by having ENABLE to logic-1. Each memory has a specific timing diagram that specifies the correct time sequence of the events.

The Figure 4.3 shows a common SRAM cell implementation, the so-called 6 transistor cell. The structure consists of two cross-coupled CMOS inverters plus two access nMOS transistors responsible for connecting the inverters to the input/output bit lines when the corresponding word line is asserted.

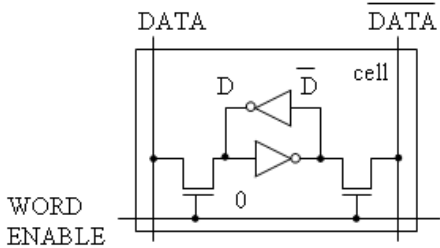


Figure 4.3 SRAM cell

Dynamic Random Access Memory (DRAM)

The inexpensive and reasonably high-speed DRAM memory is a single transistor memory cell array. A DRAM cell consist of one transistor, which is used as a switch to allow a charge to be moved into or out of the second component, a capacitor, as is shown in Figure 4.4 .

To **w**rite data into a DRAM cell the enable input value has to be logic-1. The data line values (logic-1 or 0) occur the charging or putting on to ground level the top plate of the capacitor. When ENABLE is returned to 0, the transistor is turned off; the charge is trapped in the capacitor and ideally cannot change until the enable will be 1 again. It is very important to select a relatively large capacitor, to lengthen its discharging time.

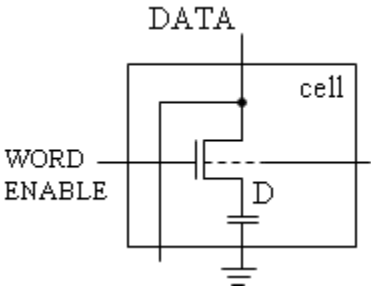


Figure 4.4 DRAM cell

To **read** a stored bit the DATA line voltage must be set to the midway between 0 and V_{cc} , and the $\overline{ENABLE} = 1$.

The value stored in the capacitor will modify the data line voltage level. A special circuit is able to detect the changing and amplifies it to logic-1 or 0. Since the reading discharges the capacitor, the RAM must immediately write the bit read back to the bit storage cell. This operation is performed automatically by a memory controller. A built-in memory controller is vital to perform a refreshing step in every few microseconds. The RAM must be refreshed periodically (e.g. 64 ms refresh interval) because information is stored onto capacitors which can lose their charge. DRAM is slower than the SRAM because every read must be followed by an automatic write.

In a DRAM memory chip, the cells are arranged in rows and columns, as shown in Figure 4.5. In this particular case, the array size is 256 rows by 256 columns of 4-bit words. The row and column addresses (a total of 16 bits) are multiplexed. The memory array is controlled by the signals (1)–(6) generated by the timing and control unit after processing \overline{OE} (output enable), \overline{WE} (write enable), \overline{RAS} (row address strobe), and \overline{CAS} (column address strobe). As in the SRAM chip, data are obviously maintained while in standby mode [31].

