

—

DIGITAL ELECTRONICS

Author: Dr.-Ing. S. I. Kamau

Contents

1	NUMBER SYSTEMS AND CODES	1
1.1	Introduction	1
1.2	Number Systems	1
1.3	Decimal System	2
1.4	Binary System	2
1.4.1	Binary to Decimal Conversion	2
1.4.2	Decimal to Binary Conversion	3
1.5	Octal System	4
1.5.1	Octal to Decimal Conversion	4
1.5.2	Decimal to Octal Conversion	5
1.5.3	Octal to Binary Conversion	5
1.5.4	Binary to Octal conversion	6
1.6	Hexadecimal Number System	6
1.6.1	Hexadecimal to Decimal Conversion	7
1.6.2	Decimal to Hexadecimal Conversion	7
1.6.3	Hexadecimal to Binary Conversion	8
1.6.4	Binary to Hexadecimal conversion	8
1.7	Signed Binary Numbers	9
1.7.1	Sign-Magnitude Notation	9
1.7.2	Ones Complement Notation	10
1.7.3	Twos Complement Notation	11
1.8	Binary Number Codes	13
1.8.1	Binary Coded Decimal (BCD) code	13
1.8.2	Excess-3 code	14

1.8.3	Gray Code	15
2	BOOLEAN ALGEBRA AND LOGIC GATES	17
2.1	Introduction	17
2.2	Basic Operations of Boolean Algebra	17
2.2.1	The OR operation	18
2.2.2	The AND operation	19
2.2.3	NOT operation	19
2.3	Other Logic Gates	20
2.3.1	The NOR gate	20
2.3.2	The NAND gate	21
2.3.3	The Exclusive-OR gate	21
2.3.4	The Exclusive-NOR gate	22
2.4	Laws of Boolean Algebra	22
2.5	Proving Boolean Theorems	23
2.5.1	Proof by truth-table	23
2.5.2	Proof by Algebraic means	24
2.6	Standard Forms for Boolean Functions	24
2.6.1	Standard Sum of Products form	24
2.6.2	Standard Product of Sums form	26
3	SIMPLIFICATION OF BOOLEAN EXPRESSIONS AND COMBINATIONAL LOGIC CIR- CUIT DESIGN	28
3.1	Simplification of Boolean Expressions	28
3.1.1	Algebraic Simplification	29
3.1.2	The Karnaugh Map	32
3.1.3	Looping	34
3.1.4	Complete Simplification Procedure	39
3.1.5	Obtaining Product of Sums Expressions from K-maps	40
3.2	Don't Care Terms	42
3.3	NAND/NOR gate circuit implementation	43
3.4	SSI IC-BASED Combinational Logic Circuit Design	44
3.4.1	Introduction	44

3.4.2	Design Procedure	44
3.4.3	Examples and exercises	45
4	SEQUENTIAL LOGIC CIRCUITS	50
4.1	Introduction	50
4.2	Flip-Flops	50
4.2.1	The NAND-gate latch	51
4.2.2	The SET-RESET flip-flop	53
4.2.3	The D Flip-Flop	53
4.2.4	The T flip-flop	54
4.2.5	The JK flip-flop	54
4.3	Clock signals and clocked flip-flops	55
4.3.1	Introduction	55
4.3.2	Level-Driven flip-flops	57
4.3.3	The Master-Slave flip-flop	58
4.3.4	Edge triggered flip-flops	59
4.3.5	Asynchronous inputs	61
4.3.6	IC flip-flops	62
4.4	Flip-flop timing parameters	62
4.5	Flip-flop excitation tables	63
4.6	Derivation of one flip-flop function form another	64
4.7	Counters	67
4.7.1	Classification of counters	68
4.7.2	Asynchronous Counters	69
4.7.3	Asynchronous Counters with MOD numbers $< 2^N$	72
4.7.4	Synchronous Counters	74
4.7.5	Shift Register Counters	79
4.7.6	I.C. Counters	81
4.8	Registers	81
5	LOGIC FAMILIES AND THEIR CHARACTERISTICS	84
5.1	Introduction	84
5.2	Characteristics	85

5.3	Definitions	86
5.4	Current Sourcing and Current Sinking	86
5.5	DTL gates	88
5.6	TTL gates	89
5.6.1	Introduction	89
5.6.2	Operation of the NAND gate	90
5.6.3	Unconnected Inputs (floating inputs)	91
5.6.4	Current Sourcing and Current Sinking – Revisited	92
5.6.5	TTL Loading (Fan-Out)	93
5.6.6	Totem Pole Outputs	95
5.6.7	TTL Open-Collector Outputs	97
5.6.8	Tristate Devices	98
5.6.9	TTL 74 series	99
5.6.10	Comparison between TTL and CMOS	99
5.6.11	CMOS handling precautions	100
5.6.12	CMOS logic sub-families	101
6	MSI COMBINATIONAL LOGIC CIRCUITS AND THEIR APPLICATIONS	102
6.1	Introduction	102
6.2	Medium Scale Integrated (MSI) devices	102
6.3	Decoders	103
6.3.1	The decoder function	103
6.3.2	ENABLE inputs	105
6.3.3	The 74138 1 of 8 decoder	105
6.3.4	BCD to decimal decoder	108
6.3.5	Decoder applications	108
6.4	Encoders	112
6.5	Multiplexers	116
6.5.1	The multiplexer function	116
6.5.2	2-input multiplexer	116
6.5.3	4-input multiplexer	117
6.5.4	4-input multiplexer with ENABLE input	117

6.5.5	Examples of IC multiplexers	118
6.5.6	The ENABLE input(s) in multiplexers	120
6.5.7	Multiplexer applications	120
6.6	Demultiplexers	124
6.7	Adders and Subtractors	125
6.7.1	The half adder	125
6.7.2	The full adder	126
6.7.3	The half subtractor	126
6.7.4	The full subtractor	127
6.7.5	Serial and parallel addition	128
6.7.6	Examples of IC adders	128
6.7.7	The 2s complement adder/subtractor	129
6.7.8	Carry propagation	130
7	SEMICONDUCTOR MEMORY DEVICES	131
7.1	Introduction	131
7.2	Memory Terminology	131
7.3	Semiconductor Memory Technologies	133
7.4	Semiconductor Memory Operation	135
7.5	Read-Only Memories (ROMs)	136
7.5.1	ROM Timing	136
7.5.2	Mask-Programmed ROM (MROM)	137
7.5.3	(User) Programmable ROM (PROM)	138
7.5.4	Erasable Programmable ROM (EPROM)	138
7.5.5	Electrically Erasable Programmable ROM (EEPROM or E^2PROM)	141
7.5.6	ROM Applications	142
7.6	Programmable Logic Devices (PLDs)	143
7.7	Semiconductor RAMs	146
7.7.1	Static RAM Architecture	146
7.7.2	Static RAM memory Cell	147
7.7.3	Static RAM Timing	148
7.8	Dynamic RAM (DRAM)	150

7.8.1	DRAM cell	150
7.8.2	DRAM structure	151
7.8.3	DRAM refreshing	153
7.9	Memory Expansion	153
7.9.1	Expanding word size	153
7.9.2	Expanding Capacity	154
8	SEQUENTIAL NETWORKS AND TIMING CIRCUITS	157
8.1	Analysis of clocked sequential networks	157
8.1.1	Introduction	157
8.1.2	State equations, flip-flop input equations and output equations	158
8.1.3	State transition table	161
8.1.4	State diagram	163
8.2	State reduction	164
8.3	Design procedure for clocked sequential circuits	166
8.4	Timing circuits	169
8.4.1	Introduction	169
8.4.2	The 555 timer	170
8.4.3	Astable operation of the 555 timer	172
8.4.4	Adjusting the duty cycle of a 555 timer	174
8.4.5	Monostable operation of the 555 timer	174
8.4.6	Other monostable multivibrators	175
9	ANALOG-TO-DIGITAL (A/D) AND DIGITAL-TO-ANALOG (D/A) CONVERSION	178
9.1	Introduction	178
9.2	Digital-to-Analog Conversion	179
9.2.1	Introduction	179
9.2.2	DAC using weighted resistors	179
9.2.3	Modified weighted resistors DAC	181
9.2.4	The R-2R Ladder Network	182
9.2.5	DAC symbol	184
9.2.6	Specifications of DACs	184
9.3	Analog to Digital Conversion	185

9.3.1	Digital Ramp Analog to Digital Converter	185
9.3.2	The Successive Approximation Converter	187
9.3.3	Dual Slope Integration ADC (the integrating converter)	188
9.3.4	Flash ADCs	191
9.3.5	ADC symbol	192
9.3.6	ADC specifications	192
10 INTRODUCTION TO MICROCOMPUTERS AND MICROCOMPUTER ORGANIZATION		193
10.1	Introduction	193
10.2	Basic Microcomputer (μC) Components	194
10.3	The Bus System (The Highway Structure)	196
10.4	Microprocessors	196
10.4.1	General introduction	196
10.4.2	General microprocessor block diagram	199
10.5	The Programmer's model of a microcomputer	199
10.5.1	Introduction	199
10.5.2	The Programmer's Model of the Intel 8085	200
10.6	The 8085 Instruction Set	201
10.7	Memory Interfacing	203
10.7.1	Memory map (Address map)	203
10.7.2	Partial Address Decoding	204
10.7.3	Full Address Decoding	207

Chapter 1

NUMBER SYSTEMS AND CODES

1.1 Introduction

Digital quantities can only take on discrete values while analog quantities vary over a continuous range of values. Examples of analog quantities include temperature, speed. A digital system is a combination of devices designed to manipulate physical quantities that are represented in digital form as opposed to analog systems which manipulate systems which are represented in analog form. Examples of digital systems include electronic calculators, digital watches, digital voltmeters and digital computers. Examples of analog devices include pointer-type instruments like speedometers, voltmeters, analog computers, etc. Advantages of digital systems over analog quantities are:

- They are easier to design than analog systems.
- Easy to store large quantities of information.
- Accuracy and precision are greater
- digital circuits are less affected by noise.

1.2 Number Systems

All number systems are based on an ordered set of numbers called digits. The total number of digits used in a system is called the *base* or *radix* of the system e.g. base 10 (or radix 10) uses then ten digits 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9. The four number systems that are used in digital systems are:

- i) Decimal - Used in everyday calculations

- ii) Binary - Used in all digital systems, including digital computers
- iii) Octal
- iv) Hexadecimal - This number system, along with the Octal system are used as shorthand notation for the Binary number system.

1.3 Decimal System

This is a positional-value system, that is, the value of a digit depends on its position in the number. It is possible to design a digital system with ten states (decimal) but this would not be easy to design as it would mean designing a circuit with ten discrete voltage levels.

1.4 Binary System

This is known as Base 2 or Radix 2. Uses only two digits, 0 and 1. In digital systems, these two digits are known as *bits*. The binary system is a positional value system, with the weights as shown in Figure 1.1.

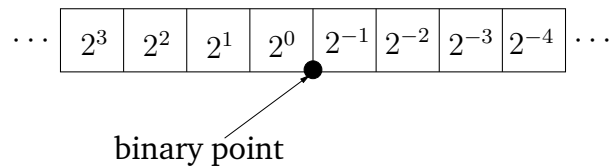


Figure 1.1: Weighting factors for the binary number system

1.4.1 Binary to Decimal Conversion

The decimal equivalent of the binary number $a_n a_{n-1} \dots a_1 a_0 . a_{-1} a_{-2} \dots a_{-m}$ is given by:

$$(2^n \times a_n) + (2^{n-1} \times a_{n-1}) + \dots + (2^1 \times a_1) + (2^0 \times a_0) + (2^{-1} \times a_{-1}) + \dots + (2^{-m} \times a_{-m})$$

Example:

$$1010.101_2 = (2^3 \times 1) + (2^2 \times 0) + (2^1 \times 1) + (2^0 \times 0) + (2^{-1} \times 1) + (2^{-2} \times 0) + (2^{-3} \times 1) = 10.625_{10}$$

1.4.2 Decimal to Binary Conversion

Integers For integers, we use repeated division by 2 (also known as successive division by 2). The example shown on Figure 1.2 shows how to apply this method to convert 53_{10} to binary.

2		53	
2		26	R 1
2		13	R 0
2		06	R 1
2		03	R 0
2		01	R 1
		00	R 1

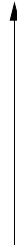


Figure 1.2: Example illustrating successive division by 2

The binary number is read in the direction shown by the arrow (upwards). In this case, we can see that $53_{10} = 110101_2$

Fractions In this case we use repeated (or successive) multiplication by 2. The table below shows this procedure to convert 0.8125_{10} to binary.

DECIMAL	BINARY
$0.8125 \times 2 = 1.625$ $1.625 - 1 = 0.625$	1
$0.625 \times 2 = 1.250$ $1.250 - 1 = 0.250$	1
$0.250 \times 2 = 0.500$	0
$0.500 \times 2 = 1$ $1 - 1 = 0$	1

In this case, the binary number is read *downwards*, so $0.8125_{10} = 0.1101_2$.

Note that in some cases, the decimal fraction cannot be represented in a finite number of bits - in this case, we have to truncate the binary number at some point e.g. conversion of 0.485_{10} to binary:

1.5.2 Decimal to Octal Conversion

Integers For integers, we use repeated (successive) division by 8. The example shown on Figure 1.4 shows how to apply this method to convert 459_{10} to octal.

$$\begin{array}{r|l}
 8 & 459 \\
 8 & 57 \quad \text{R } 3 \\
 8 & 07 \quad \text{R } 1 \\
 & 00 \quad \text{R } 7
 \end{array}
 \quad
 \begin{array}{c}
 \uparrow \\
 \\
 \\
 \end{array}$$

Figure 1.4: Example illustrating successive division by 8

The octal number is read in the direction shown by the arrow (upwards). In this case, we can see that $459_{10} = 713_8$.

Fractions In this case we use repeated (or successive) multiplication by 8. The table below shows this procedure to convert 0.78125_{10} to octal.

DECIMAL	OCTAL
$0.78125 \times 8 = 6.25$	
$6.25 - 6 = 0.25$	6
$0.25 \times 8 = 2.00$	
$2 - 2 = 0$	2

In this case, the octal number is read *downwards*, so $0.78125_{10} = 0.62_8$.

Exercise

Show that 0.485_{10} expressed in octal to six decimal places equals 0.370243_8 .

1.5.3 Octal to Binary Conversion

The procedure here is to convert each octal digit to its 3-bit binary equivalent then to juxtapose these codes to give us the equivalent binary code. The 3-bit binary codes corresponding to the octal digits are shown on the table below:

OCTAL	BINARY
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

As an example, suppose we would like to convert 713.62_8 to binary. From the above table, we can see that the 3-bit binary codes for 7, 1, 3, 6 and 2 are respectively 111, 001, 011, 110 and 010. We can therefore directly write that $713.62_8 = 111001011.110010_2$.

1.5.4 Binary to Octal conversion

In this case, we divide the binary number in groups of 3 bits, starting from the binary point. We then use the table shown in the previous section to get the corresponding octal digits. As an example, suppose we want to convert 10111101.1111_2 to octal. The procedure is illustrated on Figure 1.5.

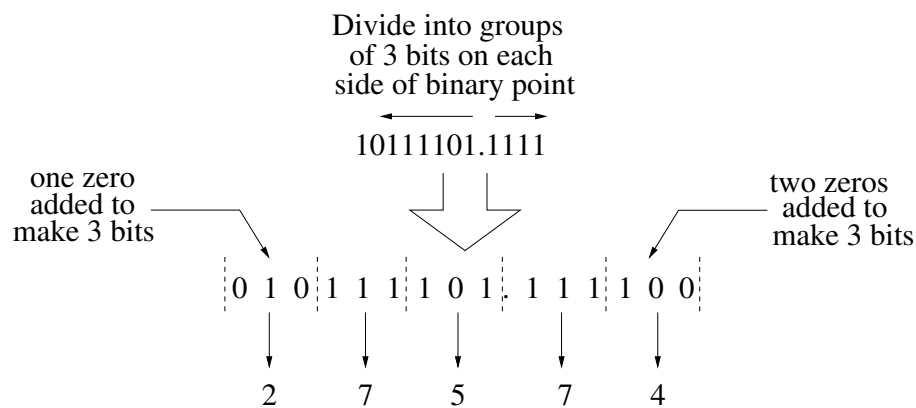


Figure 1.5: Binary to octal conversion

Hence $10111101.1111_2 = 275.74_8$.

We can see that it is very easy to perform binary to octal operation and vice-versa.

1.6 Hexadecimal Number System

The hexadecimal number system uses 16 digits, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F. The table below shows the correlation between these digits and the decimal, (4-bit) binary and octal systems.

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

The weighting factors are shown in Figure 1.6.

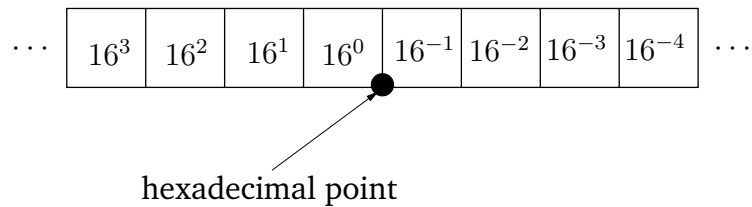


Figure 1.6: Weighting factors for the hexadecimal number system

1.6.1 Hexadecimal to Decimal Conversion

The decimal equivalent of the hexadecimal number $a_n a_{n-1} \cdots a_1 a_0 . a_{-1} a_{-2} \cdots a_{-m}$ is given by:

$$(16^n \times a_n) + \cdots + (16^1 \times a_1) + (16^0 \times a_0) + (16^{-1} \times a_{-1}) + \cdots + (16^{-m} \times a_{-m})$$

Example:

$$2EA.B_{16} = (16^2 \times 2) + (16^1 \times 14) + (16^0 \times 10) + (16^{-1} \times 11) = 746.6875_{10}$$

1.6.2 Decimal to Hexadecimal Conversion

Integers For integers, we use repeated (successive) division by 16. Note that when a remainder exceeds 9, we replace it with the corresponding hexadecimal digit as shown in the above table. The example shown on Figure 1.7 shows how to apply this method to convert 428_{10} to hexadecimal.

The hexadecimal number is read in the direction shown by the arrow (*upwards*). In this case, we can see that $428_{10} = 1AC_{16}$.

$$\begin{array}{r}
 16 \overline{) 428} \\
 16 \overline{) 26} \quad \text{R } 12 \\
 16 \overline{) 01} \quad \text{R } 10 \\
 \quad \quad 00 \quad \text{R } 1
 \end{array}
 \begin{array}{c}
 \uparrow \\
 \uparrow \\
 \uparrow
 \end{array}$$

Figure 1.7: Example illustrating successive division by 16

Fractions In this case we use repeated (or successive) multiplication by 16. The table below shows this procedure in converting 0.75390625_{10} .

DECIMAL	HEXADECIMAL
$0.75390625 \times 16 = 12.0625$	
$12.0625 - 12 = 0.0625$	$12 = C$
$0.0625 \times 16 = 1.00$	
$1 - 1 = 0$	1

In this case, the hexadecimal number is read *downwards*, so $0.75390625_{10} = 0.C1_{16}$.

Exercise

Show that 0.485_{10} expressed in hexadecimal to five decimal places equals $0.7C28F_{16}$.

1.6.3 Hexadecimal to Binary Conversion

The procedure is similar to that of Octal to Binary conversion, the only difference being that we first convert each hexadecimal digit into *4-bit* binary. The 4-bit binary codes representing each hexadecimal digit are tabulated at the beginning of this section.

As an example, suppose we want to convert $2EA.B_{16}$ to binary. From the above table, we can see that the 4-bit binary codes for 2, E, A, and B are respectively 0010, 1110, 1010 and 1011 so we can therefore directly write that $2EA_{16} = 001011101010.1011_2$.

1.6.4 Binary to Hexadecimal conversion

In this case, we divide the binary number in groups of 4 bits, starting from the binary point. We then use the table shown at the beginning of this section to get the corresponding hexadecimal digits. As an example, suppose we want to convert 110110011.01011_2 to octal. The procedure is illustrated on Figure 1.8 below.

Hence $110110011.01011_2 = 1B3.58_{16}$.

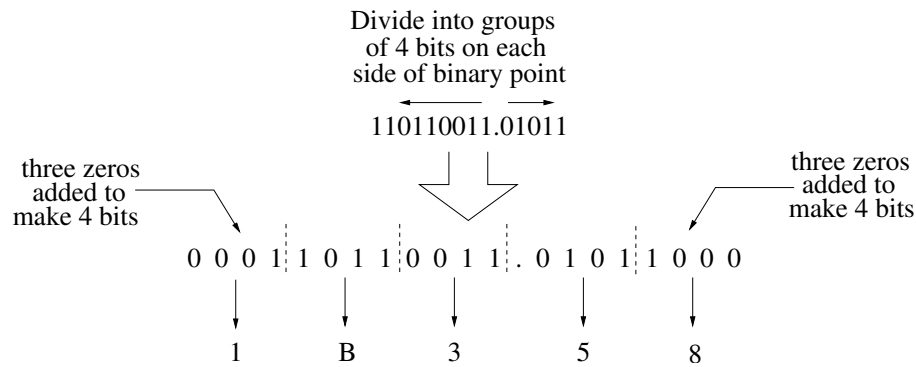


Figure 1.8: Binary to hexadecimal conversion

1.7 Signed Binary Numbers

Our discussion so far has assumed that we are dealing with positive numbers. The binary numbers discussed so far are known as unsigned binary numbers.

Digital systems represent all information with binary digits (bits 0,1). Since digital computers and calculators handle negative as well as positive numbers, some means is required for representing the sign of the number (+ or -). This is done by the use of a sign bit. The most significant bit of a binary number is used to denote the sign of the number.

There are three notations that are commonly used representing signed numbers. These are:

- i) Sign-Magnitude Notation
- ii) Ones (1's) Complement Notation
- iii) Twos (2's) Complement Notation

In all these notations, positive numbers have the Most Significant Bit (MSB) as zero, while negative numbers have an MSB of 1.

1.7.1 Sign-Magnitude Notation

To obtain the sign-magnitude notation of a given number, we first obtain its unsigned binary equivalent using the methods described in the previous sections. If the number is positive, we then add a zero (0) to become the MSB, and if the number is negative, we add a one (1) to become the MSB.

Example

Convert +53 and -53 to binary in sign-magnitude notation.

Solution:

The unsigned binary code for 53 can be obtained by successive division by 2 as 110101. For +53, we add a '0' sign bit as MSB to give the binary sign-magnitude code for +53 as 0110101. For -53, we add a '1' for a sign bit to get 1110101. We can tabulate the decimal equivalents of the 4-bit binary codes, assuming these codes are in sign-magnitude notation, as shown below:

Sign-Magnitude Code	Decimal
0000	+0
0001	+1
0010	+2
0011	+3
0100	+4
0101	+5
0110	+6
0111	+7
1000	-0
1001	-1
1010	-2
1011	-3
1100	-4
1101	-5
1110	-6
1111	-7

Generally, for N bits, the range of integers which can be represented using this notation $= -(2^{(N-1)} - 1) \leq I \leq (2^{(N-1)} - 1)$.

However, as you can see from the above table, this notation has two distinct patterns for zero, a positive zero and a negative zero. This creates complications in arithmetic operations, and for this reason, this notation is not commonly used.

1.7.2 Ones Complement Notation

To get the ones complement notation for a positive number, the unsigned binary notation of the number is obtained, after which the a zero (0) is added to the number as the MSB (This is similar to the Sign-Magnitude notation).