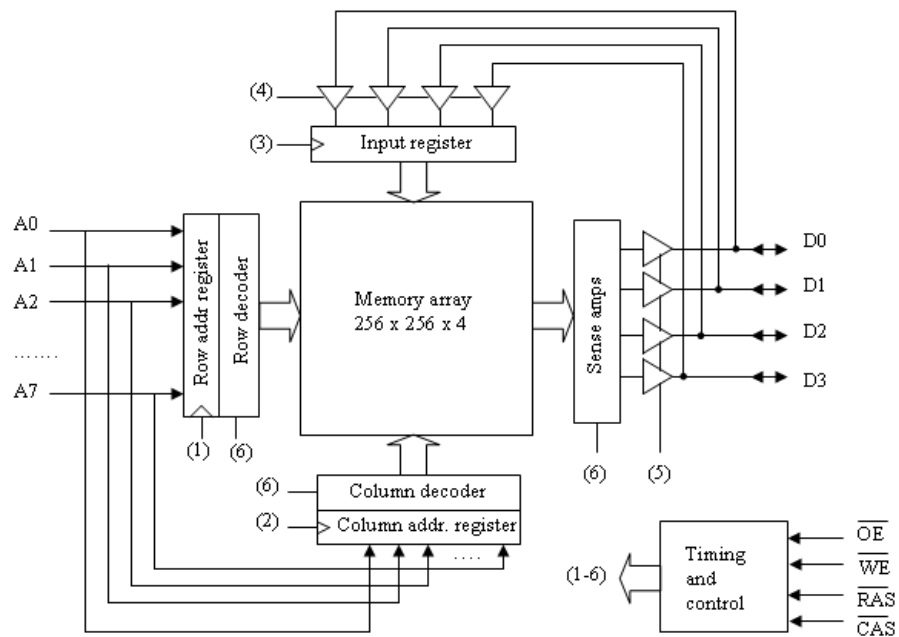


Figure 4.5 Simplified view of a conventional DRAM chip

Content-Addressable Memory (CAM) for Cache Memories

While in the cases of the above memory types during the reading operation an address is selected, to which the memory responds with the value stored in that address, in a CAM instead of an address, a **content** is presented. In this approach the memory responds with a hit if such content is stored in the memory. This type of memory, also called **associative memory**, is used in applications where performing a “match” operation is necessary.

4.2 Nonvolatile Memories

In this subsection we will discuss ROM memory in general presenting the PROM, EPROM, and EEPROM memory types.

The read-only memory, or ROM, is a special kind of memory which does not lose its contents when power is shut off. A ROM reads faster and consumes less power than a RAM [20]. The ROMs are applied in such of systems where it is important to store programs that should not be modified. Examples: arithmetic circuits might use tables to speed up computations of logarithms or divisions or in many computers the BIOS system, etc. Figure 4.6 shows the black box of a 1024 x 32 ROM. The ROMs internal structure is very similar to RAM architecture (see Figure 4.2). Since this memory can be read from but not written to, the WR and WDATA inputs are not needed. The ROM does not have a clock input because no synchronous writes occur in a ROM. From this reason a ROM is like a combinational circuit where the inputs are the address lines, and it produces some data as the output.

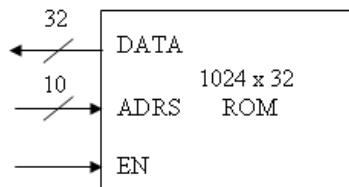


Figure 4.6 1024 x 32 ROM black box

In the next two popular methods are described to understand how bit storage is implemented.

Mask-Programmed ROM

Figure 4.7 illustrates mask-programmed ROM cells which are programmed during fabrication. The left cell is programmed with 1 by directly wiring logic-1. The right cell is programmed with 0. Mask-programmed ROM has the best compactness of any ROM type.

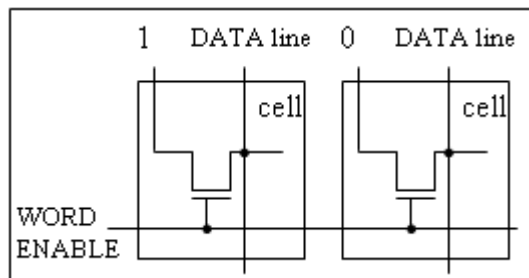


Figure 4.7 Mask programmed ROM cells

Fuse-Based Programmable ROM - PROM

This ROM type has fuses in each bit storage cell as is shown in Figure 4.8. In the initial phase, the ROM is manufactured with intact fuses; all stored contents are logic-1.

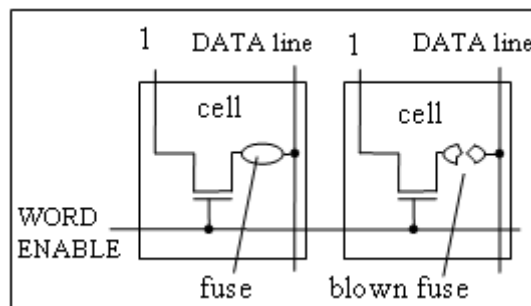


Figure 4.8 Fuse-based ROM cells

If the fuse is intact, like a wire, the cell logic-1 contents is enabled which appears on the data line. A special device, a programmer is able to set the ROM contents. By passing a higher than normal current through the fuse, the connection is eliminated, the fuse is blown, and a logic-0 is occurred. The ROM which can be programmed (written) only once is called one time programmable ROM or PROM.