The ones complement notation of a negative number is obtained from the corresponding positive binary number by changing each zero in the digit to a 1, and each 1 in the positive binary number to a zero. As an example, we saw in the previous section that $53 = 110101$ in unsigned binary. +53 would be represented by 0110101 in Ones Complement Notation (OCN). To represent -53 in OCN, we simply complement all the bits in +53 to get 1001010. We can have a table similar to one in the previous section, this time assuming the binary codes are in the Ones Complement Notation.

Ones Complement	Decimal
0000	+0
0001	+1
0010	+2
0011	+3
0100	+4
0101	+5
0110	+6
0111	+7
1000	-7
1001	-6
1010	-5
1011	-4
1100	-3
1101	-2
1110	-1
1111	-0

Generally, for N bits, the range of integers which can be represented using this notation $= -(2^{(N-1)} - 1) \leq I \leq (2^{(N-1)} - 1)$.

Just as in the previous case, you can see from the above table that this notation also has two distinct patterns for zero, a positive zero and a negative zero.

To illustrate the problem created by the two patterns for zero, suppose we want to perform the operation $(7-4)$. This can be rewritten as $7+(-4)$. From the above table, $+7 = 0111$ and $-4 = 1011$. Adding these two codes gives 10010. Since we are dealing with 4-bit binary in this case, we can ignore the fifth bit to get 0010. From the above table once again, we see that 0010 corresponds to +2. But we know that $7 - 4 = +3$. The incorrect result obtained above is as a result of having two zeros. Generally, the presence of two distinct patterns for zero complicates arithmetic operations and for this reason, this notation is not commonly used.

1.7.3 Twos Complement Notation

The procedure for obtaining the Twos Complement Notation (TCN) of a positive number is similar to that of obtaining OCN for a positive number. For a negative number, you add 1 to the Least Significant Bit (LSB) position of the ones complement notation of the number.

Example

Obtain the TCN of +53 and -53.

Solution:

$53 = 110101$ in unsigned binary. Adding a sign bit gives $+53 = 0110101$ in TCN. To obtain the code for -53, we first obtain the ones complement notation of the code 0110101, which is 1001010. We then add 1 to the LSB position to get 1001011, which is the TCN of -53.

The decimal equivalent of the TCN binary code $a_n a_{n-1} \cdots a_1 a_0 . a_{-1} a_{-2} \cdots a_{-m}$ is given

by:

$$(-2^n \times a_n) + (2^{n-1} \times a_{n-1}) + \dots + (2^1 \times a_1) + (2^0 \times a_0) + (2^{-1} \times a_{-1}) + \dots + (2^{-m} \times a_{-m})$$

A table showing 4-bit TCN codes and their decimal equivalents is shown below:

Twos Complement	Decimal
0000	+0
0001	+1
0010	+2
0011	+3
0100	+4
0101	+5
0110	+6
0111	+7
1000	-8
1001	-7
1010	-6
1011	-5
1100	-4
1101	-3
1110	-2
1111	-1

Generally, for N bits, the range of integers which can be represented using this notation $= -2^{(N-1)} \leq I \leq 2^{(N-1)} - 1$.

In this case, there is only one zero, so there are no problems with arithmetic. In fact, digital computers use twos complement binary in arithmetic operations since addition can be carried out just as addition (e.g. $7 - 4 = 7 + (-4)$). This means that the same circuit can be used for both addition and subtraction, which saves on the hardware to be used for these operations.

Example

Convert -29.625 into Twos Complement Binary.

Solution:

$$\begin{array}{rcl}
 29.625 & = & 1 \ 1 \ 1 \ 0 \ 1 \ . \ 1 \ 0 \ 1 \text{ (unsigned)} \\
 +29.625 & = & 0 \ 1 \ 1 \ 1 \ 0 \ 1 \ . \ 1 \ 0 \ 1 \text{ (signed)} \\
 -29.625 & = & 1 \ 0 \ 0 \ 0 \ 1 \ 0 \ . \ 0 \ 1 \ 0 \text{ (OCN)} \\
 & & \qquad \qquad \qquad + \qquad \qquad \qquad 1 \\
 \hline
 -29.625 & = & 1 \ 0 \ 0 \ 0 \ 1 \ 0 \ . \ 0 \ 1 \ 1 \text{ (TCN)}
 \end{array}$$

Hence the Twos Complement notation for -29.625 = 100010.011. As a cross-check, using the formula for converting a TCN number to decimal, we get:

$$\begin{aligned}
 & (-2^5 \times 1) + (2^4 \times 0) + (2^3 \times 0) + (2^2 \times 0) + (2^1 \times 1) + (2^0 \times 0) + (2^{-1} \times 1) + (2^{-2} \times 0) + (2^{-3} \times 1) \\
 & = -32 + 2 + 0.5 + 0.125 = -29.625
 \end{aligned}$$

In the above example, -29.625 is represented using 9 bits. Suppose we wanted to represent the number using, say, 16 bits. Then the procedure is shown below:

$$\begin{array}{rcl}
29.625 & = & 1\ 1\ 1\ 0\ 1\ .\ 1\ 0\ 1\ \text{(unsigned)} \\
+29.625 & = & 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 0\ 1\ .\ 1\ 0\ 1\ \text{(signed)} \\
-29.625 & = & 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ 1\ 0\ .\ 0\ 1\ 0\ \text{(OCN)} \\
& & + 1 \\
\hline
-29.625 & = & 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ 1\ 0\ .\ 0\ 1\ 1\ \text{(TCN)}
\end{array}$$

As we can see, after obtaining the unsigned binary, we add leading zeros and a '0' sign-bit to make 16 bits then we proceed as usual. Use the formula for TCN to decimal conversion to show that the result obtained, 1111111100010.011 is equal to -29.625.

1.8 Binary Number Codes

These are binary codes which have special applications.

1.8.1 Binary Coded Decimal (BCD) code

BCD code represents each digit of a decimal number by a 4-bit binary number. The codes used are tabulated below:

DECIMAL	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Note that BCD code uses binary codes 0000 to 1001 to represent decimal digits, it does not use codes 1010, 1011, 1100, 1101, 1110 and 1111. It is a weighted code. The weightings for a 12-bit BCD number are shown below:

800	400	200	100	80	40	20	10	8	4	2	1
-----	-----	-----	-----	----	----	----	----	---	---	---	---

To convert a decimal number to BCD code, we simply write out the BCD code for each digit e.g. to convert the decimal number 137 to BCD, we can see from the above table that the BCD code for 1 is 0001, for 3 is 0011 and for 7 is 0111, so the BCD code for 137 is 000100110111.

To convert a BCD code number to decimal, we simply group the bits in groups of 4 bits each and write out the decimal digit corresponding to each decimal digit.

The main advantage of BCD code is the relative ease of converting to and from decimal. BCD code is used in digital machines whenever decimal information is either applied as inputs or displayed as outputs e.g. digital voltmeters, digital clocks, e.t.c. use BCD because they display information in decimal. Electronic calculators use BCD because the input numbers are entered in decimal via the keypad and the output numbers displayed in decimal.

However, BCD code is not used in modern high-speed digital computers because:

- It requires more storage space
- The arithmetic with BCD is more complicated (*can you explain these points??*)

1.8.2 Excess-3 code

The excess-3 code (also known as Xs-3 code) for a decimal number is obtained in the same manner as for BCD, except that 3 is added to each digit before encoding it in binary. The example below shows how to convert 59 to Xs-3 code.

$$\begin{array}{r}
 5 \\
 + 3 \\
 \hline
 8 \\
 \downarrow \\
 1000
 \end{array}
 \qquad
 \begin{array}{r}
 9 \\
 + 3 \\
 \hline
 12 \\
 \downarrow \\
 1100
 \end{array}$$

The Xs-3 code for 59 is therefore 10001100. The table below shows the codes used by Xs-3 code, and these are listed alongside BCD codes.

DECIMAL	BCD	Xs-3 CODE
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100

The Xs-3 code does not use codes 0000, 0001, 0010, 1101, 1110 and 1111.

The advantage of this code is that at least one 1 is present in all codes, providing an error-detection ability.

1.8.3 Gray Code

In gray code, only one bit changes in going from one number to the next. It is a non-weighted code - bit positions in the code do not have any specific weights attached to them.

Converting from Binary to Gray Code

- i) Record the MSB of the binary number.
- ii) Add the binary MSB to the next bit position, record the sum and neglect any carries.
- iii) Record successive sums until completed.

Applying this procedure on the binary code 101110110, we get the equivalent Gray Code to be 111001101.

Converting from Gray to Binary

- i) Record the MSB of the Gray Code number.
- ii) Add the binary MSB to the next significant bit position of the Gray Code number, again recording the sum and ignoring any carries.
- iii) Continue the process until completed.

This procedure applied to Gray Code number 111001101 yields 101110110.

The table below shows the Gray codes for the first 16 decimal digits.

DECIMAL	BINARY	GRAY
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

Observe that only one bit position changes in the binary code in moving from one number to the next. Gray code is often used in situations where other codes might produce erroneous or ambiguous results during those transitions in which more than 1 bit of the code is changing e.g. in the transition from 7 to 8 in binary, all bit positions change, and in a practical circuit, these bit positions may not change at exactly the same time and this could cause problems in some circuits.

Chapter 2

BOOLEAN ALGEBRA AND LOGIC GATES

2.1 Introduction

In Boolean algebra, the variables (known as Boolean variables) are allowed to have only two possible values, usually denoted as 0 and 1, unlike ordinary algebra where variables can take on infinitely many values.

In Boolean algebra, we can have expressions such as:

$$x = f(A, B)$$

which is read as “ x is a function of variables A and B ”. A and B are Boolean variables and can only take on two possible values, 0 or 1. $f()$ is the Boolean operation on the variables.

2.2 Basic Operations of Boolean Algebra

Boolean algebra has only 3 basic operations:

- i) Logical addition (the OR operation), Symbol $+$
- ii) Logical multiplication (the AND operation), Symbol \cdot
- iii) Logical complementation (the NOT operation). Different books have different symbols for this operation, including $*$, $'$ and $-$

Any Boolean function, however complex, can be broken down to a combination of these three operations.

2.2.1 The OR operation

This operation operates on two or more variables. The expression:

$$x = A + B$$

is read as “ x equals A OR B ”. We can write the operation of the 2-variable in the form of a table as shown below:

A	B	x
0	0	0
0	1	1
1	0	1
1	1	1

A table, such as the one above, which shows how a logic circuit’s outputs respond to various combinations of logic levels at the inputs is known as a *truth-table*.

From the truth-table above, we can write that:

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

In general, a truth-table of m inputs has 2^m input combinations e.g. a 3-input OR operation (for the operation $x = A + B + C$) has $2^3 = 8$ input combinations, and it is shown on the table below:

A	B	C	x
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

The OR operation is implemented using an OR gate. The Figure 2.1 shows a 2-input OR gate and a 3-input OR gate.

In general, the OR operation produces a result of 1 when any of the input variables is 1. The OR operation produces a result of zero only when all the input variables are 0.

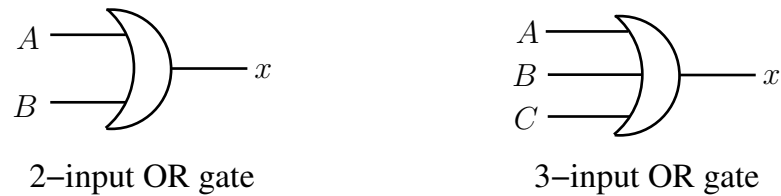


Figure 2.1: 2 and 3 input OR gates

2.2.2 The AND operation

This operates on two or more variables. The expression:

$$x = A \cdot B$$

is read as “ $x = A$ AND B ”. Note that in most cases, the dot between A and B is omitted and the expression simply written as $x = AB$. The truth-table for the 2-input AND operation is shown below:

A	B	x
0	0	0
0	1	0
1	0	0
1	1	1

The AND operation is implemented using an AND gate, and Figure 2.2 shows a 2-input and 3-input AND gate.



Figure 2.2: 2 and 3 input AND gates

For the AND operation, an output equal to 1 occurs only for the single case when all the inputs are 1. The output is 0 for any case where one or more inputs are 0.

2.2.3 NOT operation

The inputs have to be reduced to a single variable before a NOT operation can be performed. It is the inversion or complementation function.

If this operation is to be applied on a variable A , we can then write:

$$x = \overline{A}$$

which is read as “ $x = \text{NOT } A$ ”. The truth-table for this is shown below:

A	x
0	1
1	0

This means that $\overline{0} = 1$ (NOT ‘0’ = ‘1’) and $\overline{1} = 0$ (NOT ‘1’ = ‘0’). Note also that $\overline{\overline{0}} = 0$ and $\overline{\overline{1}} = 1$.

The NOT operation is implemented using a NOT gate, illustrated on Figure 2.3. The NOT gate is also known as an inverter.

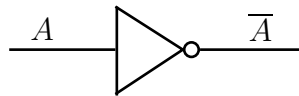


Figure 2.3: A NOT gate (inverter)

2.3 Other Logic Gates

2.3.1 The NOR gate

For two variables A and B , the NOR operation is defined as:

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

In this case, we read this as “ x equals NOT (A OR B). This case is equivalent to a 2-input OR gate and an inverter connected in series. The symbol for a 2-input NOR gate is shown on Figure 2.4.

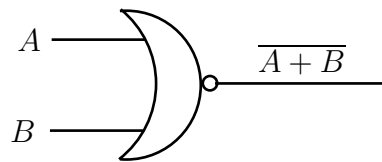


Figure 2.4: A 2-input NOR gate

The truth-table for a 2-input NOR gate is shown below:

A	B	x
0	0	1
0	1	0
1	0	0
1	1	0

2.3.2 The NAND gate

For three variables A , B and C , the NAND operation is defined as:

$$x = \overline{A \cdot B \cdot C}$$

In this case, we read this as “ x equals NOT (A AND B AND C). This is equivalent to a 3-input AND gate and an inverter connected in series. The symbol for a 3-input NAND gate is shown on Figure 2.5.

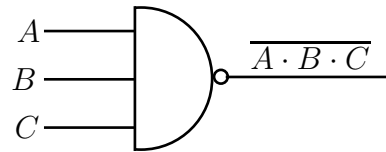


Figure 2.5: A 3-input NAND gate

The truth-table for a 3-input NAND gate is shown below:

A	B	C	x
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

2.3.3 The Exclusive-OR gate

For variables A and B , the Exclusive-OR function is defined as:

$$x = A \oplus B$$

and the truth-table is shown below:

A	B	x
0	0	0
0	1	1
1	0	1
1	1	0

The Exclusive-OR operation is sometimes abbreviated as XOR or EXOR. The operation is implemented using an Exclusive-OR gate, illustrated on Figure 2.6.

Note that the XOR gate has only two inputs.

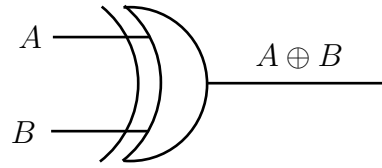


Figure 2.6: An Exclusive-OR gate

2.3.4 The Exclusive-NOR gate

This is usually abbreviated as XNOR or EXNOR gate. It is the complement of the XOR operation. For variables A and B ,

$$x = A \odot B = \overline{A \oplus B}$$

and the truth-table is shown below:

A	B	x
0	0	1
0	1	0
1	0	0
1	1	1

The operation is implemented using an Exclusive-NOR gate, illustrated on Figure 2.7.

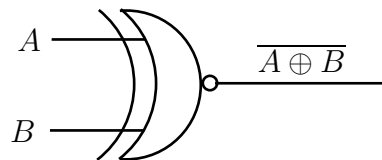


Figure 2.7: An Exclusive-NOR gate

Just as for the XOR gate, the XNOR gate has only two inputs.

2.4 Laws of Boolean Algebra

Basic Theorems	
$A + 0 = A$	$A \cdot 1 = A$
$A + 1 = 1$	$A \cdot 0 = 0$
$A + A = A$	$A \cdot A = A$
$A + \bar{A} = 1$	$A \cdot \bar{A} = 0$

Looking at the above table, we can see that the corresponding laws on either side are related by:

- i) Interchanging + and \cdot symbols
- ii) Interchanging 0 and 1

Theorems which are related to another by this double interchange are known as *duals*.

Other theorems, each listed along with its dual, are tabulated below:

1:	$A + B = B + A$	$A \cdot B = B \cdot A$
2:	$A + (B + C) = (A + B) + C$	$A \cdot (B \cdot C) = (A \cdot B) \cdot C$
3:	$A + B \cdot C = (A + B) \cdot (A + C)$	$A \cdot (B + C) = A \cdot B + A \cdot C$
4:	$A + A \cdot B = A$	$A \cdot (A + B) = A$
5:	$A + \bar{A} \cdot B = A + B$	$A \cdot (\bar{A} + B) = A \cdot B$
6:	$A \cdot B + A \cdot \bar{B} = A$	$(A + B) \cdot (A + \bar{B}) = A$
7:	$A \cdot B + \bar{A} \cdot C = (A + C) \cdot (\bar{A} + B)$	$(A + B) \cdot (\bar{A} + C) = A \cdot C + \bar{A} \cdot B$
8:	$A \cdot B + \bar{A} \cdot C + B \cdot C = A \cdot B + \bar{A} \cdot C$	$(A + B) \cdot (\bar{A} + C) \cdot (B + C) = (A + B) \cdot (\bar{A} + C)$
9:	$\overline{A + B + C + \dots} = \bar{A} \cdot \bar{B} \cdot \bar{C} \dots$	$\overline{A \cdot B \cdot C \dots} = \bar{A} + \bar{B} + \bar{C} + \dots$

Law 1 is the commutative law, 2 the associative law, 3 the distributive law, 4 is commonly referred to as the absorption theorem, 5 the simplification theorem, 6 the reduction theorem and 9 are the De Morgan's Theorems.

2.5 Proving Boolean Theorems

The theorems above may be proved by use of truth-table or by algebraic means.

2.5.1 Proof by truth-table

Example

Use a truth-table to prove that $AB + \bar{A}C + BC = AB + \bar{A}C$.

Solution:

A	B	C	AB	$\bar{A}C$	BC	$AB + \bar{A}C + BC$	$AB + \bar{A}C$
	0	0	0	0	0	0	0
0	0	1	0	1	0	1	1
0	1	0	0	0	0	0	0
0	1	1	0	1	1	1	1
1	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0
1	1	0	1	0	0	1	1
1	1	1	1	0	1	1	1

Looking at the table above, we can see that the columns for $AB + \overline{A}C + BC$ and $AB + \overline{A}C$ are identical so the two expressions are equivalent. This has been shown by means of a truth-table. *Proving Boolean expressions by use of truth-table is known as proof by perfect induction.*

2.5.2 Proof by Algebraic means

This requires a mastery of the laws of Boolean algebra so a lot of practice is needed to be able to use this technique effectively.

Example

Use algebraic means to show that $A + A \cdot B = A$

Solution:

$$\begin{aligned} A + A \cdot B &= A \cdot 1 + A \cdot B \\ &= A \cdot (1 + B) = A \cdot 1 = A \end{aligned}$$

Example

Use algebraic means to show that $A + \overline{A} \cdot B = A + B$

Solution:

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

2.6 Standard Forms for Boolean Functions

There are two standard forms for Boolean expressions: Standard sum of products form and Standard product of sums form.

2.6.1 Standard Sum of Products form

Given a function:

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

we can use the distributive rule (informally known as opening the brackets) to write:

$$f(A, B, C) = ABB + CB + ABAC + CAC$$

By use of Boolean rules, we can simplify the above expression to:

$$f(A, B, C) = AB + BC + ABC + AC$$

From the expression above, the terms AB , BC , ABC and AC are products, and they are all combined with an OR operation (logical addition or summation) so the expression is said to be in Sum of Products form. (Note however that expressions like $ABC + \overline{ABC}$ are not in Sum of Products form since the inversion signs cover more than one variable).

Now consider the expression we obtained above:

$$f(A, B, C) = AB + BC + ABC + AC$$

This is a function of variables A , B and C , but not all the product terms contain all these variables e.g. the product term AB lacks the variable C , BC lacks the variable A , and so on. To express the function in Standard Sum of Product form, we must add the missing variables to all the product terms so that every variable appears in each product term (either in its true form or in its complement form). To do this, we use the Boolean algebra laws:

$$(A + \overline{A}) = 1 \quad \text{and} \quad A \cdot 1 = A$$

We can then write the above expression as:

$$\begin{aligned} f(A, B, C) &= AB(C + \overline{C}) + (A + \overline{A})BC + ABC + AC(B + \overline{B}) \\ &= ABC + AB\overline{C} + ABC + \overline{A}BC + ABC + ABC + A\overline{B}C \\ &= ABC + AB\overline{C} + \overline{A}BC + A\overline{B}C \end{aligned}$$

This form, in which a sum of products appears, each term involving all variables is called the *Standard Sum of Products form* or *Canonical Sum of Products form*. Each individual term in the expression is known as a *minterm*, e.g. ABC is a minterm. Each minterm will have a logical value of 1 only when all the terms have a logical value of 1 e.g. minterm ABC will have a logical value of 1 only when $A = B = C = 1$, $AB\overline{C} = 1$ only if $A = B = \overline{C} = 1$ ($A = 1$, $B = 1$, and $C = 0$) e.t.c. The table below shows the minterms of the 3 variables A , B and C .

A	B	C	$minterm$
0	0	0	$m_0 = \overline{A}\overline{B}\overline{C}$
0	0	1	$m_1 = \overline{A}\overline{B}C$
0	1	0	$m_2 = \overline{A}B\overline{C}$
0	1	1	$m_3 = \overline{A}BC$
1	0	0	$m_4 = A\overline{B}\overline{C}$
1	0	1	$m_5 = A\overline{B}C$
1	1	0	$m_6 = AB\overline{C}$
1	1	1	$m_7 = ABC$

Going back to the function we started off with and using the above table, we can write:

$$f(A, B, C) = m_7 + m_6 + m_3 + m_5$$

Sometimes the above expression is written as:

$$f(A, B, C) = \Sigma m(3, 5, 6, 7)$$

2.6.2 Standard Product of Sums form

Given a function:

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

we can use the distributive rule to write:

$$\begin{aligned} f(A, B, C) &= (A + C)(B + C)(B + A)(B + C) \\ &= (A + B)(A + C)(B + C) \end{aligned}$$

The above expression is said to be in product of sums form. To convert this to the Standard product of sums form, we add the missing variables in each term, using the Boolean rules:

$$A \cdot \bar{A} = 0 \quad \text{and} \quad (A + 0) = A$$

We can therefore write:

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

Again using the distributive rule:

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

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

This is known as the *Standard Product of Sums form* or *Canonical Product of Sums form*. Each of the factors in the expression above is known as a *maxterm*, e.g. $A + B + C$ is a maxterm. Each maxterm will have a logical value 0 only when all the terms in it have a logical value 0, e.g. maxterm $(A + \bar{B} + C)$ will have a logical value 0 when $A = 0$, $B = 1$ and $C = 0$.