Erasable PROM - EPROM

Most of the commercial reprogrammable ROMs are based on floating-gate transistors. Figure 4.9 depicts a logical view of an erasable PROM cell. Each cell consists of a special type of transistor with a particular gate in which the electrons are captured. At the beginning an EPROM cell stores a logic-1 value, see Figure 4.9 left cell. When a programmer device applies higher than normal voltage to those transistors in cells that should store 0s, in the floating gate transistors appear the trapped electrons, see Figure 4.9 right cell.

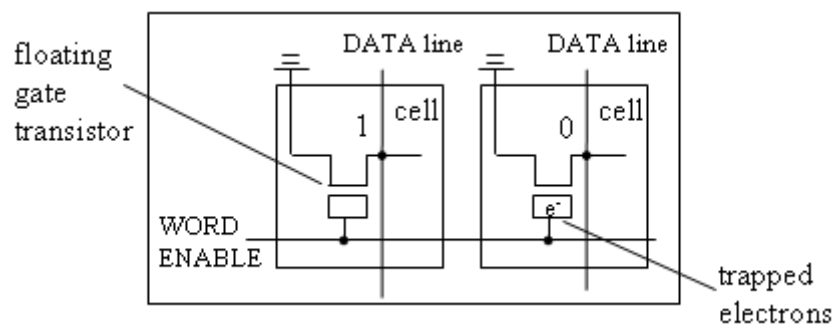


Figure 4.9 EPROM cells

Exposing an EPROM chip to ultraviolet (UV) light of a particular wavelength causes all the stored 0s (charges) to disappear (erase) after which the memory can be programmed again. This kind of chip can typically be erased and reprogrammed about ten thousand times or more and can retain its contents without power for ten years or more [20]. Usually, an EPROM chip has a window in the package through which UV light can pass.

EEPROM and Flash Memory

EEPROM solves the erasure problem of EPROM with a slight modification in the floating-gate transistor. To program an electrically erasable PROM (EEPROM), a high voltage has to be applied in order to trap the electrons in the floating gate transistor, and another high voltage must be used to free the electrons. Figure 4.10 shows a black box of an EEPROM.

The data lines are bidirectional. Because EEPROMs use voltages for erasing, instead of UV light, it is possible to erase and reprogram certain words without changing the contents of other words. If EN = 1, while WRITE = 1 indicates that the data on the data lines should be programmed into the word at the

address specified by the address line. The BUSY input is responsible to show that programming is not complete.

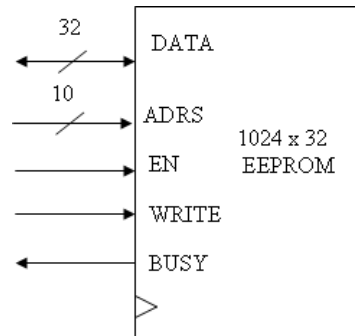


Figure 4.10 1024 x 32 EEPROM black box

Modern EEPROMs can be programmed tens of thousands to millions of times, and can retain their contents for tens to one hundred years or more without power [20].

Flash memory (also called Flash EEPROM) is a combination of EPROM (requires only one transistor per cell) with EEPROM (electrically erasable and electrically programmable). A flash memory can erase very quickly a memory block, sector or the whole memory.

The next generation memories should be nonvolatile, with very high density, fast read and fast write cycles, low power consumption, and low cost.

4.3 Memory Expansion

In many applications there is a need to expand the system memory capacity. In the next examples we will present two approaches which use small RAMs as building blocks for making larger memories [32].

Example 1 Expand the address line

The task is to build a 256K x 8 RAM from 64K x 8 RAMs. A 64K x 8 RAM has ($2^6 \cdot 2^{10} = 2^{16}$) 16 address lines, and 8 data lines. In order to expand the address to 256K ($2^8 \cdot 2^{10} = 2^{18}$) we have to introduce two extra address lines, because we have twice as many words to address. We will need four (2^2) 64K x 8 RAMs. The 256K x 8 memory design with the corresponding address

ranges is shown in Figure 4.11. The two most significant address lines go to the decoder, which selects one of the four 64K x 8 RAM chips. The other 16 address lines are shared by the 64K x 8 chips. The 64K x 8 chips also share WR and DATA inputs.

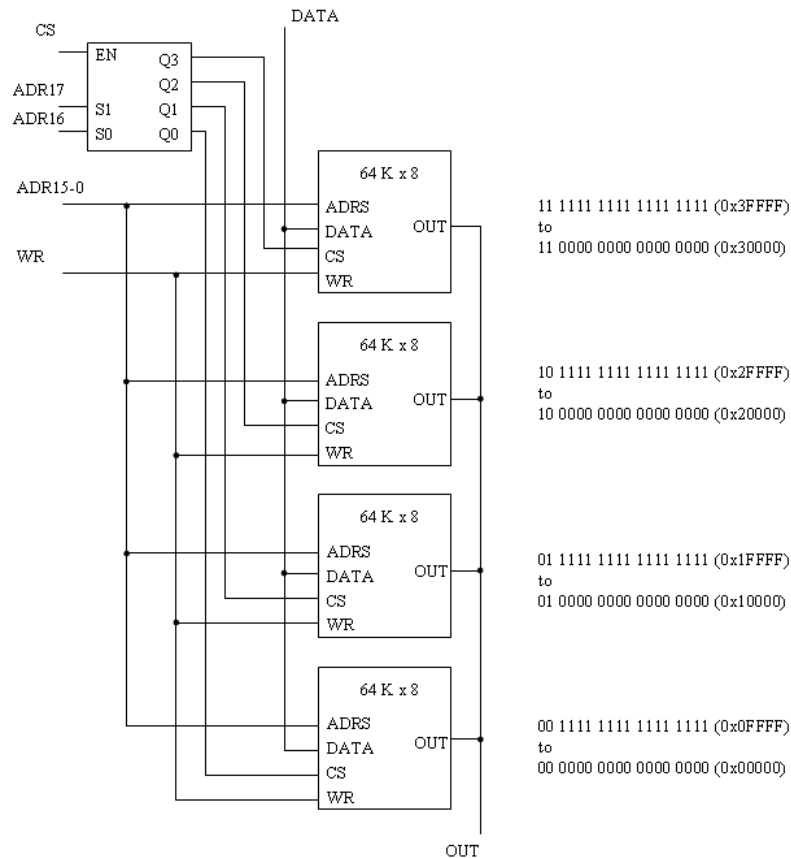


Figure 4.11 256K x 8 memory from 64K x 8 chips and address ranges

Example2 Expand the data line

For example, suppose we have available a large number of 64K x 8 of RAMs, but we need a 64K x 16 RAM. We have to use two 64K x 8 of RAMs to obtain 16 bits per word. We connect the 16 address inputs and the enable input to the two ROMs, as is shown in Figure 4.12. We group the two 8-bit outputs into our desired 16-bit output. Thus, each ROM stores one byte of the 16-bit word.

The left chip contains the most significant 8 bits of the data. The right chip contains the lower 8 bits of the data.

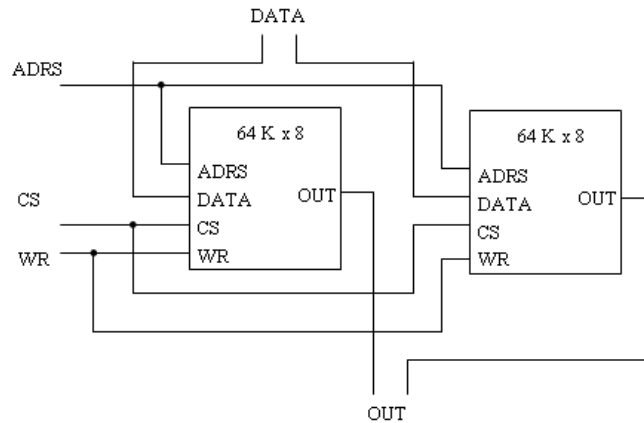


Figure 4.12 64K x 16 RAM, created from two 64K x 8 chips

Example 3 Expand the address and data line

The task is to build a 4096 x 32 RAM from 1024 x 8 RAMs. In this case it is necessary to create more and wider words. In the first step we generate a 4096 x 8 RAM by using 4 RAMs. The top two address lines are connected to a 2x4 decoder to select the appropriate RAM. Finally, we widen the RAM by adding 3 more RAMs to each row.