A	B	C	$maxterm$
0	0	0	$M_0 = A + B + C$
0	0	1	$M_1 = A + B + \bar{C}$
0	1	0	$M_2 = A + \bar{B} + C$
0	1	1	$M_3 = A + \bar{B} + \bar{C}$
1	0	0	$M_4 = \bar{A} + B + C$
1	0	1	$M_5 = \bar{A} + B + \bar{C}$
1	1	0	$M_6 = \bar{A} + \bar{B} + C$
1	1	1	$M_7 = \bar{A} + \bar{B} + \bar{C}$

We can therefore write:

$$f(A, B, C) = M_0 \cdot M_1 \cdot M_2 \cdot M_4$$

Sometimes this is written as:

$$f(A, B, C) = \Pi M(0, 1, 2, 4)$$

Chapter 3

SIMPLIFICATION OF BOOLEAN EXPRESSIONS AND COMBINATIONAL LOGIC CIRCUIT DESIGN

3.1 Simplification of Boolean Expressions

Introduction

Suppose you wanted to implement the Boolean function:

$$x = AB (\overline{\overline{A} + BC})$$

We can implement this directly as shown on Figure 3.1.

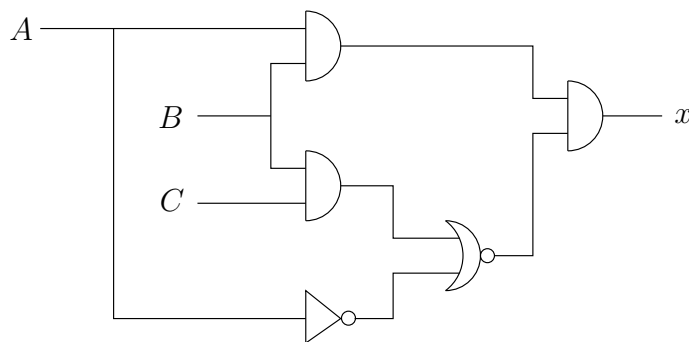


Figure 3.1: Circuit for the unsimplified expression

This implementation requires a total of five gates.

Suppose now we decided to simplify the expression before implementing it. We first use De Morgan's theorems to get:

$$x = AB (\overline{\overline{A} + BC})$$

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

This can be implemented using only two gates, as shown on Figure 3.2.

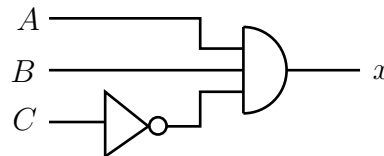


Figure 3.2: Circuit for the simplified expression

Generally, it is necessary to reduce Boolean expressions before implementing them as it makes the final circuit:

- i) cheaper - less gates used, needs a smaller circuit board.
- ii) more reliable as there are fewer interconnections.
- iii) have a lower power consumption.

There are two methods of simplifying logic expressions:

- i) Algebraic Simplification - Uses theorems of Boolean Algebra.
- ii) The Karnaugh map method - Graphical.

3.1.1 Algebraic Simplification

To use this method, you need to know the theorems of Boolean algebra very well - you will need a lot of practice to improve your skills. There are generally two steps when using this method:

- i) Put the expression in Sum of Products form (*not* Standard Sum of Products form). This may require the use of De Morgan's theorem or the distributive rules.
- ii) Check for common factors and factor out whenever possible. Factoring usually results in the elimination of some of the terms.

Example

Simplify algebraically:

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

The expression is already in sum of products form, so we shall just factor out the expression:

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

This expression is not in sum of products form, so we shall first apply De Morgan's Theorems to get:

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

Now the expression is in sum of products form, so we can proceed with the simplification:

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

Example

A student may register for course X only if he satisfies the following conditions:

- (1) Has completed at least 20 courses AND is an engineering student AND of good conduct, OR
- (2) Has completed at least 20 courses AND is an engineering student AND has departmental approval, OR
- (3) Has completed fewer than 20 courses AND is an engineering student AND not of good conduct, OR
- (4) Is of good conduct AND has departmental approval, OR

(5) Is an engineering student AND does not have departmental approval.

We can convert the conditions listed to letter symbols as follows:

A: Has completed at least 20 courses

B: Is an engineering student

C: Is of good conduct

D: Has departmental approval

Y: Student may register for course X

We can then write:

$$\begin{aligned}
 Y &= ABC + ABD + \overline{A}B\overline{C} + CD + B\overline{D} \\
 &= ABC + B(\overline{D} + AD) + \overline{A}B\overline{C} + CD + B\overline{D} \\
 &= ABC + B(\overline{D} + A\overline{D}) + \overline{A}B\overline{C} + CD + B\overline{D} \\
 &= (ABC + AB\overline{D}) + B\overline{D} + \overline{A}B\overline{C} + CD \\
 &= AB + ABC + B\overline{D} + CD \\
 &= B(\overline{A} + A) + B\overline{D} + CD \\
 &= B + B\overline{D} + CD
 \end{aligned}$$

Recall the theorem:

$$AB + \overline{A}C + BC = AB + \overline{A}C$$

We can use this theorem to rewrite the expression in brackets above as:

$$\overline{B}D + CD = \overline{B}D + CD + BC$$

Hence:

$$\begin{aligned}
 Y &= AB + B\overline{C} + B\overline{D} + CD + BC \\
 &= AB + B(\overline{C} + \overline{D}) + B\overline{D} + CD \\
 &= AB + B + B\overline{D} + CD \\
 &= B + B\overline{D} + CD \\
 &= B + CD
 \end{aligned}$$

Hence a student may register for the course X if he is an engineering student OR he is of good conduct AND has departmental approval.

3.1.2 The Karnaugh Map

Introduction

This is a graphical method used to simplify a Boolean expression. It represents the information in a truth-table in a different format. Each combination of inputs is represented by a cell in the map.

Once a Karnaugh Map (K-map) has been filled with ones and zeros, the sum of products expression can be obtained by ORing together the those squares that contain 1s. The product of sums expression can be obtained by ANDing together those squares that contain 0s.

Two variable K-map

Consider the 2-variable truth-table shown below:

A	B	x	minterm
	0	1	m_0
0	1	1	m_1
1	0	0	m_2
1	1	0	m_3

(This truth-table is arbitrarily chosen - it is for purposes of illustration only). There are four input combinations, so this truth-table can be converted to a K-map with 4 cells, as shown on Figure 3.3. Note that in this case, variable A is treated as the MSB.

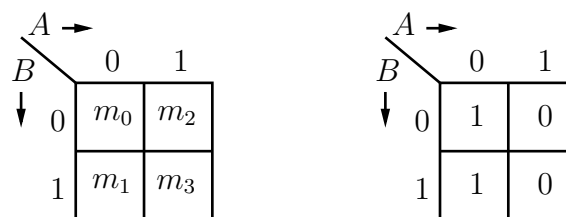


Figure 3.3: Two variable K-map

The K-map may be alternatively drawn with the variable A on the vertical side and variable B on the horizontal side, as shown on Figure 3.4:

3-variable K-map

Consider the 3-input truth-table shown below:

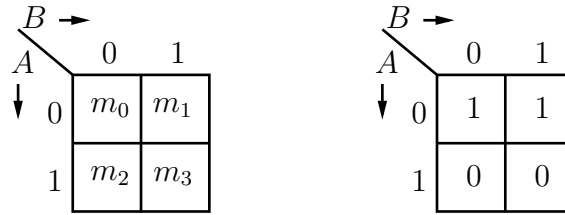


Figure 3.4: Two variable K-map: alternative representation

A	B	C	x	minterm
0	0	0	1	m_0
0	0	1	1	m_1
0	1	0	1	m_2
0	1	1	0	m_3
1	0	0	0	m_4
1	0	1	0	m_5
1	1	0	1	m_6
1	1	1	0	m_7

This can be represented using a K-map as shown on Figure 3.5.

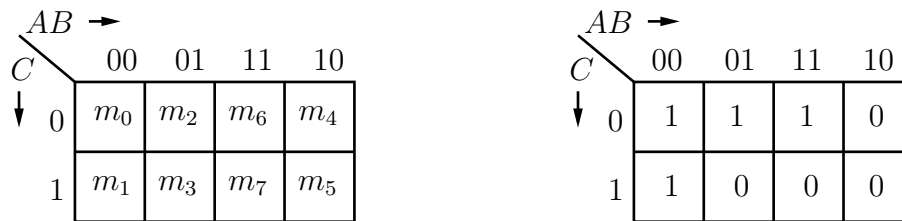


Figure 3.5: Three variable K-map

Note that the K-map cells are labelled in such a way that *adjacent cells differ only in one variable*.

The 3-variable K-map may be alternatively drawn as shown on Figure 3.6.

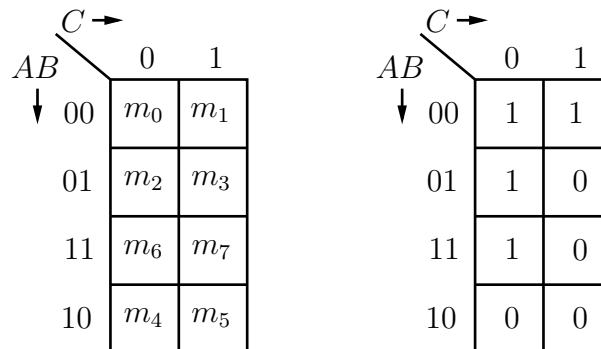


Figure 3.6: Three variable K-map: alternative representation

Note that the two representations shown (Figure 3.5 and Figure 3.6) are equivalent, and you may work with whichever representation that you are more comfortable

with. The only thing you should keep in mind is the *order in which the variables appear* – in this case, they appear in the order A (MSB), B and C (LSB).

4-variable K-map

Consider the 4-input truth-table shown below:

A	B	C	D	x	minterm
0	0	0	0	0	m_0
0	0	0	1	1	m_1
0	0	1	0	0	m_2
0	0	1	1	0	m_3
0	1	0	0	0	m_4
0	1	0	1	1	m_5
0	1	1	0	0	m_6
0	1	1	1	0	m_7
1	0	0	0	0	m_8
1	0	0	1	0	m_9
1	0	1	0	0	m_{10}
1	0	1	1	0	m_{11}
1	1	0	0	0	m_{12}
1	1	0	1	1	m_{13}
1	1	1	0	0	m_{14}
1	1	1	1	1	m_{15}

This may be represented using a K-map as shown on Figure 3.7.

		AB →			
		00	01	11	10
CD ↓	00	m_0	m_4	m_{12}	m_8
	01	m_1	m_5	m_{13}	m_9
	11	m_3	m_7	m_{15}	m_{11}
	10	m_2	m_6	m_{14}	m_{10}

		AB →			
		00	01	11	10
CD ↓	00	0	0	0	0
	01	1	1	1	0
	11	0	0	1	0
	10	0	0	0	0

Figure 3.7: Four variable K-map

The K-map may be alternatively drawn as shown on Figure 3.8.

3.1.3 Looping

The Logic function can be simplified by properly combining those squares in the that contain 1s. The process of combining 1s is called looping. The number of 1s that can be looped together should be a *power of 2* (1, 2, 4, 8, 16 e.t.c.).

		$CD \rightarrow$			
		00	01	11	10
$AB \downarrow$	00	m_0	m_1	m_3	m_2
	01	m_4	m_5	m_7	m_6
	11	m_{12}	m_{13}	m_{15}	m_{14}
	10	m_8	m_9	m_{11}	m_{10}

		$CD \rightarrow$			
		00	01	11	10
$AB \downarrow$	00	0	1	0	0
	01	0	1	0	0
	11	0	1	1	0
	10	0	0	0	0

Figure 3.8: Four variable K-map: alternative representation

Groups of 2 (Pairs)

Two ones can be looped together if they are *horizontally* or *vertically* adjacent. Two ones next to each other diagonally are not adjacent. Looping a pair of adjacent 1s in a K-map eliminates the variable that appears in the complemented and uncomplemented form. Variables that are the same for all cells of the loop must appear in the final expression.

Example

Consider the K-map shown on Figure 3.9.

		$AB \rightarrow$			
		$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$C \downarrow$	\bar{C}	0	1	1	0
	C	0	0	0	0

Figure 3.9: Looping two 1s which are horizontally adjacent

The two 1s shown in the K-map are horizontally adjacent and can be looped together as a pair as shown in the figure. Looking at the cells enclosed in the loop, we can see that variable A appears as \bar{A} in one cell (complemented form) and as A (uncomplemented form) in the other cell hence it is eliminated. Variable B appears as B in both cells and variable C appears as \bar{C} so the simplified expression is:

$$x = B\bar{C}$$

Example

Consider the K-map shown on Figure 3.10.

The two ones are vertically adjacent and the variable C is the one that changes and is eliminated hence:

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

A Karnaugh map with columns labeled $\bar{A}\bar{B}$, $\bar{A}B$, AB , and $A\bar{B}$ under the heading $AB \rightarrow$. The rows are labeled \bar{C} and C under the heading $C \downarrow$. The cells contain the following values:

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
\bar{C}	0	1	0	0
C	0	1	0	0

The two 1s in the second column ($\bar{A}B$) are circled with a red rectangle, indicating a vertical loop.

Figure 3.10: Looping two 1s which are vertically adjacent

The leftmost column and the rightmost column of the K-map are considered to be adjacent. Similarly, the top row and the bottom row of a K-map are considered to be adjacent.

Example

Consider the K-map shown on Figure 3.11.

A Karnaugh map with columns labeled $\bar{A}\bar{B}$, $\bar{A}B$, AB , and $A\bar{B}$ under the heading $AB \rightarrow$. The rows are labeled \bar{C} and C under the heading $C \downarrow$. The cells contain the following values:

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
\bar{C}	1	0	0	1
C	0	0	0	0

The 1s in the first and fourth columns of the top row (\bar{C}) are circled with a red rectangle, indicating a horizontal loop.

Figure 3.11: Looping two 1s which are horizontally adjacent

In this case,

$$x = \bar{B}\bar{C}$$

Groups of 4 (Quads)

Four 1s can be looped together if they are horizontally adjacent, vertically adjacent or form a square. A loop of four 1s eliminates 2 variables that appear in both complemented and uncomplemented form.

Example

Consider the K-map shown on Figure 3.12.

A Karnaugh map with columns labeled $\bar{A}\bar{B}$, $\bar{A}B$, AB , and $A\bar{B}$ under the heading $AB \rightarrow$. The rows are labeled \bar{C} and C under the heading $C \downarrow$. The cells contain the following values:

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
\bar{C}	0	0	0	0
C	1	1	1	1

The four 1s in the bottom row (C) are circled with a red rectangle, indicating a horizontal loop of four 1s.

Figure 3.12: Looping four 1s which are horizontally adjacent

The four 1s are horizontally adjacent and are looped together to give:

$$x = C$$

Example

Consider the K-map shown on Figure 3.13.

		$AB \rightarrow$			
$CD \downarrow$	$\bar{C}\bar{D}$	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
	$\bar{C}D$	0	1	0	0
	$C\bar{D}$	0	1	0	0
	CD	0	1	0	0
	$C\bar{D}$	0	1	0	0

Figure 3.13: Looping four 1s which are vertically adjacent

The four 1s are vertically adjacent and are looped together to give:

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

Example

		$AB \rightarrow$			
$CD \downarrow$	$\bar{C}\bar{D}$	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
	$\bar{C}D$	0	1	1	0
	$C\bar{D}$	0	1	1	0
	CD	0	0	0	0
	$C\bar{D}$	0	0	0	0

Figure 3.14: Looping four 1s which form a square

The four 1s in Figure 3.14 form a square and are looped together to give:

$$x = B\bar{C}$$

Other examples of quads are shown on Figure 3.15, where $x = \bar{B}D$ for the K-map on

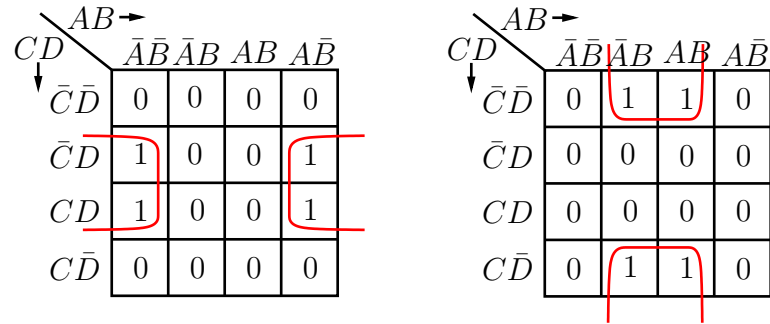


Figure 3.15: Looping four 1s

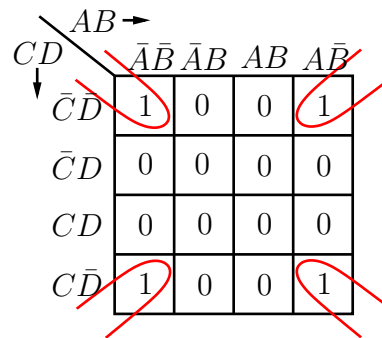


Figure 3.16: The 1s at the corners are adjacent and can be looped together

Since the top row of a K-map is adjacent to the bottom row, and the right column to the left, the corner cells of a K-map are also considered adjacent and can be looped together if they all contain 1s, as shown on Figure 3.16.

In this case, $x = \bar{B}\bar{D}$.

Groups of 8 (octets)

Eight ones may be looped together if they are adjacent. A loop of eight 1s eliminates 3 variables. Examples of octets are shown on Figure 3.17.

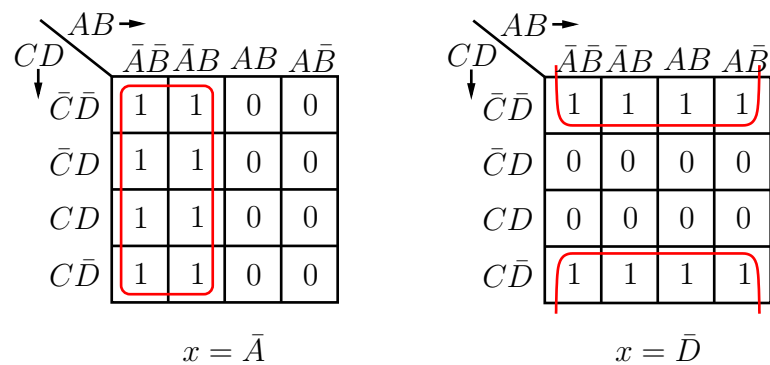


Figure 3.17: Looping eight 1s (octets)

3.1.4 Complete Simplification Procedure

1. Construct the K-map and place 1s in those cells corresponding to 1s in the truth-table. Place 0s in the other squares.
2. Examine the map and loop those 1s which are not adjacent to any other 1s. these are called *isolated 1s*.
3. Identify those 1s that are adjacent to *only one other 1*. Loop any pair containing such a 1. Adjacent ones which can be combined in more than one way are temporarily bypassed.
4. Identify those 1s which can be combined with three other 1s *in only one way*. If not all four 1s so involved have already been looped as pairs, loop the four 1s. The 1s that can be looped in a group of four in more than one way are temporarily bypassed.
5. Repeat the preceding steps for groups of 8, 16 e.t.c.
6. Loop any quad that contains one or more 1s that have not yet been looped.
7. Loop any pairs necessary to include any 1s that have not yet been looped, making sure to use the minimum number of loops.
8. Form the OR sum of all the terms generated by each loop.

Note: *If the expression you obtain using the steps above can be simplified further by algebraic means, it means you have not looped properly - you may be using too many loops and/or your loops may not be large enough.*

Carefully go through the examples below, and see if you can come up with the same loops. These examples have been looped in such a way as to obtain the simplest possible expressions.

Example

From Figure 3.18, $x = \bar{A}\bar{B}\bar{C}\bar{D} + ABC + BD$

Example

From Figure 3.19, $x = \bar{A}\bar{B}\bar{C} + A\bar{C}D + ABC + \bar{A}CD$

Example

From Figure 3.20, $x = \bar{A}BC\bar{D} + \bar{A}\bar{B}\bar{C} + AB\bar{C} + \bar{C}D + \bar{B}D + AD$

$AB \rightarrow$ $CD \downarrow$		$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
		$\bar{C}\bar{D}$	0	0	0
	$\bar{C}D$	0	1	1	0
	CD	0	1	1	0
	$C\bar{D}$	0	0	1	0

Figure 3.18: Example

$AB \rightarrow$ $CD \downarrow$		$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
		$\bar{C}\bar{D}$	0	1	0
	$\bar{C}D$	0	1	1	1
	CD	1	1	1	0
	$C\bar{D}$	0	0	1	0

Figure 3.19: Example

$AB \rightarrow$ $CD \downarrow$		$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
		$\bar{C}\bar{D}$	1	0	1
	$\bar{C}D$	1	1	1	1
	CD	1	0	1	1
	$C\bar{D}$	0	1	0	0

Figure 3.20: Example

3.1.5 Obtaining Product of Sums Expressions from K-maps

So far, we have talked about obtaining Sum of Products expressions from K-maps. It is also possible to obtain Product of Sums expressions – only this time we loop the zeros together, not the ones. The examples below illustrate the procedure.

Example

From Figure 3.21, $x = (\bar{B} + \bar{C} + \bar{D})(A + \bar{D})$

Example (

		AB →			
CD ↓		00	01	11	10
	00	1	1	1	1
	01	0	0	1	1
	11	0	0	0	1
	10	1	1	1	1

Figure 3.21: Example: looping the 0s to obtain a product of sums expression

		AB →			
CD ↓		00	01	11	10
	00	0	0	1	1
	01	1	0	0	1
	11	0	0	0	0
	10	1	0	0	1

Figure 3.22: Example: looping the 0s to obtain a product of sums expression

From Figure 3.22, $x = (A + C + D) (\bar{B} + \bar{D}) (\bar{C} + \bar{D}) (\bar{B} + \bar{C})$

Note that for a given K-map, looping the 1s and looping the zeros gives the same results, only that the results are expressed in different ways. Consider the example shown on Figure 3.23 where in one case the 1s are looped and in the other case the 0s are looped.

		AB →			
CD ↓		00	01	11	10
	00	0	0	0	0
	01	0	1	1	1
	11	0	1	1	1
	10	0	1	1	1

		AB →			
CD ↓		00	01	11	10
	00	0	0	0	0
	01	0	1	1	1
	11	0	1	1	1
	10	0	1	1	1

Figure 3.23: Example: looping 0s or 1s gives same result expressed in different ways

The Sum of Products expression obtained by looping the 1s is:

$$x = AD + BD + BC + AC$$

while the Product of Sums expression obtained by looping 0s is:

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

Opening the brackets of the Product of Sums expression will yield the Sum of Products expression obtained by looping the 1s, showing the two expressions are equivalent.

3.2 Don't Care Terms

These are also referred to as Unused terms, Forbidden terms or Redundant terms. These terms describe combinations of variables which never occur in practice. In a truth-table or a K-map, these inputs are represented by an X , an R or a d . As an example, suppose that we have a digital circuit with three inputs and one output, and the input combinations 000, 001 and 010 give an output 0, input combinations 101, 110 and 111 give an output 1, and input combinations 011 and 100 never occur in practice. The truth-table for this circuit is as shown below:

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

When designing with K-maps containing don't care variables, the designer can make the output of any don't care condition either a 1 or a 0 in order to produce the simplest output expression. This is illustrated on Figure 3.24 for the truth-table above.

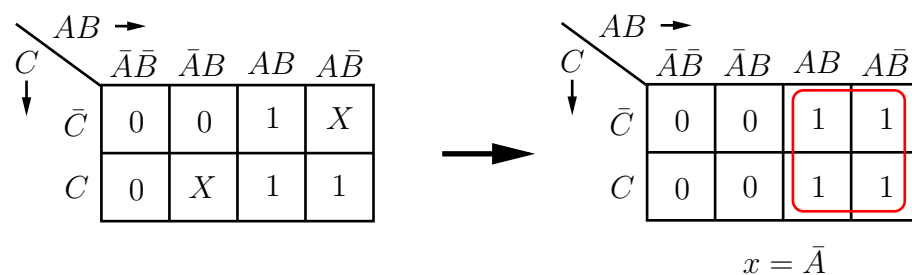


Figure 3.24: Example: simplifying a Boolean expression with don't care terms

3.3 NAND/NOR gate circuit implementation

To implement a logic circuit using NAND gates only:

1. Derive the minimized expression for the function in *sum of products form* (obtained by minimizing Boolean expressions algebraically or by looping 1s in a K-map).
2. Apply double negation and De Morgan's theorem to convert the expression in a form suitable for NAND gate implementation.

As an example, suppose a design problem resulted in a *minimized sum of products* expression:

$$x = AB + BC + AC$$

and we were to implement this expression using NAND gates only, we then apply the above steps as follows:

$$\begin{aligned} x &= AB + BC + AC = \overline{\overline{AB + BC + AC}} \\ &= \overline{AB \cdot BC \cdot AC} \end{aligned}$$

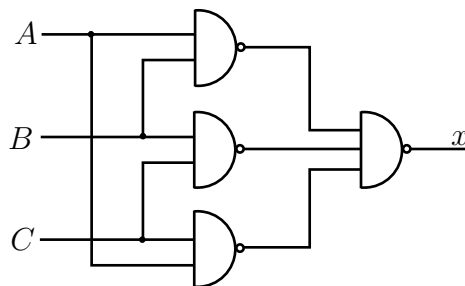


Figure 3.25: Example: implementation using NAND gates only

To implement a logic circuit using NOR gates only:

1. Derive the minimized expression for the function in *product of sums form* (obtained by looping the 0s in a K-map).
2. Apply double negation and De Morgan's theorem to convert the expression in a form suitable for NOR gate implementation.

As an example, suppose a design problem resulted in a *minimized product of sums* expression:

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

and we were to implement this expression using NOR gates only, we then apply the above steps as follows:

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

which is implemented as shown on Figure 3.26.

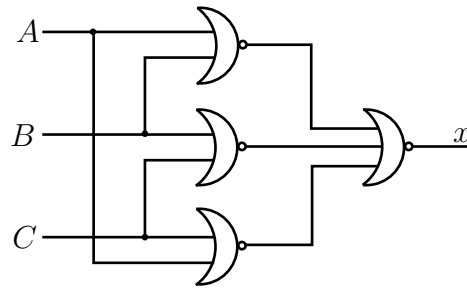


Figure 3.26: Example: implementation using NOR gates only

3.4 SSI IC-BASED Combinational Logic Circuit Design

3.4.1 Introduction

SSI Small Scale Integration - Digital ICs with less than 12 gates

MSI Medium Scale Integration - Digital ICs with 12 to 99 gates

LSI Large Scale Integration - Digital ICs with 100-9999 gates

VLSI Very Large Scale Integration - More than 10000 gates

A Combinational logic circuit is a logic circuit whose outputs are functions of the present inputs only. It cannot 'remember' the effects of the previous inputs.

3.4.2 Design Procedure

1. Derivation of the truth-table.
 - Understand the problem
 - Define the input variables
 - Define the output variables
 - Relate the output variables to the input variables using a truth-table

2. Derivation of Boolean expressions from the truth-table.
 - Outputs are expressed as functions of the input variables.
 3. Minimization (or Simplification) of the expressions for outputs.
 - Done to minimize the number of gates used in the design and hence minimize costs, reduce power consumption and increase circuit reliability.
- Note: Use of a K-map makes it possible to combine steps 2 and 3 above.
4. Conversion of the minimized expressions to the form that allows the implementation of the circuit using the available gates.
 - In some cases, we might want to implement the circuit using AND-OR-NOT logic, NAND gates only or NOR gates only. The expression(s) obtained at step 3 can be implemented directly using AND-OR-NOT logic (using a combination of AND, OR and NOT logic gates). However, AND-OR-NOT logic design requires three different types of ICs, all of which may not be available at the time of the design, and in many cases, AND-OR-NOT logic design leads to the use of a large number of logic gates, so it is usually preferred to implement the circuits using NAND gates only or NOR gates only. The expression obtained in step 3 then needs to be converted to a form suitable for implementation using these gates.
 5. Implementation of the circuit.

3.4.3 Examples and exercises

Example

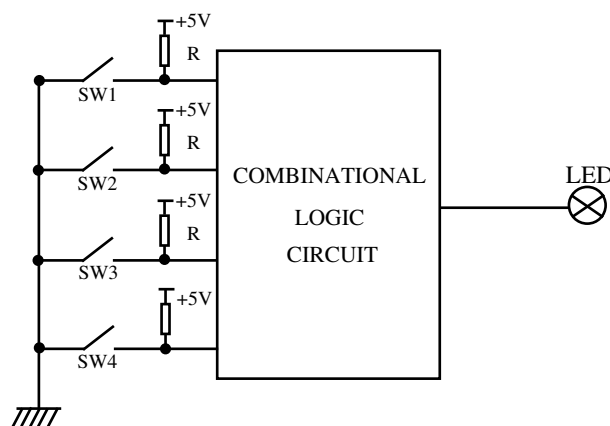


Figure 3.27: Example: combinational logic circuit design

Figure 3.27 shows four switches that are part of a control circuitry in a copy machine. The switches are at various points along the path of the copy paper as the paper passes through the machine. Each switch is normally open and as the paper passes over a switch, the switch closes. It is impossible for switches SW1 and SW4 to be closed at the same time (they are far apart and the paper cannot cover them at the same time). The LED is to light if two or more switches are closed. Design a combinational logic circuit for the system.

Solution

The inputs in this case are switches SW1, SW2, SW3 and SW4. We shall denote these switches as A , B , C and D respectively. Note that if a switch is open, the corresponding *input* to the circuit is HIGH (Logic 1), and if a switch is closed, the corresponding *input* is LOW (Logic 0). The output of the circuit is denoted as z (and the state of this output is visually indicated by the LED): when two or more switches closed, the output of the circuit should be HIGH (LED is LIT), otherwise the output should be 0 (LED NOT LIT). The truth-table for the circuit is shown below:

A	B	C	D	z
0	0	0	0	X
0	0	0	1	1
0	0	1	0	X
0	0	1	1	1
0	1	0	0	X
0	1	0	1	1
0	1	1	0	X
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

The K-map corresponding to this is shown on Figure 3.28.

$AB \rightarrow$		$CD \downarrow$			
		$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}\bar{D}$	X	X	1	1	
$\bar{C}D$	1	1	0	1	
CD	1	0	0	0	
$C\bar{D}$	X	X	0	1	

Figure 3.28: K-map corresponding to the truth table

Figure 3.29 shows the best way to convert the don't care variables to 1s and 0s, achieving the biggest loops possible, and minimizing the number of loops used. The loops have been made to obtain the minimized sum of products expression.

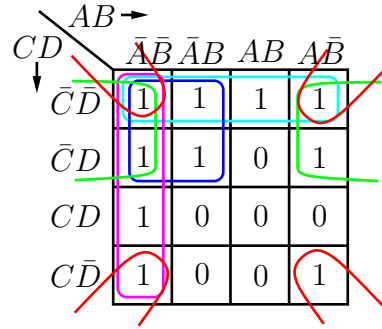


Figure 3.29: Simplification of the Boolean function

From Figure 3.29, we can write:

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

For NAND gate implementation,

$$\begin{aligned} z &= \overline{(\bar{B}\bar{D} + \bar{C}\bar{D} + \bar{A}\bar{B} + \bar{A}\bar{C} + \bar{B}\bar{C})} \\ &= \overline{(\bar{B}\bar{D} \cdot \bar{C}\bar{D} \cdot \bar{A}\bar{B} \cdot \bar{A}\bar{C} \cdot \bar{B}\bar{C})} \end{aligned}$$

For implementation using NOR gates only, a simplified product of sums expression is required, and this is obtained by looping the zeros as shown on Figure 3.30.

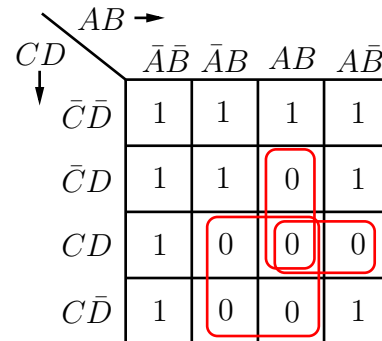


Figure 3.30: Looping the 0s to obtain a simplified product of sums expression

From the figure,

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

Example

Design a circuit to convert (4-bit) BCD code to Xs-3 code.

Solution

The truth-table for this is shown below:

BCD CODE				EXCESS-3 CODE			
<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>W</i>	<i>X</i>	<i>Y</i>	<i>Z</i>
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

Do you understand how to derive the truth-table above? Complete the rest of the exercise.

Exercise

A majority function combinational logic circuit is a circuit whose output is equal to 1 if the input variables have *more ones than zeros*. The output is zero otherwise. Design a 4-input majority function combinational logic circuit and implement the circuit using NAND gates only.

Exercise

In a certain corporation, the four board members *A*, *B*, *C* and *D* own all stock, which is distributed as follows:

A: 40%

B: 30%

C: 20%

D: 10%

Each member has a percentage vote equal to his holdings and a total vote *greater than 50%* is required to pass a motion. In the boardroom, each member is to have a switch with which to indicate a YES or NO vote. A lamp is to light if the total vote cast is *more than 50%* indicating the motion being voted on is passed. Design an electronic voting system for the corporation and implement the circuit using:

i) NAND gates only.

ii) NOR gates only.

Exercise

Figure 3.31 shows a diagram for an automobile circuit used to detect certain undesirable conditions. The 4 switches D, I, L and S are used to indicate the status of the driver's door, the ignition, the headlights and the driver's seatbelt respectively. The LED is to light under the following undesirable conditions:

- (i) The headlights are ON while the ignition is OFF
- (ii) The door is OPEN while the ignition is ON
- (iii) The seatbelt is UNFASTENED while the ignition is ON.

Under any of these undesirable conditions, the logic circuit should produce a HIGH output z to light the LED. Design a logic circuit to light the LED when an undesirable condition occurs, and implement the circuit using

- NAND gates only.
- NOR gates only.

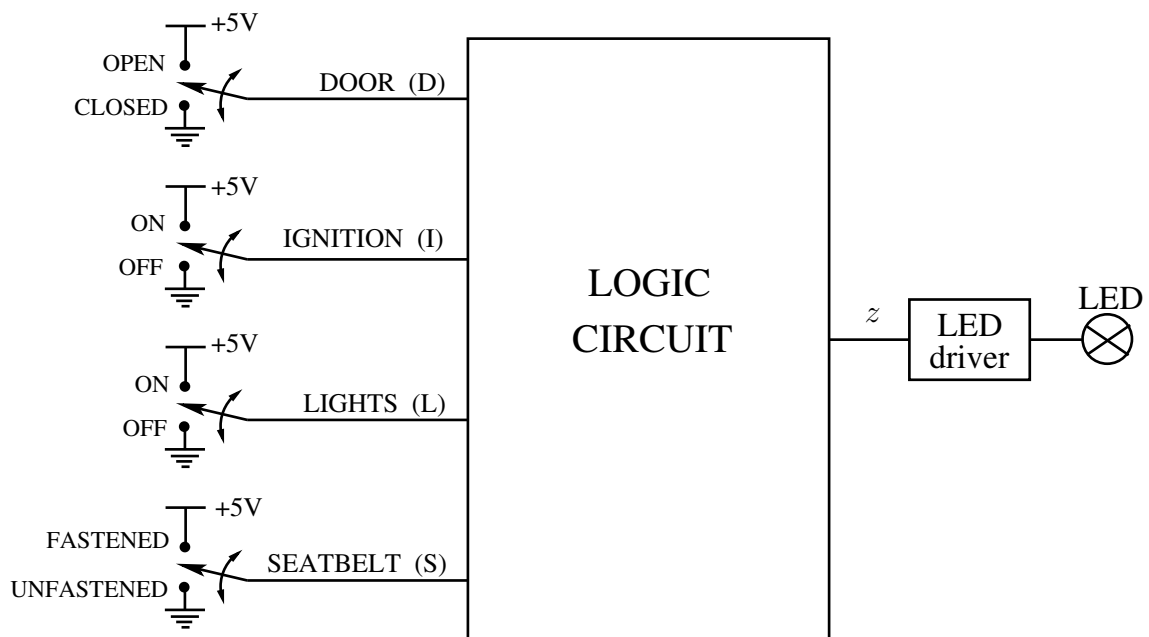


Figure 3.31:

Chapter 4

SEQUENTIAL LOGIC CIRCUITS

4.1 Introduction

So far, we have dealt with combinational logic circuits, whose outputs are functions of the current inputs only. This chapter deals with sequential logic circuits. A sequential logic circuit is a circuit whose present outputs are functions of the present inputs, as well as previous inputs. It has in it a unit called the memory which stores the effect of the previous sequence of inputs. The stored effects of the previous inputs are called the STATE of the circuit.

A sequential circuit can be represented by the block diagram on Figure 4.1.

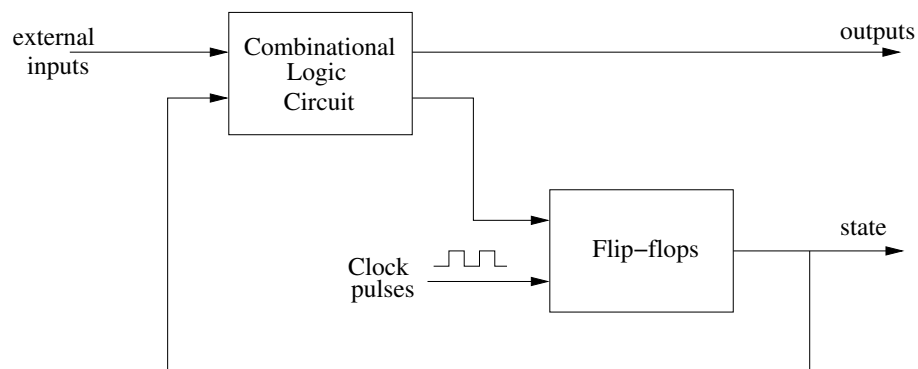


Figure 4.1: Block diagram of a sequential logic circuit

4.2 Flip-Flops

A flip-flop is a logic circuit that is capable of storing one bit of information (0 or 1). It stores the one bit of information as long as power is supplied to the circuit. It is the

simplest memory element. It is also referred to as a bistable multivibrator. Flip-flops are the basic building blocks for sequential logic circuits. The block diagram of a flip-flop is shown on Figure 4.2.

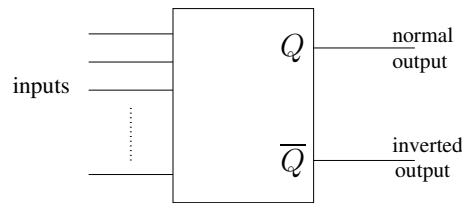


Figure 4.2: A flip-flop

A Flip-flop can have one or more inputs, but it has only two outputs, the normal output Q and the inverted (or complemented) output \overline{Q} . Under normal operating conditions Q and \overline{Q} are always complements of each other, i.e. either $Q = 0$ and $\overline{Q} = 1$, or $Q = 1$ and $\overline{Q} = 0$.

When we refer to the STATE of a flip-flop, we are referring to the state of its normal (Q) output i.e. if we say that the state of a flip-flop is 1, we mean $Q = 1$.

4.2.1 The NAND-gate latch

The NAND-gate latch is the simplest flip-flop. It is also referred to as a bistable latch. It is illustrated on Figure 4.3.

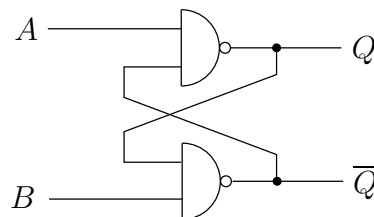


Figure 4.3: A NAND gate latch

The truth-table for the NAND-gate latch is shown below (Q_n stands for present state, while Q_{n+1} stands for next state):

A	B	Q_n	Q_{n+1}	\bar{Q}_{n+1}
0	0	0	1	1
0	0	1	1	1
0	1	0	1	0
0	1	1	1	0
1	0	0	0	1
1	0	1	0	1
1	1	0	0	1
1	1	1	1	0

The truth-table can be summarized as shown below:

A	B	Q_{n+1}
0	0	Disallowed as it causes $Q = \bar{Q} = 1$
0	1	1
1	0	0
1	1	Q_n - 'remembers' previous state (Memory State)

A NAND-gate latch is used as a building block for other more complicated flip-flops, and for debouncing switches. Figure 4.4 shows how the NAND gate latch can be used to debounce a switch.

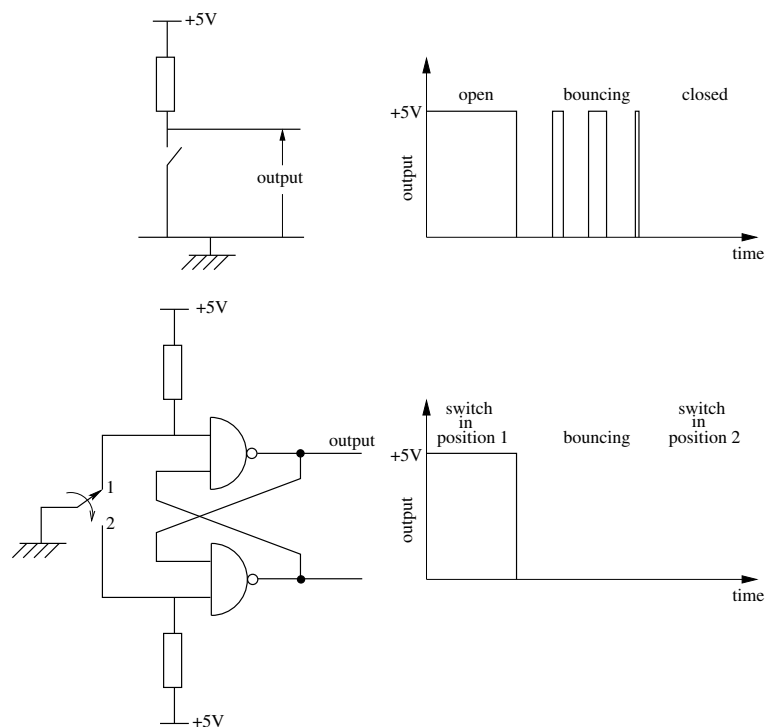


Figure 4.4: Switch debouncing

4.2.2 The SET-RESET flip-flop

This is illustrated on Figure 4.5.

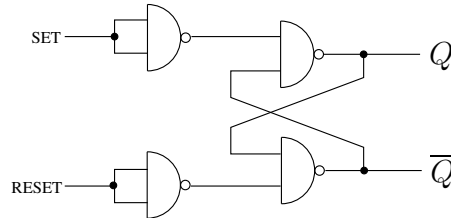


Figure 4.5: SET-RESET (SR) flip-flop

The truth-table for this flip-flop is shown below:

<i>SET</i>	<i>RESET</i>	Q_{n+1}
0	0	Q_n - 'remembers' previous state (Memory State)
0	1	0 - Flip-Flop RESET
1	0	1 - Flip-Flop SET
1	1	Disallowed as it causes $Q = \bar{Q} = 1$

When we have $SET = 0$ and $RESET = 1$, the flip-flop is RESET ($Q = 0$), and when $SET = 1$ and $RESET = 0$, the flip-flop is SET ($Q = 1$). If we have $SET = 1$ and $RESET = 1$, it is similar to setting and resetting the flip-flop at the same time, and this mode is disallowed as it causes $Q = \bar{Q} = 1$.

4.2.3 The D Flip-Flop

The D-type flip-flop is illustrated in Figure 4.6.

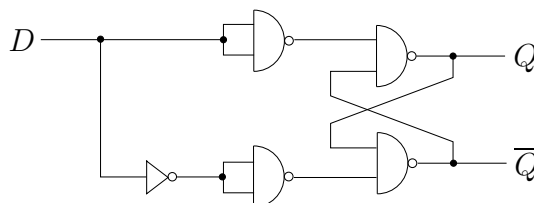


Figure 4.6: D flip-flop

It is a modified version of the *SET-RESET* flip-flop where $SET = \overline{RESET}$. It has only one input, D . The truth-table for this flip-flop is shown below:

D	Q_{n+1}
0	0
1	1

From the truth-table, we can see that Q transparently follows the input D , and for this reason, the D flip-flop is sometimes referred to as a transparent latch.

4.2.4 The T flip-flop

A T-flip-flop (also known as a toggle flip-flop) is illustrated on Figure 4.7.

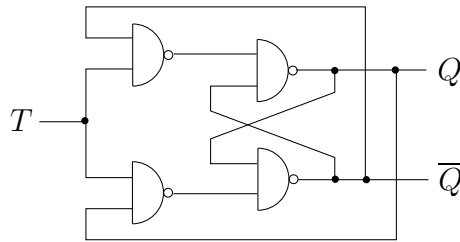


Figure 4.7: T flip-flop

The corresponding truth-table is shown below:

T	Q_n	Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

This truth-table can be summarized as shown below:

T	Q_{n+1}
0	Q_n
1	\bar{Q}_n

4.2.5 The JK flip-flop

The JK flip-flop is shown on Figure 4.8.

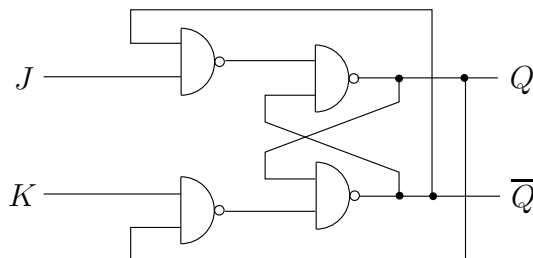


Figure 4.8: JK flip-flop

Its truth-table is shown below:

J	K	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

This truth-table can be summarized as shown below:

J	K	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	\bar{Q}_n

From the truth-table above, we can see that the JK flip-flop does not have invalid inputs and it can complement. These properties make the JK flip-flop very versatile, and it is used in many sequential logic circuits.

4.3 Clock signals and clocked flip-flops

4.3.1 Introduction

Clock waveforms

A clock signal is a *rectangular pulse train* or a *square wave*. A clock waveform can have many shapes, as shown on Figure 4.9.

Duty cycle

For a clock waveform,

$$\text{Duty Cycle} = \frac{T_{HIGH}}{T_{HIGH} + T_{LOW}} \times 100\%$$

where T_{HIGH} is the time the waveform is at logic level 1 and T_{LOW} is the time the waveform is at logic level 0.