Problems

- 4.1 Draw a logic structure of a 1K x 8 RAM built up from 1K x 1 RAMs
- 4.2 Draw a logic structure of a 32 x 4 RAM built up from 16 x 4 RAMs
- 4.3 Draw a logic structure of a 32 x 8 ROM built up from 16 x 4 ROMs, giving the corresponding address ranges
- 4.4 Summarize the main differences between the DRAM and SRAM memories
- 4.5 Summarize the main differences between the EPROM and EEPROM memories

Chapter 5

Microprocessors Basics

In general, a microcomputer or microprocessor system is a set of components built up from microprocessor, memory elements and input/output units. The so called single purpose processors [20] are able to perform a single processing task. They are fast and have power efficient computation. In the second main group belong the popular and more widely known general purpose processors, the programmable processors. They are mass-produced and used in many applications. This chapter presents a microcomputer organization, and introduces a typical, simple microprocessor, illustrating its few instructions.

5.1 Basic Microcomputer Organization

The microcomputer main units are: the **input unit**, the **memory unit**, the **arithmetic unit**, the **control unit**, and the **output unit** [24]. The basic microcomputer organization is shown in Figure 5.1.

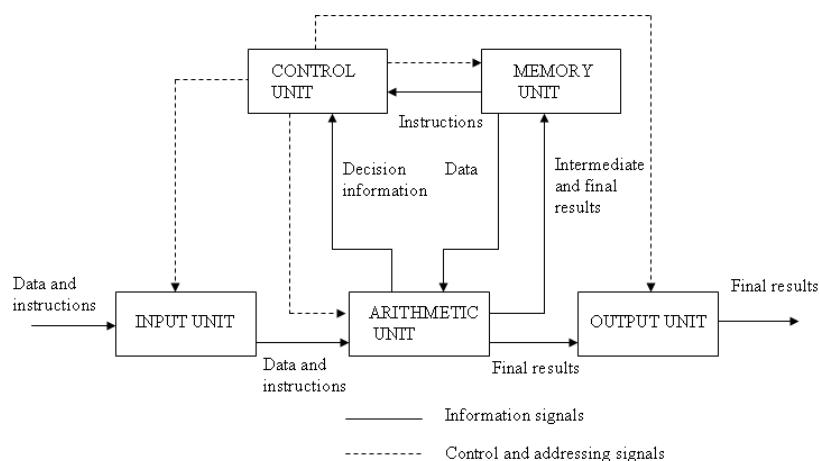


Figure 5.1 Basic microcomputer organization

The program and data are first stored in the memory unit via the input unit. The control unit interprets and executes the introduced instructions. The arithmetic

unit contains the necessary data for execution, and here occurs most of the data handling. Finally, the output unit delivers the results to output. The arithmetic unit and control unit constitutes the **central processing unit (CPU)**, which is the **microprocessor** in a microcomputer system [24].

The Memory Unit

The memory unit consists of substorage elements, called **registers** [34]. They are able to hold one computer word. The memory place where the substorage elements are present is known as a **location**. Each location has a unique **address** (integer number). The computer data and programs are stored in the memory forms like: ROM and RAM (see Chapter 4).

Beside the memory unit, the registers are able to store the information, as groups of digits. The group of binary digits handled together is called **word**. Modern processors, including embedded systems, usually have a word size of 8, 16, 24, 32 or 64 bits, while modern general purpose computers usually use 32 or 64 bits [33].

The Input/Output Units

Via the Input/Output units the computer communicates with the outside world. These devices are called **peripherals**.

The ALU and control unit properties and roles will be discussed in the next subsection.

In many applications, in order to achieve a faster computational speed, a **direct memory access (DMA)** is allowed, when the I/O units could access memory directly, not through the arithmetic unit. Figure 5.1 shows a direct link between the I/O units and the arithmetic unit.

5.2 General Purpose Microprocessor

As is shown in Figure 5.2 a programmable **processor** consists of two main parts: a datapath and a control unit.