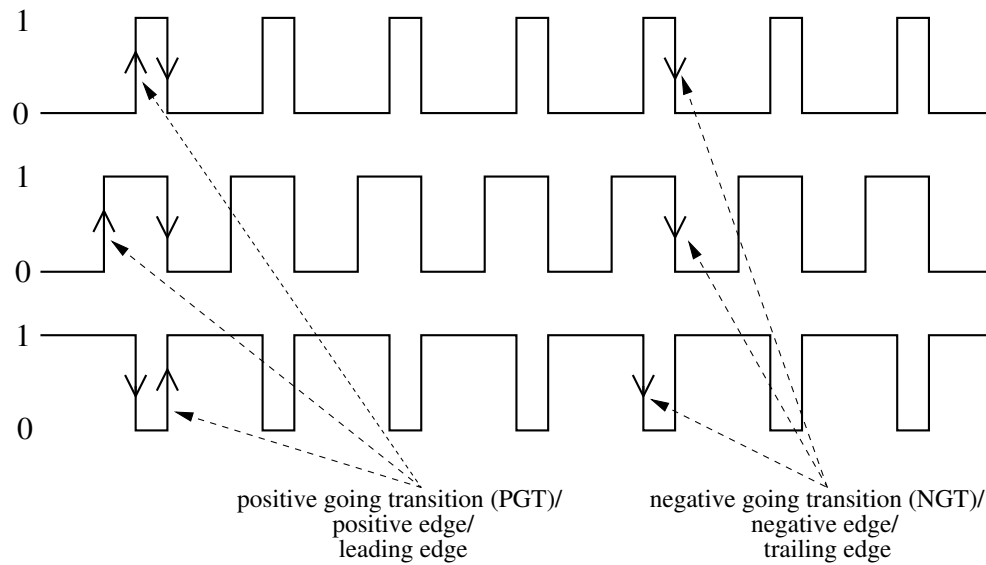


Figure 4.9: Clock waveforms

Synchronous and asynchronous operation

Digital systems can operate either synchronously or asynchronously:

Asynchronous Outputs of a logic circuit change any time one or more of the inputs change. This kind of circuits are difficult to troubleshoot since the outputs can change at any time.

Synchronous Outputs can only change at specific instants of time, these instants determined by the clock.

Types of clocked flip-flops

There are three types of clocked flip-flops:

- Level Driven flip-flops
- Master-Slave flip-flops
- Edge-triggered flip-flops

The first two types are no longer used in modern digital systems, but are covered here for completeness.

4.3.2 Level-Driven flip-flops

A Level-driven flip-flop is one where one level of the clock enables the data inputs to affect the state of the flip-flop, whereas the other level disables the data inputs from affecting the state of the flip-flop. This type of flip-flop is illustrated using a JK flip-flop, which is shown in Figure 4.10.

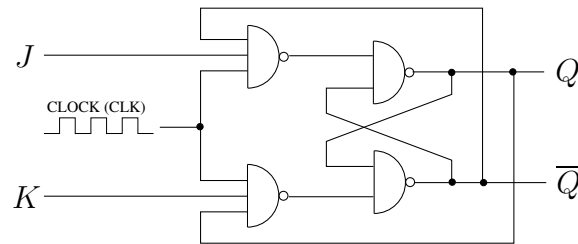


Figure 4.10: Level-driven JK flip-flop

From the figure, we can see that when the CLK signal is at logic level 0, $X = Y = 1$, and from the truth-table of the NAND gate latch, we can see that the flip-flop will be in the memory state, hence the inputs J and K will not have any effect on the outputs Q and \bar{Q} . This means that the inputs J and K are disabled from affecting the state of the flip-flop when $CLK = 0$.

When the CLK signal is HIGH, $X = (\overline{J \cdot \bar{Q}})$ and $Y = (\overline{K \cdot Q})$, hence the next state of the flip-flop is determined by the inputs J and K. This means that when $CLK = 1$, the flip-flop inputs (J and K) are enabled to affect the state of the flip-flop.

In this case, we say that the enabling level of the clock is HIGH (Logic 1).

Figure 4.11 below shows the logic symbols for level driven JK flip-flops, one case having the enabling level of the clock as HIGH as in the case above, and the other case with the enabling level of the clock being LOW.

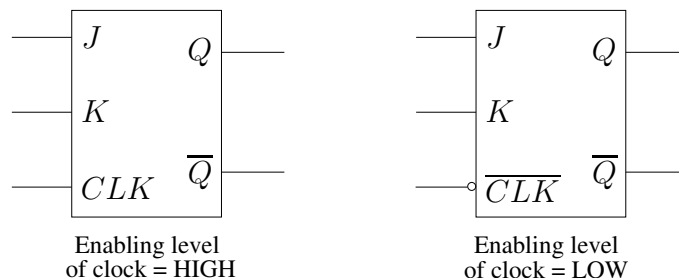


Figure 4.11: Symbols of level-driven JK flip-flops

Exercise

Draw the circuit diagrams for level-driven SET-RESET flip-flop, D flip-flop and the T flip-flop and explain their operation.

4.3.3 The Master-Slave flip-flop

The block diagram for a Master-Slave¹ flip-flop is shown on Figure 4.12.

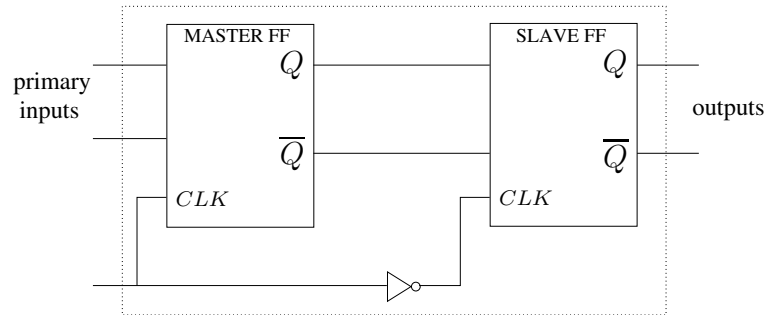


Figure 4.12: A master-slave flip-flop

Assume that the basic flip-flops are enabled by the HIGH level of the clock:

CLK = HIGH Primary inputs are enabled to determine the next state of the Master flip-flop. Clock input to the Slave flip-flop is LOW so that the Slave inputs are disabled from affecting the state of the Slave flip-flop.

CLK = LOW Primary inputs are disabled from affecting the state of the Master flip-flop. Clock input to the Slave flip-flop is HIGH so that the Master's outputs are transferred to the Slave flip-flop to determine the next state of the Slave flip-flop.

Figure 4.13 shows a JK Master-Slave flip-flop.

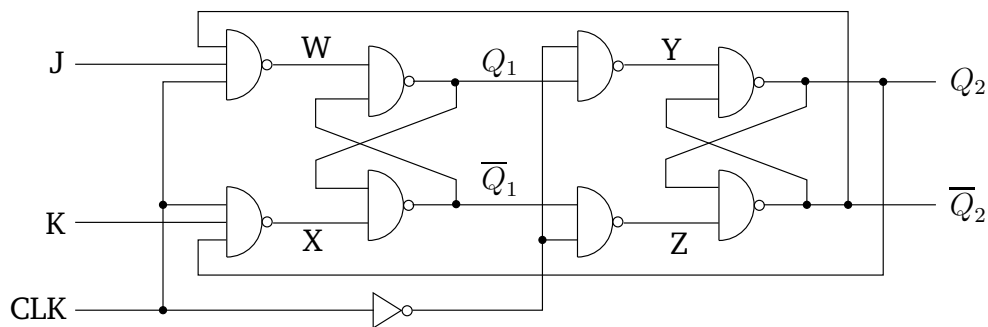


Figure 4.13: JK Master-Slave flip-flop

$CLK = 1$: $W = \overline{(JQ)}$ and $X = \overline{(KQ)}$, hence the primary inputs J and K determine the next state of the Master flip-flop. At the same time, $Y = Z = 1$ hence the slave flip-flop cannot change state when $CLK = 1$.

¹About the “political correctness” of the Master-Slave notation, see a news article at <http://www.cnn.com/2003/TECH/ptech/11/26/master.term.reut/>

$CLK = 0$: $W = X = 1$ hence the Master flip-flop cannot change states. At the same time, $Y = \bar{Q}_1$ and $Z = Q_1$ hence the outputs of the Master are transferred to the Slave flip-flop to determine the next output of the Master-Slave flip-flop.

The operation of the Master-Slave flip-flop can be summarized as shown in Figure 4.14.

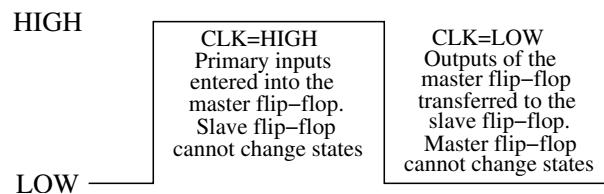


Figure 4.14: Response of a Master-Slave flip-flop to clock signal

The symbol for a JK Master-Slave flip-flop is shown on Figure 4.15.

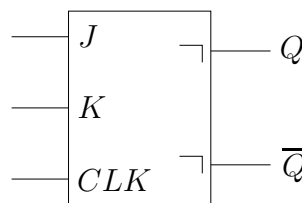


Figure 4.15: Symbol of a Master-Slave flip-flop

Exercise

Draw the circuit diagram for SET-RESET Master-Slave flip-flop and explain its operation.

4.3.4 Edge triggered flip-flops

Edge triggered flip-flops are those that only change state during the clock transitions (HIGH to LOW or LOW to HIGH). The flip-flops that change state during the HIGH to LOW transitions of the clock (negative going transitions) are known as negative edge triggered flip-flops, while those that change state at the LOW to HIGH transitions of the clock are known as positive edge triggered flip-flops. Note that there are no flip-flops that trigger on both the positive and the negative going transitions of the clock.

Figure 4.16 shows the logic symbol for a positive edge triggered JK flip-flop.

The corresponding truth-table is shown below:

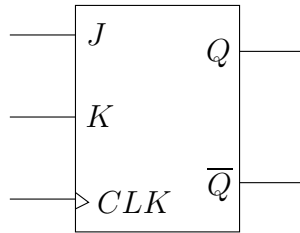


Figure 4.16: Symbol of a positive-edge-triggered JK flip-flop

J	K	CLK	Q_{n+1}
0	0	\uparrow	Q_n
0	1	\uparrow	0
1	0	\uparrow	1
1	1	\uparrow	\bar{Q}_n

The interpretation of the truth-table is as follows:

If the inputs of the flip-flop are $J = 0$, $K = 0$ and a Positive Going Transition (PGT) of the clock occurs, the flip-flop will remain in its present state (Q_n), while if the inputs are $J = 0$, $K = 1$, then a PGT of the clock will make the output to be LOW (0) e.t.c.

Figure 4.17 shows the logic symbol for a negative edge triggered JK flip-flop.

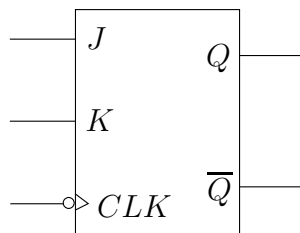


Figure 4.17: Symbol of a negative-edge-triggered JK flip-flop

The corresponding truth-table is shown below:

J	K	CLK	Q_{n+1}
0	0	\downarrow	Q_n
0	1	\downarrow	0
1	0	\downarrow	1
1	1	\downarrow	\bar{Q}_n

The interpretation of the truth-table is similar to that of the positive edge triggered flip-flop, the only difference is that changes in the state occur only on the negative going transitions of the clock.

Exercise

Figure 4.18 shows a clock waveform and inputs J and K applied to a positive-edge triggered JK flip-flop. Using the truth-table for a positive edge triggered JK flip-flop,

explain what happens at t_0 , t_1 , t_2 , t_3 , t_4 and t_5 to justify the output waveform shown for Q. (It is assumed that the initial of Q is HIGH)

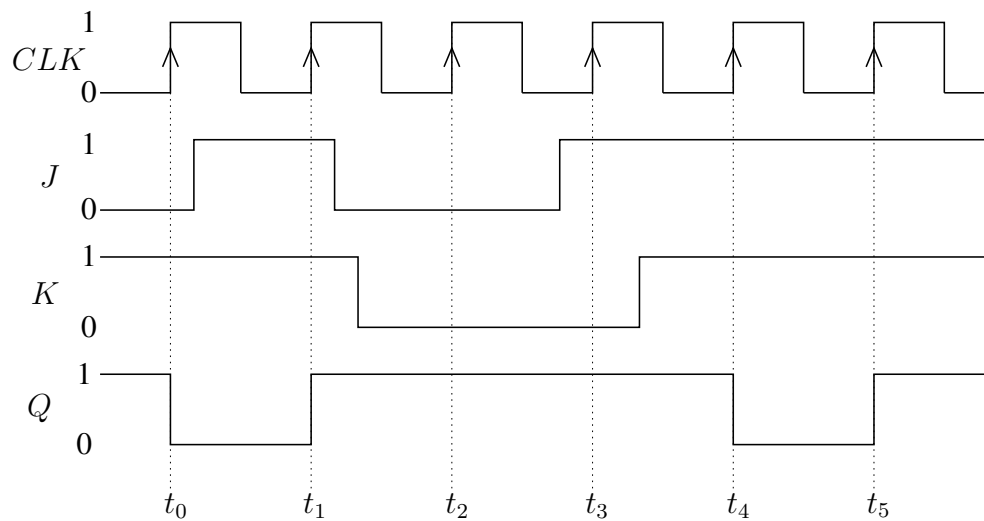


Figure 4.18: Exercise

4.3.5 Asynchronous inputs

The flip-flop inputs we have talked about so far (J,K, D, SET, RESET, T) are known as synchronous inputs. This is because the effect of these inputs to the flip-flop output is synchronized with the clock.

There is another type of inputs known as asynchronous inputs. Asynchronous inputs operate independently of the clock input. There are two asynchronous inputs in a flip-flop, the PRESET (sometimes referred to as SET) and CLEAR (sometimes referred to as RESET). The PRESET input is used to set the state of the flip-flop to 1 ($Q = 1$) regardless of the state of the synchronous inputs and the clock, while the CLEAR input is used to reset the flip-flop ($Q = 0$). These inputs are usually active-LOW, and a JK flip-flop having these two inputs is shown on Figure 4.19.

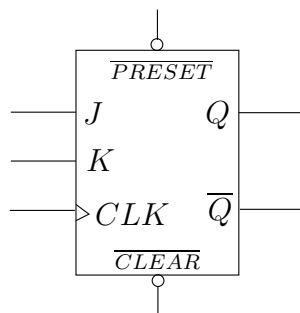


Figure 4.19: JK flip-flop with asynchronous inputs

The truth-table corresponding to this flip-flop is shown below:

\overline{PRESET}	\overline{CLEAR}	<i>Flip – flop response</i>
1	1	Responds to J, K and CLK (synchronous operation)
1	0	Flip-Flop RESET ($Q = 0$)
0	1	Flip-Flop SET ($Q = 1$)
0	0	Not used (= Setting and Resetting at the same time)

4.3.6 IC flip-flops

Examples of flip-flop ICs are the 7474 Dual D-type positive edge-triggered flip-flop IC, the 74107A Dual JK negative edge-triggered flip-flop IC, the 74109 Dual JK positive-edge triggered flip-flop IC, etc.

4.4 Flip-flop timing parameters

1. Set-up time t_S - Minimum time just before the triggering edge of the clock arrives during which the synchronous inputs must be held stable (5-50ns for TTL devices).
2. Hold-time t_H - Minimum time after the triggering edge of the clock during which the inputs must be held stable (about 10ns for TTL devices).
3. Propagation delays
 - t_{PLH} Time taken after the triggering edge of the clock for the output Q to change from LOW to HIGH.
 - t_{PHL} Time taken after the triggering edge of the clock for the output Q to change from HIGH to LOW.
4. Maximum clocking frequency f_{MAX} The highest frequency that may be applied to the clock input of a flip-flop and still have trigger reliably (TTL about 15MHz, CMOS about 5 MHz).
5. Clock pulse HIGH and LOW times - The manufacturer will specify the maximum duration that the clock must remain LOW before it goes HIGH ($t_w(L)$), and the minimum time that the clock must be kept HIGH before it goes LOW ($t_w(H)$).
6. Asynchronous active pulse width - Minimum time duration that the PRESET or the CLEAR input has to be kept in its active state in order to reliably set or clear a flip-flop.

7. Clock transition times - The time the clock pulse takes to change from LOW to HIGH (t_r), and from HIGH to LOW (t_f) should be short for reliable triggering. (for TTL devices, t_r and $t_f \leq 50\text{ns}$, for CMOS devices, t_r and $t_f \leq 200\text{ns}$)

4.5 Flip-flop excitation tables

SET-RESET flip-flop

Q_n	Q_{n+1}	SET	RESET
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

JK flip-flop

Q_n	Q_{n+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

D flip-flop

Q_n	Q_{n+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

T flip-flop

Q_n	Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

Exercise

Starting from the characteristic tables of the SET-RESET, JK, D and T flip-flops, derive the excitation tables above.

4.6 Derivation of one flip-flop function from another

The procedure:

1. Construct the state transition table for the *required* flip-flop function.
2. Determine the combination of inputs of the *flip-flop being used* that gives the same transition (this is done with the help of the flip-flop excitation tables in section 4.5 above).
3. Determine the relationship between the required inputs and the available flip-flop inputs.
4. Construct the circuit.

Example

Realize a JK flip-flop function using a SET-RESET (SR) flip-flop.

REQUIRED FUNCTION				FLIP-FLOP BEING USED	
J	K	Q_n	Q_{n+1}	S	R
0	0	0	0	0	X
0	0	1	1	X	0
0	1	0	0	0	X
0	1	1	0	0	1
1	0	0	1	1	0
1	0	1	1	X	0
1	1	0	1	1	0
1	1	1	0	0	1

The K-maps for S and R are shown on Figure 4.20.

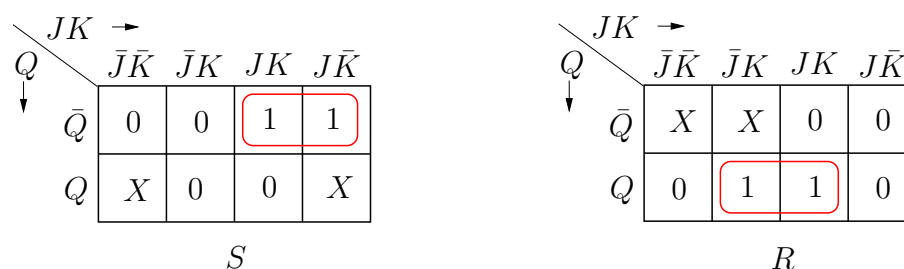


Figure 4.20: Example

From the K-maps, we can see that:

$$S = J\bar{Q} \quad \text{and} \quad R = KQ$$

The circuit is shown on Figure 4.21.

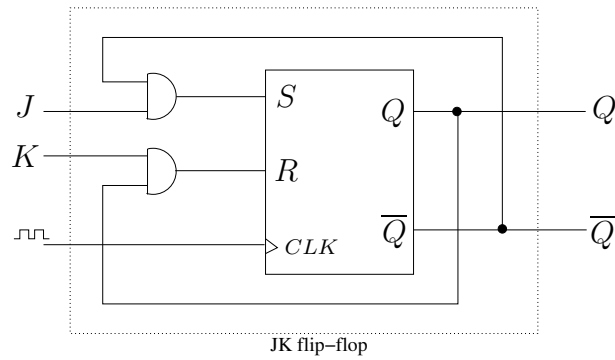


Figure 4.21: Realizing a JK flip-flop function from an SR flip-flop

Example

Realize a D flip-flop function using an SR flip-flop.

REQUIRED FUNCTION			FLIP-FLOP BEING USED	
D	Q_n	Q_{n+1}	S	R
0	0	0	0	X
0	1	0	0	1
1	0	1	1	0
1	1	1	X	0

The K-maps for S and R are shown on Figure 4.22.

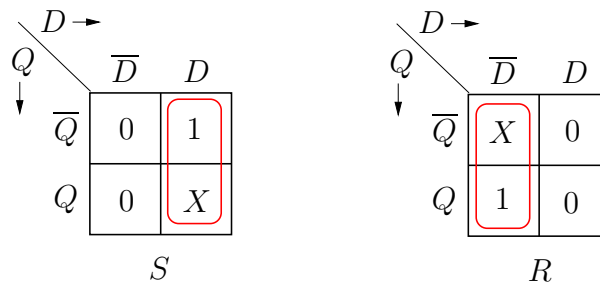


Figure 4.22: Example

From the K-maps, we can see that:

$$S = D \quad \text{and} \quad R = \bar{D}$$

The circuit is shown on Figure 4.23.

Example

Realize a JK flip-flop function using a D flip-flop.

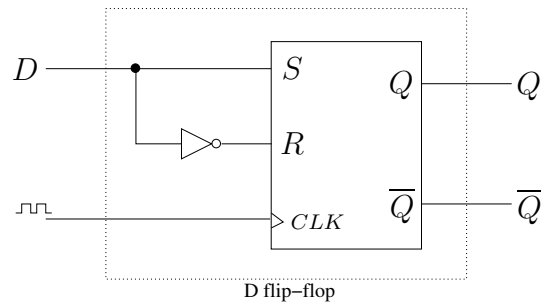


Figure 4.23: Realizing a D flip-flop function from an SR flip-flop

REQUIRED FUNCTION				FLIP-FLOP BEING USED
J	K	Q_n	Q_{n+1}	D
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	0	0

The K-map for D is shown on Figure 4.24.

		$JK \rightarrow$			
		$\bar{J}\bar{K}$	$\bar{J}K$	JK	$J\bar{K}$
Q	\bar{Q}	0	0	1	1
	Q	1	0	0	1

S

Figure 4.24: Example

From the K-map, we can see that:

$$D = J\bar{Q} + \bar{K}Q$$

The circuit is shown on Figure 4.25.

Exercise

From the circuit of Figure 4.26, complete the truth-table below and hence design a JK flip-flop based circuit which has the same truth-table, but in which the asynchronous inputs (\overline{PRESET} , \overline{CLEAR}) are not used.

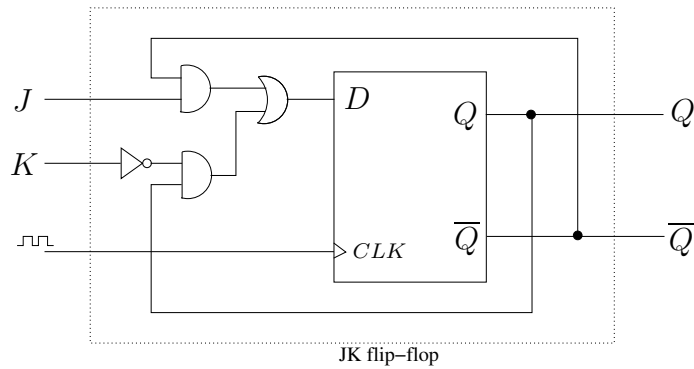


Figure 4.25: Realizing a JK flip-flop function from an D flip-flop

X	Q_n	Q_{n+1}
0	0	
0	1	
1	0	
1	1	

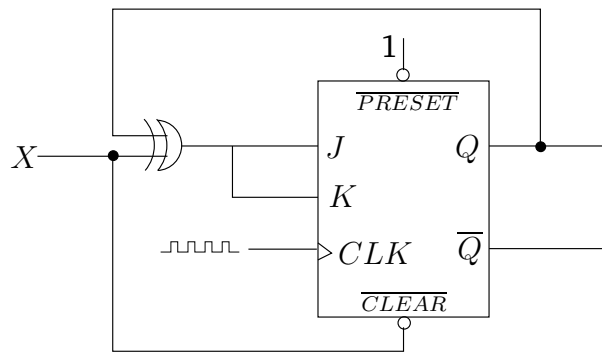


Figure 4.26: Exercise

4.7 Counters

A counter is a circuit made up of a series of flip-flops connected in a suitable manner to record sequences of pulses presented to it in digital form. Counters are used in digital systems to:

1. Count events - convert a given number of event cycles into a prescribed binary code.
2. To time events - The duration of an event can be determined from the count and the clock frequency.
3. To generate special code sequences - In this application, the counter is used as

the heart of a logic sequencer. Such a circuit generates the next-state information as well as the control information.

4. Frequency division

4.7.1 Classification of counters

1. Mode of operation

- Single mode counter - Counts without external control inputs.
- Multi-mode counter - Counters with external control inputs that can be used to alter the counting sequence, the beginning of a sequence or the end of a sequence.