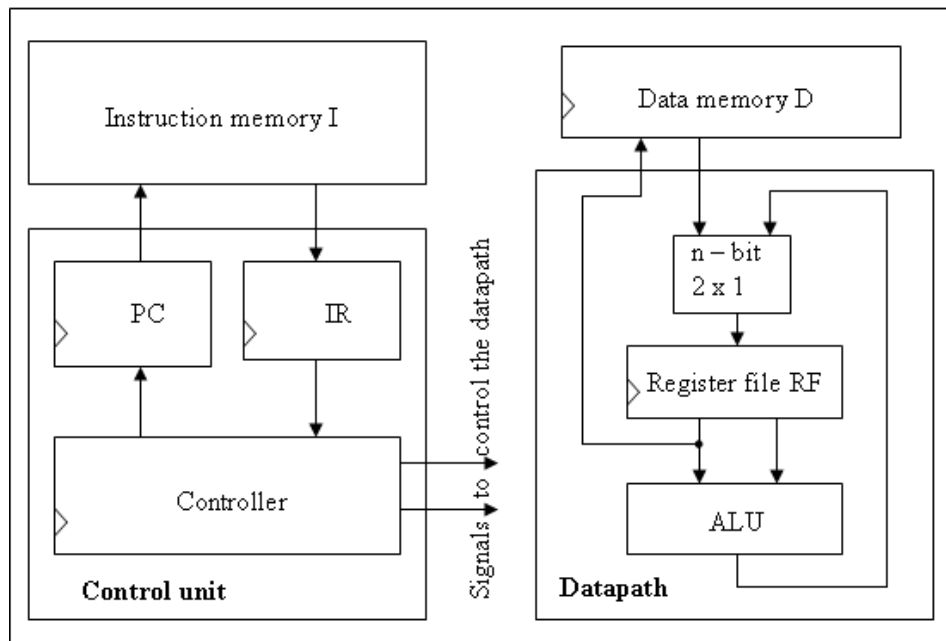


Figure 5.2 Basic architecture of a general purpose processor

Basic Datapath

Programmable processor datapath (see Figure 5.2) main units are: data memory, register file and ALU [20]. The **data memory D** manipulates all the accessible data (input, output data). The **load operation** reads data from data memory and loads them into one of the register files. **ALU operation** (typical includes addition, subtraction, AND, OR, shift operations etc., while the data is being transferred) transforms register data and the result is send back into any register of the register file. **Store operation** writes data from any register in the register file to any data memory location. Each operation can be executed in one clock cycle. In ALU the data are incorporated in the **accumulator** and **scratchpad registers**. The **flag bit** is also included in the arithmetic unit. Its

value provides information regarding to the course of computation. The set of flag bits is kept in a special register, the **program status word** (PSW).

Basic Control Unit

A **program** is a sequence of **instructions** (able to handle **data**), written to perform a specified task. The program is stored in the processor **instruction memory** I (see Figure 5.2). The control unit requires a **controller**, which executes the instructions [20].

The control unit controls and supervises the computer operations. Its role is to receive the instructions, to decode them, and to generate the necessary signals for execution. Within the control unit takes place a **program counter** (PC), which indicates the instructions location. A program counter keeps the address of the instruction to be executed [36]. It is incremented after each instruction byte (the instruction locations are usually in sequence).

The information transmission between the microprocessor and memory and I/O devices is done via the **address** (unidirectional), **data** (bidirectional), and **control** buses. The control bus lines could be uni- or bidirectional. Over this set of lines the signals manage timing and status information [20].

The **instruction register** (IR) stores the control unit instructions, maintains the acceptance of the first byte of every instruction via the data bus from memory.

The control unit's role is to provide the synchronization between various units. Several clock periods are needed for example, to **fetch** the instruction from memory, to **decode** it (determine the operation and operands of the instruction), and finally to **execute** it (carry out the instruction's operations using the datapath) [35, 39].

Each of these time intervals, including one or more clock periods, is called a **machine cycle** [24]. The whole time period which involves the fetching, decoding and executing of an instruction is called an **instruction cycle**.

Each particular microprocessor has its own **instruction format**.