2. Modulus of the counter - The maximum number that a counter can sequence through is called the modulus of the counter, e.g. if the counter goes through the sequence 000, 001, 010, 011, 100, it goes through five states hence the modulus of the counter is five. A modulus N counter is also referred to as a MOD N counter or a DIVIDE by N counter hence the above counter can be referred to as a MOD 5 counter or a DIVIDE by 5 counter.

Generally speaking, a counter that can sequence through N states is known as a MODULO N counter or a DIVIDE by N counter. If N is a power of 2 (2, 4, 8, 16 e.t.c.), the counter is called a binary counter. If N is a power of 10 (10, 100, 1000 e.t.c.), the counter is known as a decimal or a decade counter.

3. Clocking method - There are two types of clocking for counters:

- i) Asynchronous (ripple) - Type of counter where each flip-flop serves as the clock input signal for the next flip-flop in the chain. Examples are shown on Figure 4.27.

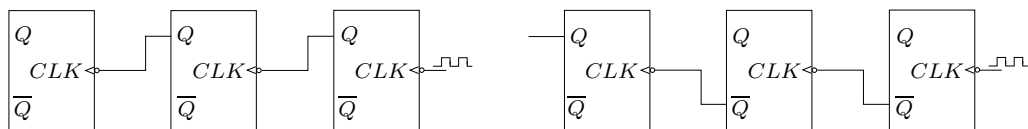


Figure 4.27: Asynchronous (ripple) counters

- ii) Synchronous - Counter in which all the flip-flops are clocked simultaneously. This is illustrated on Figure 4.28.

4. Code sequence generated - A counter may be a straight forward up or down counter (a counter generating codes $00 \rightarrow 01 \rightarrow 10 \rightarrow 11$ is an up counter, while one generating codes $11 \rightarrow 10 \rightarrow 01 \rightarrow 00$ is a down counter). Other counters

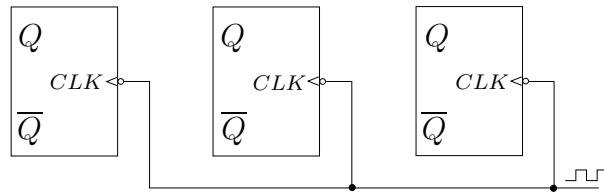


Figure 4.28: Synchronous counter

include Gray Code counters and the shift register counters (ring counter and Johnson counters).

Note that counters can be designed using a combination of flip-flops and gates, and counters are also available in I.C. form. In the next sections, we shall look at counters designed using a combination of flip-flops and gates.

4.7.2 Asynchronous Counters

An asynchronous counter comprises a chain of flip-flops in which one flip-flop is under the command of an external clock input. The other flip-flops in the chain are indirectly controlled by the external clock. The flip-flops are connected such that each flip-flop generates the clock input to the next flip-flop in the chain. An asynchronous counter is also referred to as a ripple counter. This is because the effect of the external clock ripples through the chain of flip-flops.

Example

Consider the circuit shown on Figure 4.29.

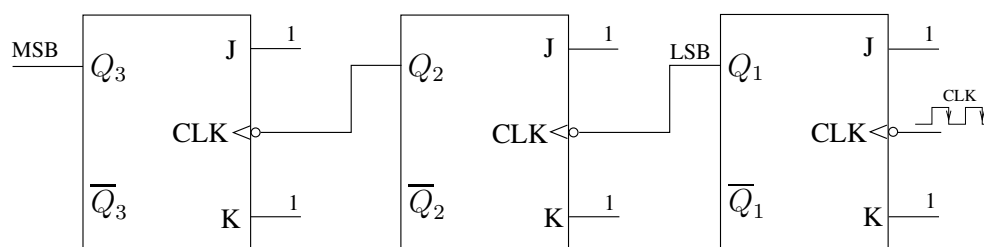


Figure 4.29: A ripple counter

Assuming that we start with $Q_3 = Q_2 = Q_1 = 0$, we can draw the waveforms shown on Figure 4.30.

The truth-table corresponding to this waveforms is shown below:

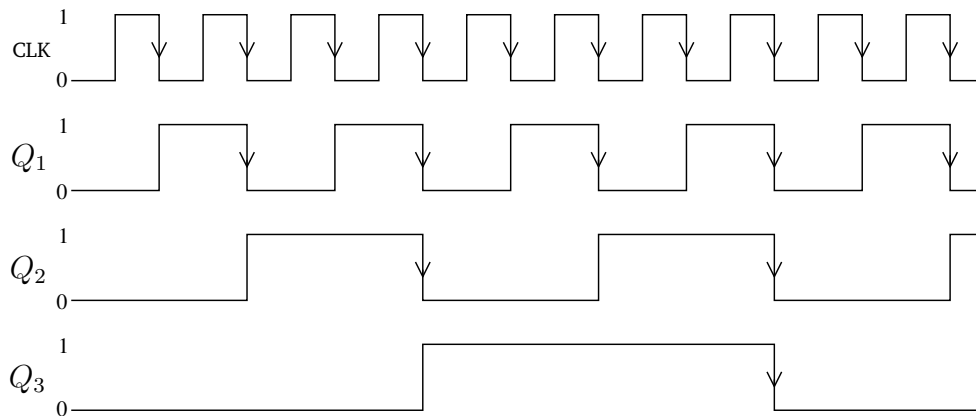


Figure 4.30: Waveforms for the counter in Figure 4.29

Q_3	Q_2	Q_1	clock pulses
0	0	0	before applying clock pulses
0	0	1	after clock pulse #1
0	1	0	after clock pulse #2
0	1	1	after clock pulse #3
1	0	0	after clock pulse #4
1	0	1	after clock pulse #5
1	1	0	after clock pulse #6
1	1	1	after clock pulse #7
0	0	0	recycles to 000 after clock pulse #8
0	0	1	after clock pulse #9

From the table above, we can see that the counter has eight distinct states and it is an up counter. The counter is therefore a modulo-8 up asynchronous (ripple) counter. Note that the frequency of the waveform for Q_3 is $\frac{1}{8}$ of the clock frequency hence this can also be referred to as a divide by 8 counter.

Generally, for N negative-edge triggered JK flip-flops connected in the form shown on Figure 4.29, the counter formed is a Modulo- 2^N ripple up counter also known as a Divide by 2^N ripple up counter. For any ripple counter, the output from the last flip-flop in the chain divides the input clock frequency by the MOD number of the counter.

Example

Consider the circuit shown on Figure 4.31.

Assuming that we start with $Q_3 = Q_2 = Q_1 = 0$, we can draw the waveforms shown on Figure 4.32.

We then get the truth-table for Q_1 , Q_2 and Q_3 , which is shown below:

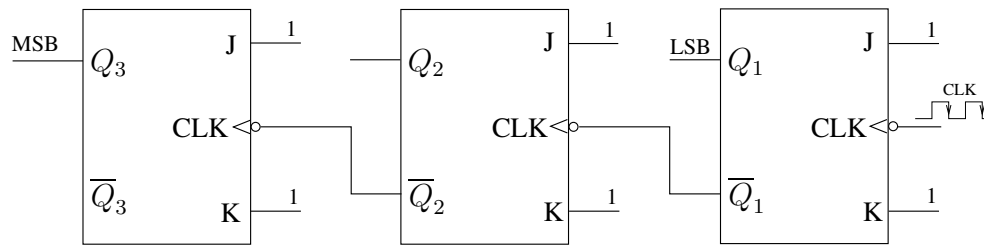


Figure 4.31:

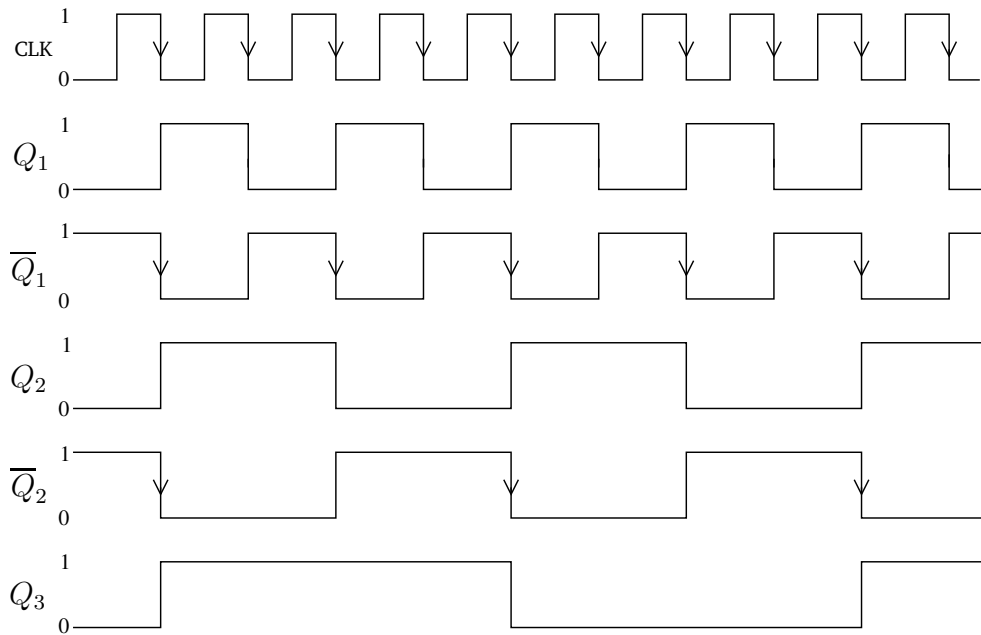


Figure 4.32: Waveforms for the counter in Figure 4.31

Q_3	Q_2	Q_1	clock pulses
0	0	0	before applying clock pulses
1	1	1	after clock pulse #1
1	1	0	after clock pulse #2
1	0	1	after clock pulse #3
1	0	0	after clock pulse #4
0	1	1	after clock pulse #5
0	1	0	after clock pulse #6
0	0	1	after clock pulse #7
0	0	0	recycles to 000 after clock pulse #8
1	1	1	after clock pulse #9

From the table above, we can see that the counter has eight distinct states and it is a down counter. The counter is therefore a modulo-8 down asynchronous (ripple) counter or a divide by 8 down asynchronous counter.

Note: Negative-edge triggered JK flip-flops connected as shown on Figure 4.29 form

up-counters, while those connected as in Figure 4.31 form down counters. *If positive-edge triggered JK flip-flops were used, the situation would reverse, such that a connection similar to that of Figure 4.29 would yield a down counter; while a connection such as the one of Figure 4.31 would yield an up-counter.* Draw the waveforms for positive-edge triggered JK flip-flops and verify this.

4.7.3 Asynchronous Counters with MOD numbers $< 2^N$

The general design procedure is as follows:

1. For a modulo- k counter, find the smallest number of flip-flops N such that $2^{N-1} < k < 2^N$, and connect them as a ripple counter.
2. Connect a NAND gate output to the asynchronous CLEAR inputs of all the flip-flops.
3. Determine which flip-flops will be in the HIGH state at a count k and then connect the normal outputs of these flip-flops to the NAND gate inputs.

Example

Design a JK flip-flop-based modulo-6 asynchronous up counter.

Solution

The MOD number $k = 6$, and this is not a power of 2. Using the equation above:

$$2^{N-1} < 6 < 2^N$$

we find that the value of N which satisfies this is $N = 3$, hence we use three flip-flops which we connect as a ripple counter. $k = 6 = 110_2$, hence the two most-significant digits of the counter are the ones that should be connected to the input of the NAND gate, to yield the circuit shown on Figure 4.33.

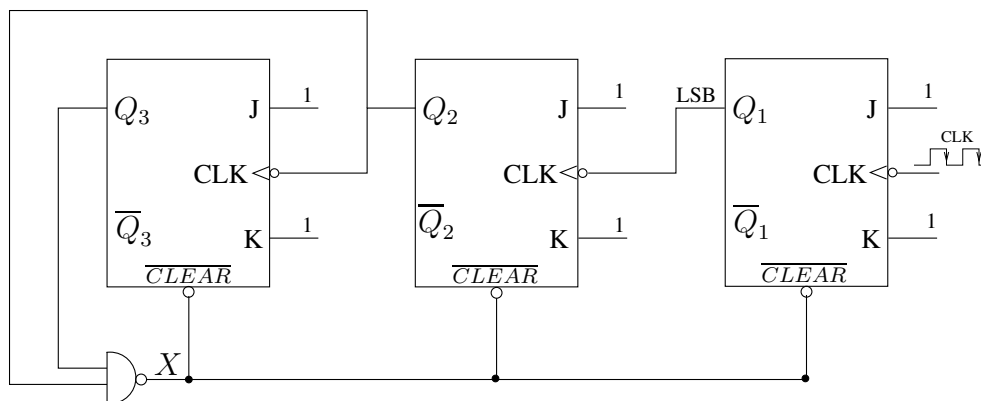


Figure 4.33: A modulo 6 up ripple counter

Assuming that we start with $Q_3 = Q_2 = Q_1 = 0$, we can draw the waveforms shown on Figure 4.34.

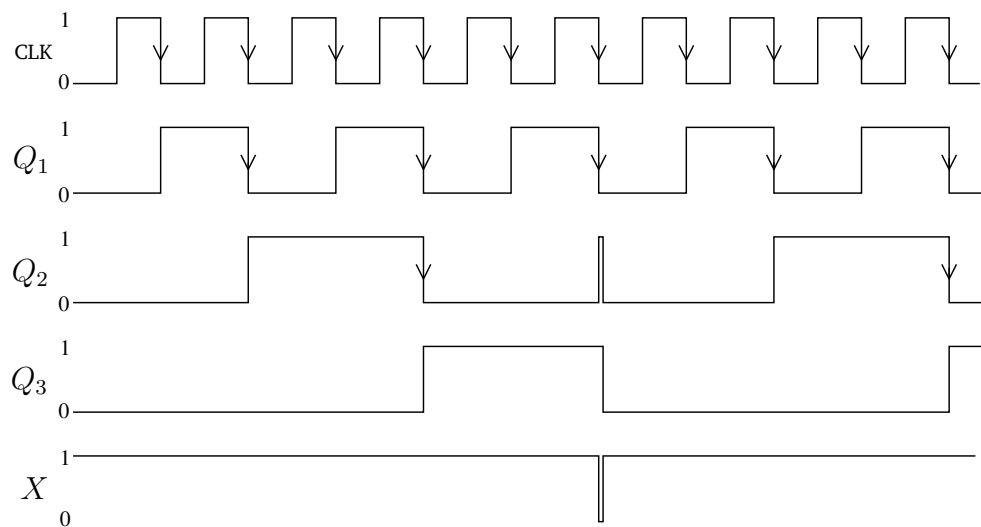


Figure 4.34: Waveforms for the counter in Figure 4.33

From Figure 4.34, we can see that when Q_3 and Q_2 go HIGH, the output of the NAND gate (X) goes LOW, and this clears all the flip-flops (Q_3 , Q_2 and Q_1 are taken to the LOW state). When Q_3 and Q_2 go LOW, the NAND gate output goes HIGH and the counter resumes normal operation. The duration of the resetting pulse is very short, typically $< 100\text{ns}$, so the counter state 110 is a temporary state and is not considered as a valid state of the counter. Assuming that we start with $Q_3 = Q_2 = Q_1 = 0$, we can draw the truth-table for this counter as shown below:

Q_3	Q_2	Q_1	clock pulses
0	0	0	before applying clock pulses
0	0	1	after clock pulse #1
0	1	0	after clock pulse #2
0	1	1	after clock pulse #3
1	0	0	after clock pulse #4
1	0	1	after clock pulse #5
1	1	0	temporary state, cannot be observed
0	0	0	recycles to 000 after clock pulse #6
0	0	1	after clock pulse #7
0	1	0	after clock pulse #8

The counter has 6 stable states (000, 001, 010, 011, 100 and 101) hence it is a modulo-6 up asynchronous counter.

NOTE: Cascading a MOD k_1 ripple counter with a MOD k_2 ripple counter results in a MOD $k_1 k_2$ counter.

Exercise

- i) Design a JK flip-flop-based MOD 3 ripple up counter.
- ii) Cascade two of these MOD 3 counters and show that they form a MOD 9 counter which goes through the sequence $0000 \rightarrow 0001 \rightarrow 0010 \rightarrow 0100 \rightarrow 0101 \rightarrow 0110 \rightarrow 1000 \rightarrow 1001 \rightarrow 1010 \rightarrow 0000$
- iii) Describe how propagation delays affect the performance of ripple counters.

4.7.4 Synchronous Counters

These are counters in which all the flip-flops are clocked simultaneously, and as a result, all flip-flop output changes occur simultaneously. Propagation delays do not add together hence synchronous counters can work at much higher frequencies than ripple counters.

The design procedure is as follows:

- i) Write the state-transition-table for the count.
- ii) Derive a minimal logic expression for each flip-flop input.
- iii) Implement the logic circuit using flip-flops and a suitable set of logic gates.

Example

Design a JK flip-flop based circuit whose state diagram is shown on Figure 4.35.

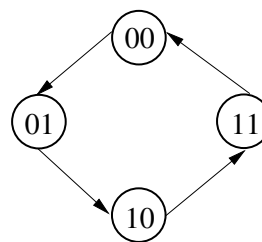


Figure 4.35:

Solution

Since we are to use JK flip-flops to implement this function, we shall use the JK flip-flop excitation table given in section 4.5. We then construct the table shown below:

PRESENT STATE		NEXT STATE		FLIP-FLOP INPUTS			
Q_A	Q_B	Q_{A+1}	Q_{B+1}	J_A	K_A	J_B	K_B
0	0	0	1	0	X	1	X
0	1	1	0	1	X	X	1
1	0	1	1	X	0	1	X
1	1	0	0	X	1	X	1

The K-maps corresponding to the four flip-flop inputs are shown on Figure 4.36.

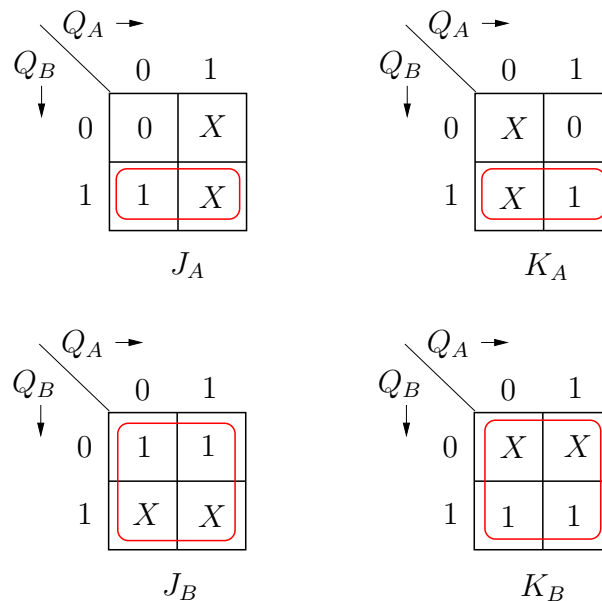


Figure 4.36: Example

From the figure, we can see that:

$$J_A = K_A = Q_B \quad \text{and} \quad J_B = K_B = 1$$

The circuit is shown on Figure 4.37.

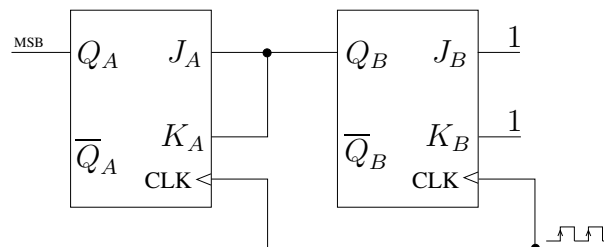


Figure 4.37: Circuit for a MOD 4 synchronous counter

Note that the flip-flops may be either positive-edge-triggered or negative-edge-triggered. For observation of the states, Q_A and Q_B may be connected to LEDs (Q_A is the MSB while Q_B is the LSB).

Example

Design a JK flip-flop based circuit whose state diagram is shown on Figure 4.38.

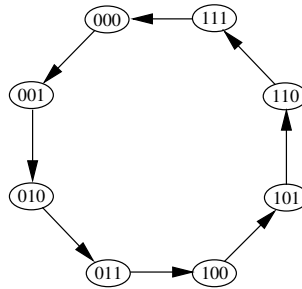


Figure 4.38:

Using the JK flip-flop excitation table given in section 4.5, we construct the table shown below:

PRESENT STATE			NEXT STATE			FLIP-FLOP INPUTS					
Q_A	Q_B	Q_C	Q_{A+1}	Q_{B+1}	Q_{C+1}	J_A	K_A	J_B	K_B	J_C	K_C
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	0	1	0	0	X	1	X	X	1
0	1	0	0	1	1	0	X	X	0	1	X
0	1	1	1	0	0	1	X	X	1	X	1
1	0	0	1	0	1	X	0	0	X	1	X
1	0	1	1	1	0	X	0	1	X	X	1
1	1	0	1	1	1	X	0	X	0	1	X
1	1	1	0	0	0	X	1	X	1	X	1

The K-maps corresponding to the four flip-flop inputs are shown on Figure 4.39.

From Figure 4.39, we can see that:

$$J_A = K_A = Q_B Q_C, \quad J_B = K_B = Q_C \quad \text{and} \quad J_C = K_C = 1$$

The circuit is shown on Figure 4.40.

In this case, Q_A is the MSB while Q_C is the LSB.

Example:

In some cases, the counter may not go through all the possible states. Such a case is illustrated on Figure 4.41.

The counter has four states, 000, 010, 011 and 101. Since each state is represented by three bits, it means that the counter is made using three flip-flops. Three flip-flops can be used to construct a counter with a maximum of eight states, 000 through to 111. It therefore means that the counter above skips states 001, 100, 110 and 111. The design procedure for such a counter is similar to the cases above, only that the skipped states are treated as don't-care variables. The table corresponding to this counter is shown below:

$Q_A Q_B \rightarrow$					
$Q_C \downarrow$		00	01	11	10
	0	0	0	X	X
	1	0	1	X	X
J_A					
$Q_A Q_B \rightarrow$					
$Q_C \downarrow$		00	01	11	10
	0	X	X	0	0
	1	X	X	1	0
K_A					
$Q_A Q_B \rightarrow$					
$Q_C \downarrow$		00	01	11	10
	0	0	X	X	0
	1	1	X	X	1
J_B					
$Q_A Q_B \rightarrow$					
$Q_C \downarrow$		00	01	11	10
	0	X	0	0	X
	1	X	1	1	X
K_B					
$Q_A Q_B \rightarrow$					
$Q_C \downarrow$		00	01	11	10
	0	1	1	1	1
	1	X	X	X	X
J_C					
$Q_A Q_B \rightarrow$					
$Q_C \downarrow$		00	01	11	10
	0	X	X	X	X
	1	1	1	1	1
K_C					

Figure 4.39: Example

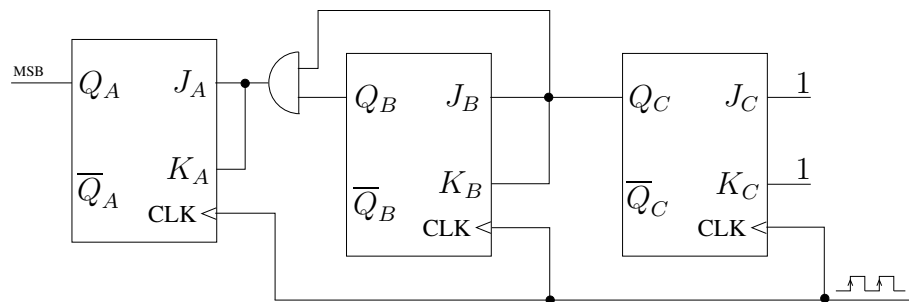


Figure 4.40: Circuit for a MOD 8 synchronous counter

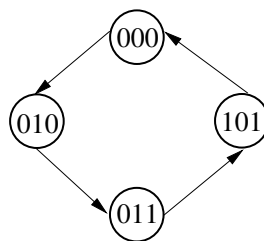


Figure 4.41:

PRESENT STATE			NEXT STATE			FLIP-FLOP INPUTS					
Q_A	Q_B	Q_C	Q_{A+1}	Q_{B+1}	Q_{C+1}	J_A	K_A	J_B	K_B	J_C	K_C
0	0	0	0	1	0	0	X	1	X	0	X
0	0	1	-	-	-	X	X	X	X	X	X
0	1	0	0	1	1	0	X	X	0	1	X
0	1	1	1	0	1	1	X	X	1	X	0
1	0	0	-	-	-	X	X	X	X	X	X
1	0	1	0	0	0	X	1	0	X	X	1
1	1	0	-	-	-	X	X	X	X	X	X
1	1	1	-	-	-	X	X	X	X	X	X

The K-maps corresponding to the four flip-flop inputs are shown on Figure 4.42.