5.3 Instruction Sets

An instruction is a collection of bits that instructs the processor to perform a specific operation. The collection of all instructions for a processor is called instruction set. The user specifies the operations to be performed by the processor and their sequence by the use of a program, which is a list of instructions. A thorough description of the instruction set for a processor is

called instruction set architecture (ISA) [37]. The microprocessor instructions set depends on the microprocessor type.

In the next we consider a processor which uses 16-bit instructions and the instruction memory is 16-bits wide [20]. We define a simple three instruction set where the most significant 4 bits refers to the operation, and the next 12 bits include register file and data memory addresses.

Load instruction – 0000 r3r2r1r0 d7d6d5d4d3d2d1d0

This instruction moves the data (leaves the memory location invariably) from data memory (whose address is given by d7d6d5d4d3d2d1d0) into the register file (whose address is indicated by r3r2r1r0).

Store instruction – 0001 r3r2r1r0 d7d6d5d4d3d2d1d0

This instruction moves the data from the register file to data memory (without changing the register contents).

Add instruction – 0010 ra3ra2ra1ra0 rb3rb2rb1rb0 rc3rc2rc1rc0

The addition result of two register files registers (represented by rb3rb2rb1rb0 and rc3rc2rc1rc0) is stored in the register file register **a** (represented by ra3ra2ra1ra0).

In a real programmable processor we need much more instructions (perhaps 100 or more) [20]. In the next we extend the instruction set by introducing three more instruction types.

Load-constant instruction – 0011 r3r2r1r0 c7c6c5c4c3c2c1c0

A binary number (represented by c7c6c5c4c3c2c1c0), known as a constant, is loaded into the register (represented by r3r2r1r0 bits). The constant is a value, part of the program.

Subtract instruction – 0100 ra3ra2ra1ra0 rb3rb2rb1rb0 rc3rc2rc1rc0

The subtraction result of two register files registers (represented by rb3rb2rb1rb0 and rc3rc2rc1rc0) is stored in the register file register **a** (represented by ra3ra2ra1ra0).

Jump-if-zero instruction – 0101 ra3ra2ra1ra0 o7o6o5o4o3o2o1o0

This instruction specifies that if the content of the register defined by ra3ra2ra1ra0 bits is **0**, we have to load the PC value plus o7o6o5o4o3o2o1o0. The result is an 8-bit number in 2's complement form.

The addressing mode of the instruction shows the method in which the address field is defined. Several addressing modes exist, as [38]:

- Direct Addressing, wherein the effective address is given in the instruction;
- Immediate Addressing, wherein the operand for the instruction is part of the instruction itself;
- Indexed Addressing, which allows that the stated address in an instruction could be added to the content of a so called index register;
- Indirect Addressing, which works as a pointer, indicating the location in which the effective address (of the stated address in the address field) can be found;
- Relative Addressing, wherein the stated address in the instruction is added to the content of the program counter to produce the effective address;
- Page Addressing Modes, used when the microcomputer memory is larger than can be directly addressed by an instruction (the memory might be divided into pages).

In a microcomputer system it is vital to handle the timing of signals which appear at the interfaces between their main components: a microprocessor, memory units, I/O registers, and peripheral devices.

The interfacing ensures the signal compatibility between the memory units and I/O registers to the microprocessor buses. This area involves the ability to handle buses timing and control, and the data transfer at a given time between the component and the microprocessor. The interfacing is responsible to make proper connections between microcomputer and peripheral devices, data channels, and controllers.

The I/O ports, handshaking, main memory interfacing, direct memory access, program interrupts, microprocessor clocks etc. support the compatible interconnection (regarding to timing, data format and signal type) between a microprocessor with various system elements.

These topics will be discussed within the framework of a new book dedicated only to microprocessors and microcomputers.