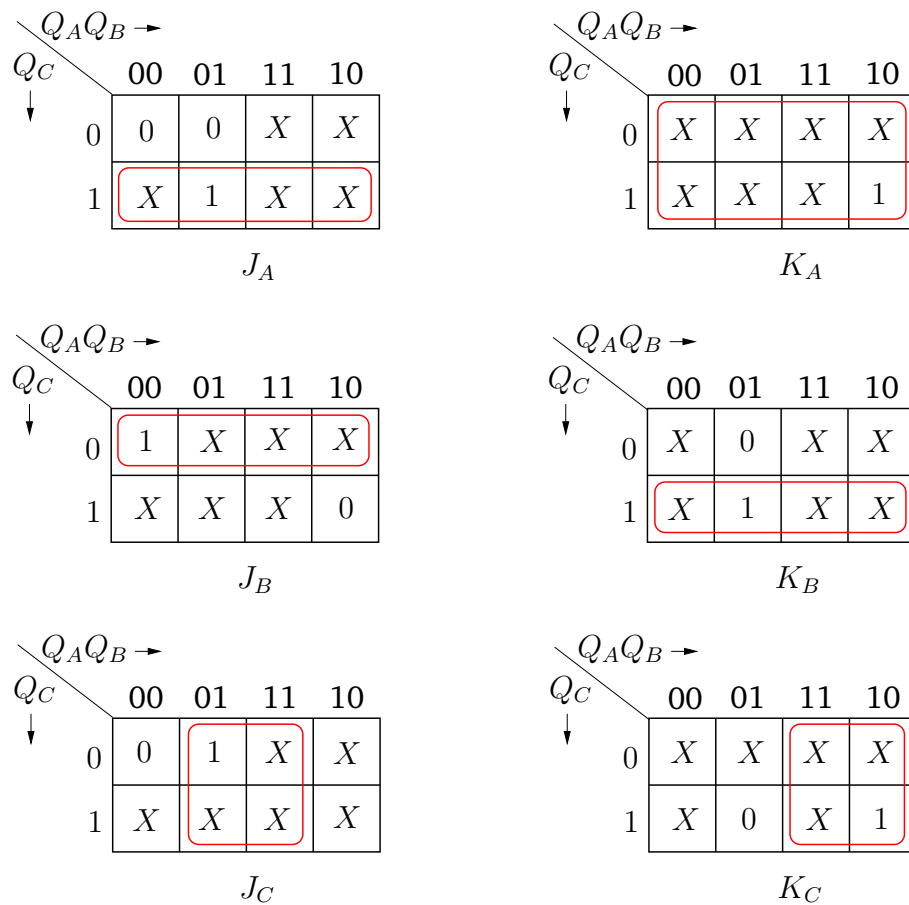


Figure 4.42: Example

From Figure 4.42, we can see that:

$$J_A = Q_C, K_A = 1, J_B = \bar{Q}_C, K_B = Q_C, J_C = Q_B, K_C = Q_A$$

Note that K_A and J_B may be looped in more than one way – we can loop such that $K_A = Q_C$ or $K_A = Q_A$, and $J_B = \bar{Q}_A$, which are also acceptable.

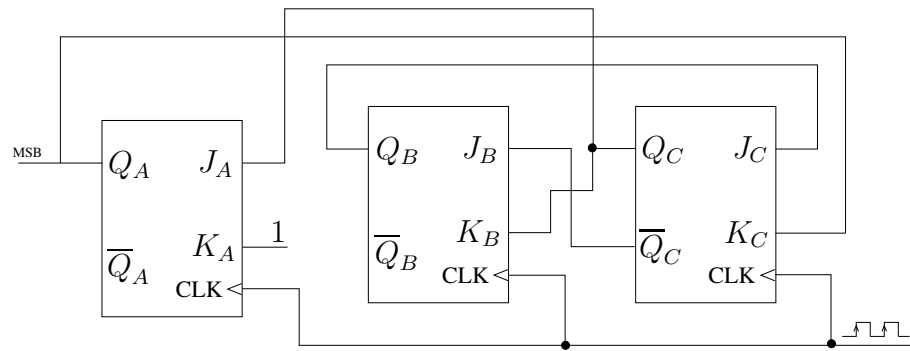


Figure 4.43:

The circuit is shown on Figure 4.43.

In this case, Q_A is the MSB while Q_C is the LSB.

4.7.5 Shift Register Counters

All shift register counters use feedback whereby the output of the last flip-flop in the shift register is in some way connected to the first flip-flop.

Ring counter

This is a circulating shift register connected so that the last flip-flop shifts its value into the first flip-flop. *It operates by recirculating either a single '1' or a '0'.* An example of a 4-bit ring counter is shown on Figure 4.44.

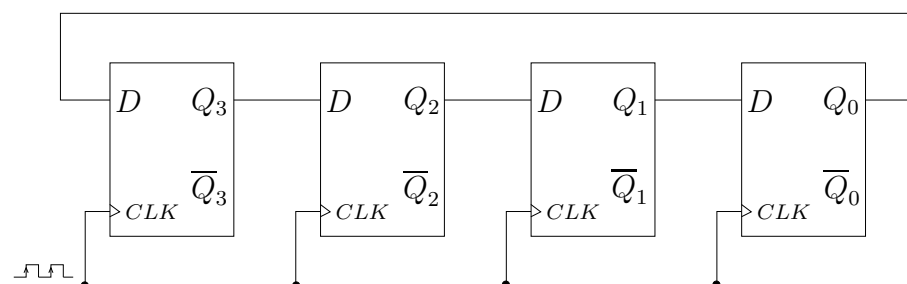


Figure 4.44: A 4-bit ring counter

Assuming that the initial state of the circuit is $Q_3Q_2Q_1Q_0 = 1000$, the timing waveforms for the circuit are shown on Figure 4.45.

The waveforms can be summarized by the table below:

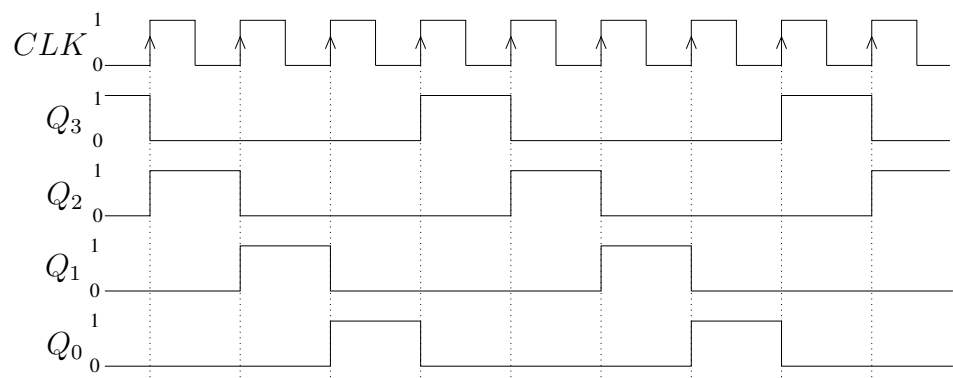


Figure 4.45: Timing waveforms for the 4-bit ring counter

Q_3	Q_2	Q_1	Q_0	clock pulses
1	0	0	0	before applying clock pulses
0	1	0	0	after clock pulse #1
0	0	1	0	after clock pulse #2
0	0	0	1	after clock pulse #3
1	0	0	0	after clock pulse #4 - recycles

Note that although this does not progress through the normal binary counting sequence, it is still a counter because each count corresponds to a particular state of the counter. The circuit shown on Figure 4.44 is therefore a MOD 4 ring counter which recirculates a ‘1’. In general, a MOD N ring counter requires N flip-flops. This type of a counter is mostly used in sequencing operations.

Johnson Counter

This type of counter is also known as the twisted-ring counter. It is constructed exactly like a normal ring counter, except that the *inverted* output of the last flip-flop is connected to the first flip-flop. Figure 4.46 shows a 3-bit Johnson counter.

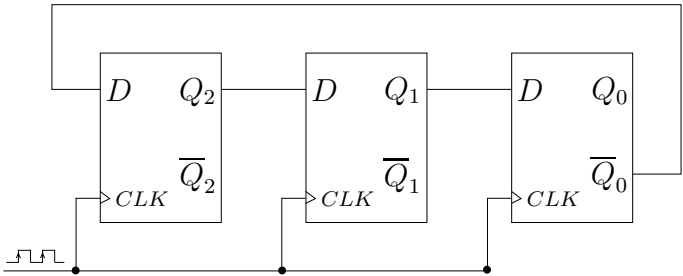


Figure 4.46: A 3-bit Johnson (twisted ring) counter

Assuming the initial state is $Q_2Q_1Q_0 = 000$, the timing waveforms are shown on Figure 4.47.

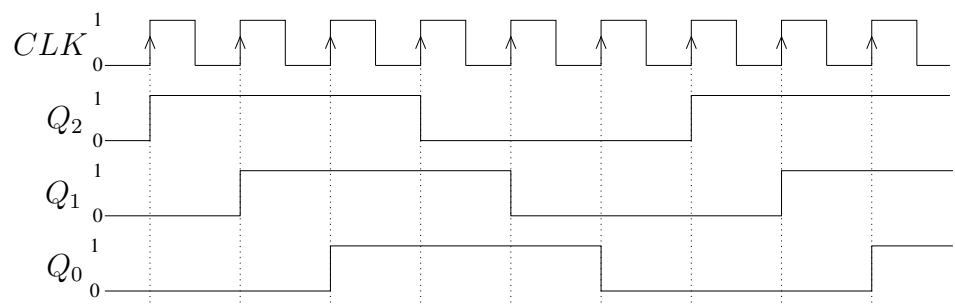


Figure 4.47: Timing waveforms for the 3-bit Johnson counter

The table below summarizes the timing waveforms:

Q_2	Q_1	Q_0	clock pulses
0	0	0	before applying clock pulses
1	0	0	after clock pulse #1
1	1	0	after clock pulse #2
1	1	1	after clock pulse #3
0	1	1	after clock pulse #4
0	0	1	after clock pulse #5
0	0	0	after clock pulse #6 - recycles
1	0	0	after clock pulse #7

Figure 4.46 therefore shows a MOD 6 Johnson (or twisted ring) counter. In general, N flip-flops connected as a Johnson counter will have a MOD number $2N$.

4.7.6 I.C. Counters

Although counters can be constructed using flip-flops, they are also available as ICs. Examples of IC counters include the asynchronous 4-bit ripple counter 74293, synchronous 4-bit 74193 and the decade counter 7490. Other IC counters include the 74176, 74196, 74177, 74197 etc. More information on the operation of these counter ICs can be obtained from the data books and the references listed at the end of this handout.

4.8 Registers

A flip-flop is used to store 1 bit of information. When flip-flops are organized to store many bits of information, they are called registers. We can therefore say that a flip-flop is a single bit register. Registers can be implemented using discrete components (flip-flops and gates), and are also available in IC form. Data can be entered into a register in parallel form (all bits simultaneously), or in

serial form (one bit at a time). If data is entered to all flip-flops in a register at the same time, the register is referred to as a *parallel register*. If data is entered and removed one bit at a time, the register is known as a *serial register* or a *shift register*. Figure 4.48 shows a 3-bit parallel register while Figure 4.49 shows a 3-bit shift (serial) register.

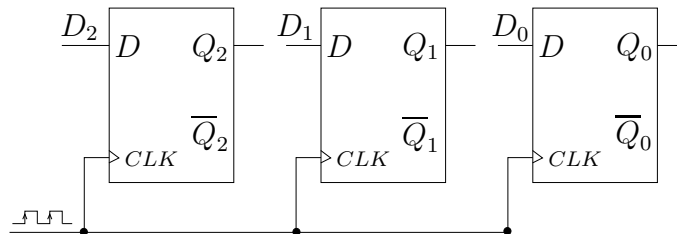


Figure 4.48: A 3-bit parallel register

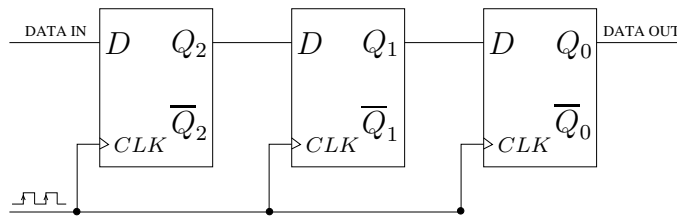


Figure 4.49: A 3-bit serial register

The operation most often performed on data that is stored in a flip-flop is the transfer operation. This involves the transfer of data from one register to another. A transfer operation is said to be a *synchronous transfer* when the clock inputs are used to perform the transfer, and *asynchronous transfer* when the asynchronous inputs (\overline{PRESET} , \overline{CLEAR}) are used. Figure 4.50 shows a synchronous parallel data transfer while Figure 4.51 shows a synchronous serial data transfer.

Registers available in IC form can be classified in the following categories:

- i) Serial In Serial Out (SISO) - For this kind of registers, data can only be entered in serial form, and removed in serial form e.g. 7491 8-bit register.
- ii) Serial In Parallel Out (SIPO) - Data entered in serial form, read in parallel form e.g. 74164 8-bit register.
- iii) Parallel In Serial Out (PISO) e.g. 7494.
- iv) Parallel In Parallel Out (PIPO).

NOTE

- Some shift registers allow a combination of the operation modes above e.g. the 74165 can be operated in SISO and PISO modes.

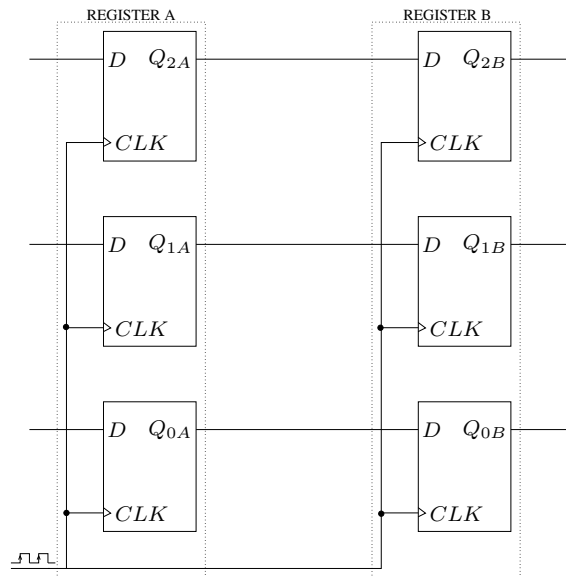


Figure 4.50: Synchronous parallel data transfer

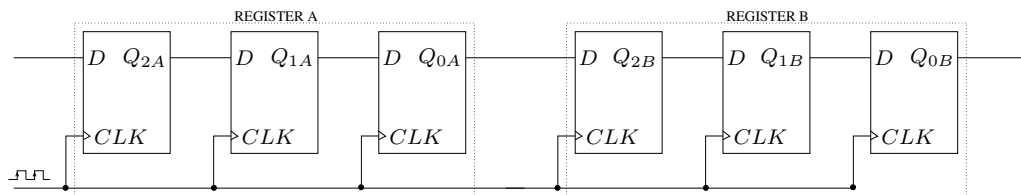


Figure 4.51: Synchronous serial data transfer

- A shift register where data can only be shifted to the right is known as a *right-shift register*, while the ones where data can only be shifted to the left are known as *left-shift registers*. The shift registers where data can be shifted both left and right are known as *bidirectional shift registers*.
- If a register can be operated in all the four modes above (SISO, SIPO, PISO, PIPO), and has a bidirectional shift, it is referred to as a *universal register* e.g. 74194.

Chapter 5

LOGIC FAMILIES AND THEIR CHARACTERISTICS

5.1 Introduction

A logic family comprises a set of devices manufactured using a particular process. Each member of a particular logic family has the same basic characteristics that are unique to that logic family. The logic families include:

1. Direct Coupled Transistor Logic (DCTL) - the oldest
2. Resistor Transistor Logic (RTL)
3. Diode Transistor Logic (DTL)
4. Transistor Transistor Logic (TTL)
5. Emitter Coupled Logic (ECL)
6. Integrated Injection Logic (IIL or I^2L)
7. P or N channel Metal Oxide Semiconductor Logic (PMOS or NMOS)
8. Complementary Metal Oxide Semiconductor Logic (CMOS).

All these logic families have different characteristics. The first six in the list use bipolar transistors while the other two use field effect transistors. The first three logic families are obsolete though you may come across DTL devices in some old electronic equipment. In this course, we are going to look at the most common logic families, the TTL and CMOS.

5.2 Characteristics

- i) Propagation Delay (speed). The speed of a logic family is measured by the propagation delay time of the basic inverter or the NAND gate. This is illustrated on Figure 5.1.

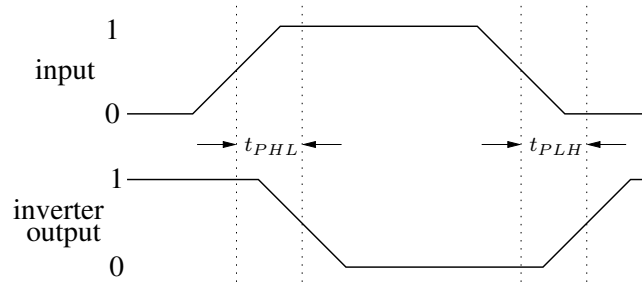


Figure 5.1: Propagation delay

- t_{PHL} is the propagation delay time for the HIGH to LOW transition.
- t_{PLH} is the propagation delay time for the LOW to HIGH transition.

The propagation delay time for a logic family is given as:

$$t_p = \frac{t_{PHL} + t_{PLH}}{2}$$

- ii) Power requirements - every gate requires a certain amount of power to operate, and this differs between the logic families. Generally, a high power consumption means higher operation costs. Battery operated appliances should be designed using logic families with low power consumption.
- iii) Fan-out (loading factor) - This is the maximum number of standard logic *inputs* that a logic output can reliably drive e.g. if the fan-out of a logic device is specified to be 10, then it can drive a maximum of 10 standard logic inputs. If this number is exceeded, the output logic-level voltages cannot be guaranteed. Buffers can be used to increase the fan-out of a gate. (The number of input terminals to a gate is referred to as the fan-in)
- iv) Noise immunity - Stray electrical and magnetic fields induce voltages on the connecting wires between logic circuits. These unwanted signals can sometimes cause the input voltages to change, and this could produce unpredictable outputs. Noise immunity of a logic family refers to the circuits' ability to tolerate noise voltages on its inputs. A quantitative measure of the noise immunity is called noise margin.
- v) Speed-Power product

$$= \text{Propagation delay} \times \text{Power Consumption}$$

5.3 Definitions

- V_{CC} - Supply voltage
- V_{IH} - Input HIGH voltage - range of input voltages that represent logic 1 in the system.
- $V_{IH}(\min)$ - Minimum input HIGH voltage - this is the minimum allowed input HIGH in a logic system.
- V_{IL} - Input LOW voltage - range of input voltages that represent a logic 0 in a system.
- $V_{IH}(\max)$ - Maximum input LOW voltage - The maximum allowed input LOW in a system.
- $V_{OH}(\min)$ - minimum output HIGH voltage - the minimum HIGH voltage at an output terminal in the HIGH state.
- $V_{OL}(\max)$ - maximum voltage at an output in its LOW state.

All logic families are designed such that:

$$V_{OH}(\min) > V_{IH}(\min)$$

$$V_{OL}(\max) < V_{IL}(\max)$$

This gives us the noise margin, which is worked out as follows:

$$\text{Noise margin (HIGH state)} = V_{NH} = V_{OH}(\min) - V_{IH}(\min)$$

$$\text{Noise margin (LOW state)} = V_{NL} = V_{IL}(\max) - V_{OL}(\max)$$

The smaller of the values V_{NH} and V_{NL} is taken to be the noise margin for a logic family. The voltages defined above are illustrated on Figure 5.2, using a NAND gate and an inverter.

5.4 Current Sourcing and Current Sinking

Consider the circuit shown on Figure 5.3, where the output of the driving gate is HIGH.

The current flows in the direction shown by the arrow, so in this case, the output of the driving gate is acting as a current source for the load gate. This is known as *current sourcing*.

Now consider the circuit shown on Figure 5.4, where the output of the driving gate is LOW.

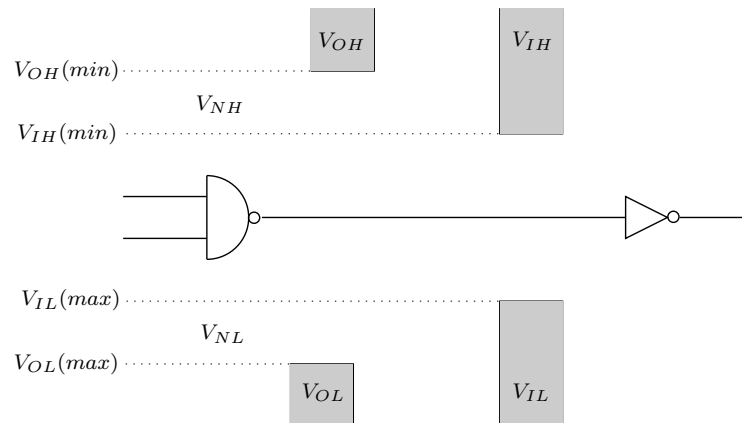


Figure 5.2:

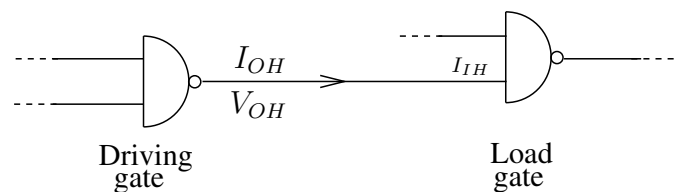


Figure 5.3: Current sourcing

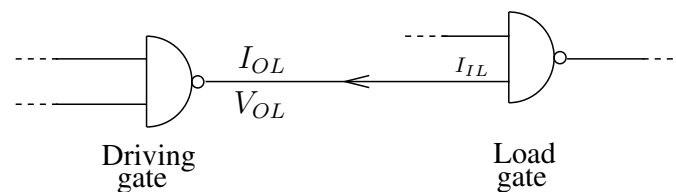


Figure 5.4: Current sinking

The current flows in the direction shown by the arrow, hence the output of the driving gate is acting as a current sink for the load gate. This is known as *current sinking*.