References

- [1] <http://www.liacs.nl/~stefanov/courses/DITE/lectures/DITE01.pdf>
- [2] <http://educyclopedia.karadimov.info/library/218-1.pdf>
- [3] <http://www.gutenberg.org/zipcat.php/15114/15114-pdf.pdf>
- [4] http://educyclopedia.karadimov.info/library/Boolean_algebra.pdf
- [5] www.cpe.ku.ac.th
- [6] <http://www.liacs.nl/~stefanov/courses/DITE/lectures/DITE02.pdf>
- [7] <http://www.liacs.nl/~stefanov/courses/DITE/lectures/DITE03.pdf>
- [8] www.ddpp.com/DDPP3_pdf/IEEEsyms.pdf
- [9] http://academic.evergreen.edu/projects/biophysics/technotes/misc/bin_math.htm
- [10] <http://www.utdallas.edu/~dodge/EE2310/lec3.pdf>
- [11] G. S. White, Coded Decimal Number Systems for Digital Computers, Proc., IRE, Vol.41, No.10, pp. 1450-1452, October, 1953.
- [12] <http://www.inf.fu-berlin.de/lehre/WS00/19504-V/Chapter1.pdf>
- [13] A. Anand Kumar, Fundamentals Of Digital Circuits, PHI learning Pvt. Ltd. 2003
- [14] <http://www.cis.upenn.edu/~palsetia/cit595s08/Lectures08/ErrorCD.pdf>
- [15] <http://en.wikipedia.org/wiki/Unicode>
- [16] <http://www.liacs.nl/~stefanov/courses/DITE/lectures/DITE04.pdf>
- [17] S.Givant, P. Halmos, Introduction to Boolean Algebras, Springer, Series: Undergraduate Texts in Mathematics, 2009
- [18] <http://www.asic-world.com/digital/combo2.htm>
- [19] http://www.electronics-tutorials.ws/combination/comb_4.html
- [20] F. Vahid, Digital Design, John Wiley & Sons Inc. 2007
- [21] <http://www.analog.com/static/imported-files/tutorials/MT-083.pdf>
- [22] R. P. Jain, Modern Digital Electronics (4th Edition), Tata McGraw Hill Education Private Limited, 2010
- [23] <http://www.ti.com/lit/ml/sgyn133/sgyn133.pdf>

- [24] D. D. Givone, R. P. Roesser, Microprocessors/Microcomputers An introduction, McGraw-Hill Book Company, 1980
- [25] <http://www.uotechnology.edu.iq/dep-eee/lectures/4th/electronic/microelectronics/part1.pdf>
- [26] <http://www.liacs.nl/~stefanov/courses/DITE/lectures/DITE07.pdf>
- [27] https://maxwell.ict.griffith.edu.au/yg/teaching/dns/ds_module3_p1.pdf
- [28] <http://www.scribd.com/doc/36619372/114/Ring-Counter-Applications>
- [29] https://maxwell.ict.griffith.edu.au/yg/teaching/dns/dns_module3_p2.pdf
- [30] https://maxwell.ict.griffith.edu.au/yg/teaching/dns/ds_module3_p1.pdf
- [31] V. A. Pedroni: Digital Electronics and design with VHDL, Elsevier 2008
- [32] <http://www.liacs.nl/~stefanov/courses/DITE/lectures/DITE11.pdf>
- [33] [http://en.wikipedia.org/wiki/Word_\(computer_architecture\)](http://en.wikipedia.org/wiki/Word_(computer_architecture))
- [34] http://en.wikipedia.org/wiki/IBM_System/360
- [35] http://en.wikipedia.org/wiki/Instruction_cycle
- [36] http://en.wikipedia.org/wiki/Honeywell_316
- [37] <http://www.liacs.nl/~stefanov/courses/DITE/lectures/DITE12.pdf>
- [38] R. S. Goankar, Microprocessor Architecture, Programming and Applications with 8085, 5th Edition, Prentice Hall
- [39] http://cseweb.ucsd.edu/classes/sp08/cse140L/lectures/lab_wk9.pdf

Further References

- J. E. Whitesitt: Boolean Algebra and Its Applications, Courier Dover Publications, 1995
- D. D. Givone, Digital Principles and Design, McGraw-Hill, 2003
- J. F. Wakerly:Review: Digital Design: Principles and Practices, Pearson Prentice-Hall, 2012
- M. M. Mano, M. D. Ciletti: Digital Design, Pearson Education, 2008
- Jr. C. H. Roth, L. L. Kinney, Fundamentals of Logic Design, Cengage Learning, 2009
- J. F. Wakerly, Digital Design: Principles and Practices Package (4th Edition), Pearson Prentice Hall, 2005
- A. P. Godse, A. Deepali, Godse Digital Techniques, Technical Publications, 2009