- I_{CC} Supply Current.
- I_{IH} Input HIGH current - Current flowing into an input when a HIGH level voltage is applied.
- I_{IL} Input LOW current - Current flowing out of an input when a LOW voltage is applied.
- I_{OH} Output HIGH current - Current flowing out of an output in the HIGH state under specified load conditions.
- I_{OL} Output LOW current - Current flowing into an output which is in the LOW state under specified load conditions.

5.5 DTL gates

NOTE:

- ◇ The basic circuit in each IC digital logic family is either a NAND gate or a NOR gate. This basic circuit is the primary building block from which all other more complex digital components are obtained.
- ◇ In all the logic gates, transistors are operated only in two modes: saturation and cut-off. That way, the transistor operates as a switch: open (cut-off), and closed (saturation).

Figure 5.5 shows a Diode-Transistor Logic (DTL) NAND gate. Note that although this type of NAND gate is no longer manufactured, the understanding of how it works will assist in understanding the operation of TTL gates that we shall come across later.

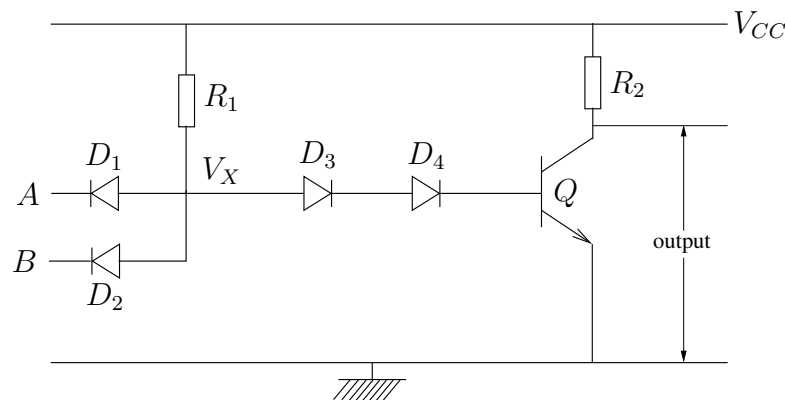


Figure 5.5: DTL NAND gate

When A or B or both are both LOW, D_1 or D_2 or both will get forward biased and will conduct, and the voltage V_X is approximately zero. This is insufficient to turn on D_3 , D_4 and Q hence the transistor is cut-off. The output is therefore HIGH.

When $A = B = \text{HIGH}$, both D_1 and D_2 are reverse-biased. Current flows from the supply (V_{CC}) through R_1 , D_3 and D_4 , and forward-biases the base-emitter junction of the transistor Q . The transistor will get into the saturation mode and the output of the gate is LOW.

The truth-table for this gate is shown below:

A	B	output
0	0	1
0	1	1
1	0	1
1	1	0

This is the truth-table for a NAND gate.

Assuming $V_{BE}(sat) = 0.7V$ and a diode drop $= 0.7V$, then V_X should be at least $2.1V$ to turn Q on. When either A or B or both are zero, $V_X = 0.7V$. This gives us a noise margin of $2.1V - 0.7V = 1.4V$. If we had used only one diode in place of $D3$ and $D4$, then V_X will need to be at least $1.4V$ to turn on the diode and transistor Q . The noise margin in this case would be $1.4V - 0.7V = 0.7V$. This means that using more diodes increases the noise margin. If both $D3$ and $D4$ were omitted, the voltage $V_X \geq 0.7V$ and hence transistor Q would be permanently on regardless of the inputs A and B (hence would not act as a NAND gate).

5.6 TTL gates

5.6.1 Introduction

Figure 5.6 shows a TTL NAND gate.

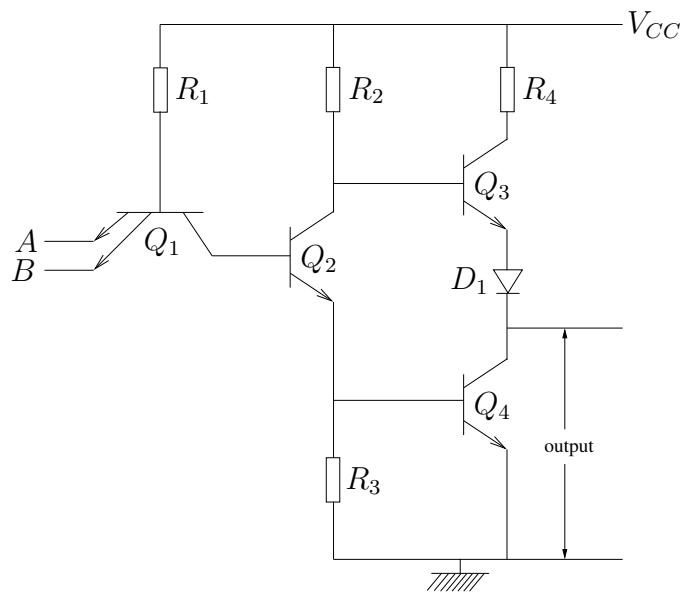


Figure 5.6: TTL NAND gate

Q_1 is a multi-emitter transistor. The transistor is equivalent to the circuit shown on Figure 5.7. (Note that the gate shown in Figure 5.6 is a 2-input NAND gate. A 3-input NAND gate would have a multi-emitter transistor with 3 emitters etc.)

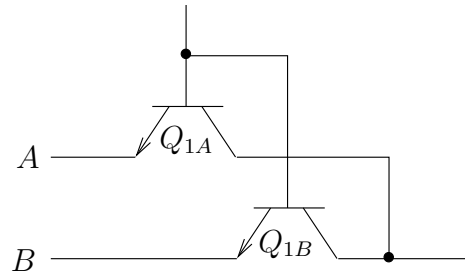


Figure 5.7:

5.6.2 Operation of the NAND gate

When both inputs are HIGH, the base-emitter junctions of the multi-emitter transistor are both reverse-biased, while the base-collector junctions are forward-biased. Q_1 is in cut-off mode. The V_{CC} supply pushes current through R_1 and the base-collector junction of Q_1 , and this turns Q_2 on (saturation mode). Current from the emitter of Q_2 will flow into the base of Q_4 and this turns Q_4 on (saturation). At the same time, the flow of Q_2 collector current produces a voltage across resistor R_2 that reduces Q_2 's collector voltage to a level that is insufficient to turn on Q_3 because of diode D_1 . The diode D_1 is needed to keep Q_3 OFF under these circumstances. Since Q_3 is OFF, the current from the supply does not pass through Q_4 . Transistor Q_4 's collector current comes from the output that the gate is connected to (*current sinking*). The flow of current in this mode can therefore be drawn as shown by the arrows in Figure 5.8.

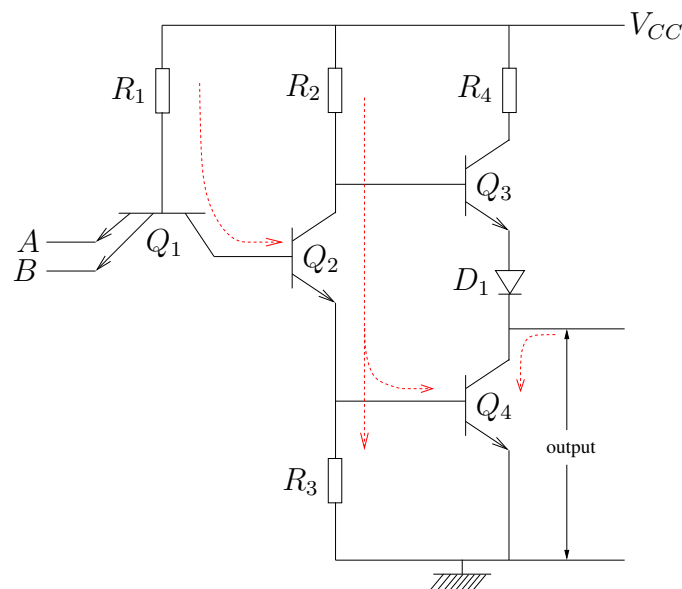


Figure 5.8:

When one or both inputs are LOW, one or both of Q_1 's base-emitter junctions will get forward-biased, hence Q_{1A} or Q_{1B} or both turn on so Q_1 is in saturation mode. In this case, the collector voltage of Q_1 is insufficient to turn on Q_2 and Q_4 so both of

these transistors are in cut-off mode. Since Q_2 's collector current is zero, the voltage at Q_3 's base is sufficient to forward-bias Q_3 and D_1 so that Q_3 gets into saturation mode. The flow of current is shown by the arrows on Figure 5.9.

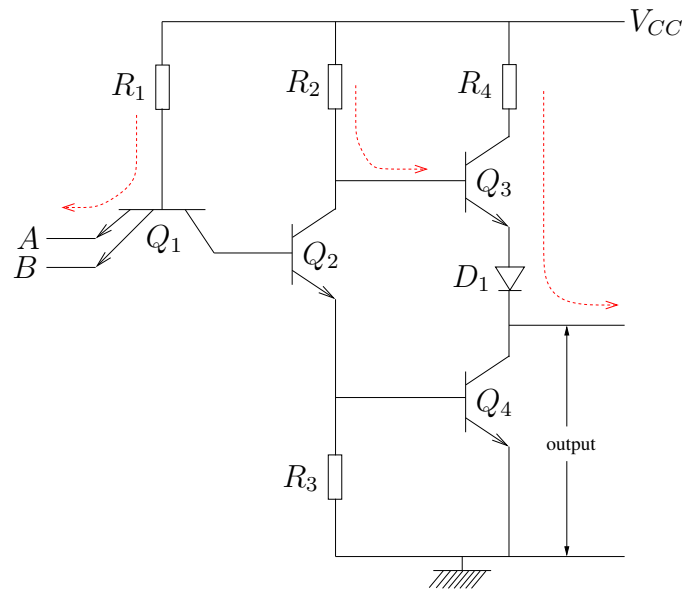


Figure 5.9:

The operation of the gate can be summarized by the truth-table below:

A	B	output
0	0	1
0	1	1
1	0	1
1	1	0

Exercise

Explain the working of the logic gate shown on Figure 5.10 and hence identify the logic gate.

5.6.3 Unconnected Inputs (floating inputs)

Unconnected inputs behave like logic 1 inputs (HIGH) because the emitter-base junctions at the inputs will not be forward-biased. *However, all unused inputs should be connected to a HIGH or LOW input as required, to prevent them from picking up noise voltages.*

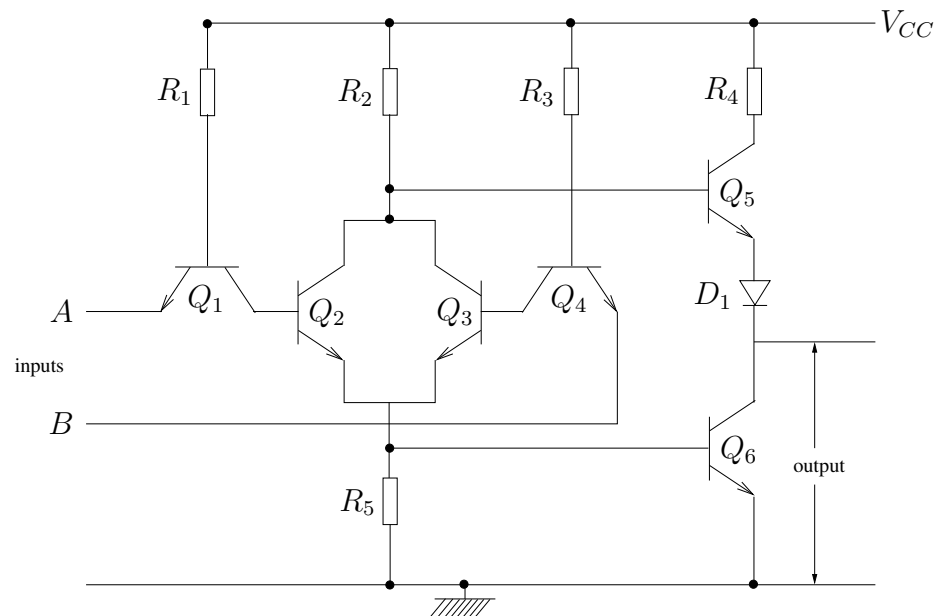


Figure 5.10: Exercise

5.6.4 Current Sourcing and Current Sinking – Revisited

The idea of current sourcing and current sinking can now be more appropriately illustrated using the NAND gate circuit. The case when the output of the driving gate is LOW is shown on Figure 5.11. (This is equivalent to the situation illustrated on Figure 5.4.)

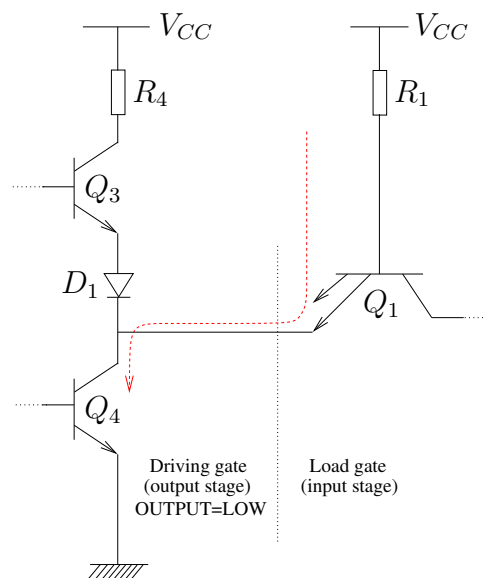


Figure 5.11: Current sinking

From the figure, we can see that Q_4 of the driving gate acts as a current sink, deriving its current from the load gate (current sinking).

Now consider the situation when the output of the driving gate is HIGH. This is illustrated on Figure 5.12 (equivalent to Figure 5.3).

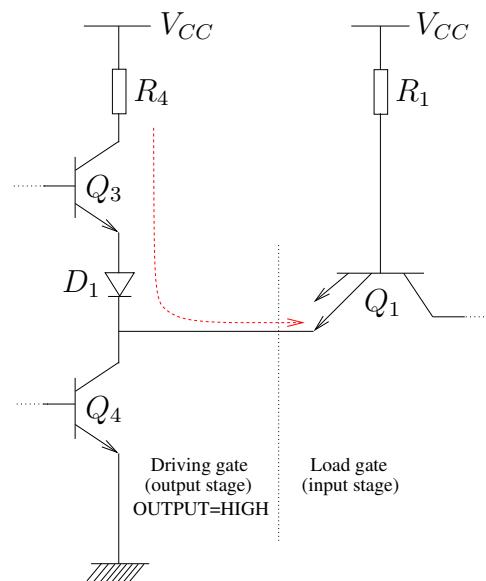


Figure 5.12: Current sourcing

This time, transistor Q_3 of the driving gate acts as a current source for the load gate (current sourcing). Note that in the case of current sourcing, the current is driven into a reverse-biased pn junction in the load gate so this current (I_{IH}) is very small, in the order of microamps.

5.6.5 TTL Loading (Fan-Out)

Consider the case where the output of the driving gate is HIGH (current sourcing). The circuit shown on Figure 5.13 is used to determine the fan-out.

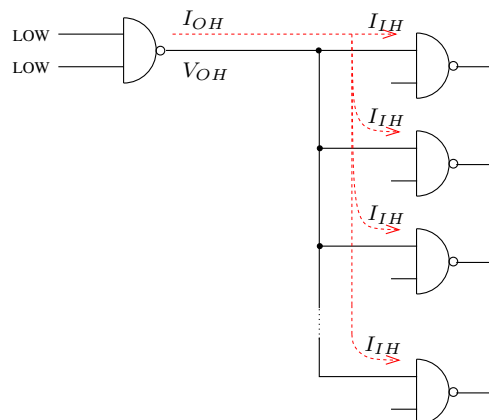


Figure 5.13: Fan out (HIGH)

Assuming that the driving gate is connected to n load gates, then we can write that:

$$I_{OH} = nI_{IH}$$

If too many gates are connected, I_{OH} will cause a higher voltage drop on R_4 (Figure 5.9), and this reduces V_{OH} . This is undesirable as it reduces the noise margin and it could make V_{OH} get into the indeterminate range.

$$\text{fan out (HIGH)} = \frac{I_{OH}(\max)}{I_{IH}(\max)}$$

In the case where the output of the driving gate is LOW, Figure 5.14 shows the diagram used to compute the fan-out.

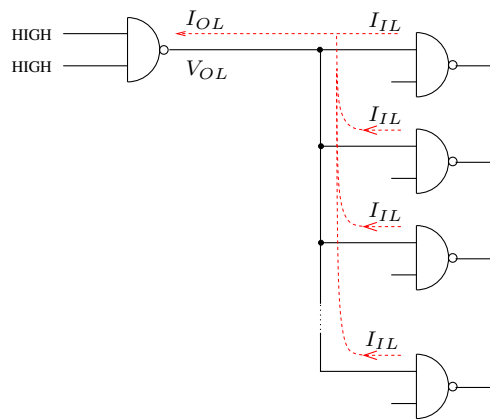


Figure 5.14: Fan out (LOW)

For n gates connected to the load gate,

$$I_{OL} = nI_{IL}$$

If too many gates are connected, I_{OL} will increase and cause V_{OL} to increase (*can you explain this?*). This reduces the noise margin and V_{OH} could get to the indeterminate range.

$$\text{fan out (LOW)} = \frac{I_{OL}(\max)}{I_{IL}(\max)}$$

The fan out of a logic family is the lower of the two values (fan-out (HIGH), fan-out (LOW)).

Example

The data below applies to logic families A and B. Use the table to determine:

- The DC noise margins for logic families A and B.
- The number of logic family B gates that can be satisfactorily driven by a logic family A gate.

LOGIC FAMILY PARAMETER	A			B			UNITS
	MIN	TYP	MAX	MIN	TYP	MAX	
V_{IH} Input Voltage (HIGH)	2			2			VOLTS
V_{IL} Input Voltage (LOW)			0.8			0.8	VOLTS
V_{OH} Output Voltage (HIGH)	2.4	3.4		2.7	3.4		VOLTS
V_{OL} Output Voltage (LOW)		0.2	0.4		0.35	0.5	VOLTS
I_{IH} Input Current (HIGH)			40			20	MICROAMPS
I_{IL} Input Current (LOW)			-1.6			-0.95	MILLIAMPS
I_{OH} Output Current (HIGH)			400			400	MICROAMPS
I_{OL} Output Current (LOW)			-16			-8	MILLIAMPS

Solution: (Note that on the table, TYP stands for Typical Value)

a) $V_{NH} = V_{OH}(\min) - V_{IH}(\min)$ and $V_{NL} = V_{IL}(\max) - V_{OL}(\max)$

Hence for logic family A, $V_{NH} = 2.4V - 2V = 0.4V$ and $V_{NL} = 0.8V - 0.4V = 0.4V$.

The two values are the same so noise margin for logic family A = 0.4V.

For logic family B, $V_{NH} = 2.7V - 2V = 0.7V$ and $V_{NL} = 0.8V - 0.5V = 0.3V$. The smaller of the two values is 0.3V, which is the noise margin for logic family B.

b) In this case, the logic family A gate is going to be the driving gate and logic family B gates the load gates. In the HIGH state, the number of B's gates that can be driven by A are given by:

$$\frac{I_{OH}(\max)(\text{for logic family A})}{I_{IH}(\max)(\text{for logic family B})} = \frac{400 \mu A}{20 \mu A} = 20$$

In the LOW state, the number of B's gates that can be driven by A

$$= \frac{I_{OL}(\max)(\text{for logic family A})}{I_{IL}(\max)(\text{for logic family B})} = \frac{16 \text{ mA}}{0.95 \text{ mA}} = 16.8 = 16$$

The number gates of logic family B that can be satisfactorily driven by A logic family A gate is the smaller of the two numbers, = 16.

5.6.6 Totem Pole Outputs

The output stage of the NAND gate described above is called a totem-pole output. It is illustrated on Figure 5.15.

All TTL gates have this type of output stage. Note that the NAND gate circuit shown on Figure 5.6 would still operate as a NAND gate even if D_1 and Q_3 were removed (and R_4 connected directly to Q_4). However, removing D_1 and Q_3 is not desirable as R_4 is usually a small resistor (about 100Ω) so a large current would flow through Q_4 when it is in saturation mode. When Q_3 is in the circuit, and Q_4 is in saturation mode, Q_3 is in cut-off mode hence there will be no current through R_4 , and this keeps the power dissipation of the circuit low. Note also that the transistors Q_3 and Q_4 operate such that when Q_4 is saturated, Q_3 is cut-off, and when Q_4 is cut-off, Q_3 is saturated.

Totem pole outputs should not be connected directly together (connecting two outputs directly together is known as a wired-AND operation). Consider the case shown on Figure 5.16 where two totem-pole outputs are connected together directly.

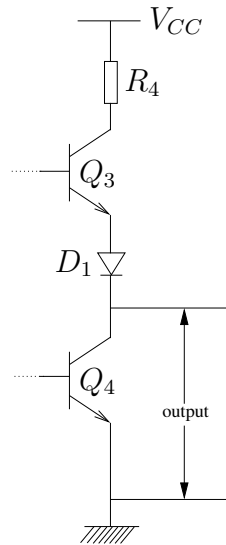


Figure 5.15: Totem pole output

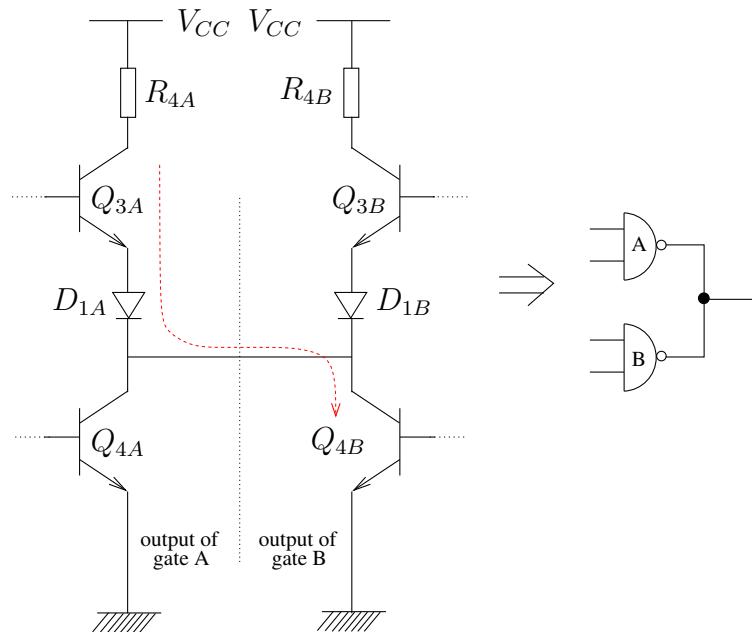


Figure 5.16: Wired-AND connection

No particular problems arise if both inputs are HIGH or are both LOW. A problem arises if one input is HIGH and the other one LOW. In the case shown on Figure 5.16, it is assumed that the output of gate A is HIGH while that of gate B is LOW. A large current flows in the direction shown by the arrow. This current is usually large enough to damage transistors Q_{3A} and Q_{4B} , thereby destroying the gates. Even if by some chance the gates did not get damaged, the high current drawn will lead to a very high power consumption (and make the ICs get very hot), and the voltage at the point where the two outputs are connected together will get into the indeterminate region.

5.6.7 TTL Open-Collector Outputs

Figure 5.17 shows an TTL open-collector NAND gate.

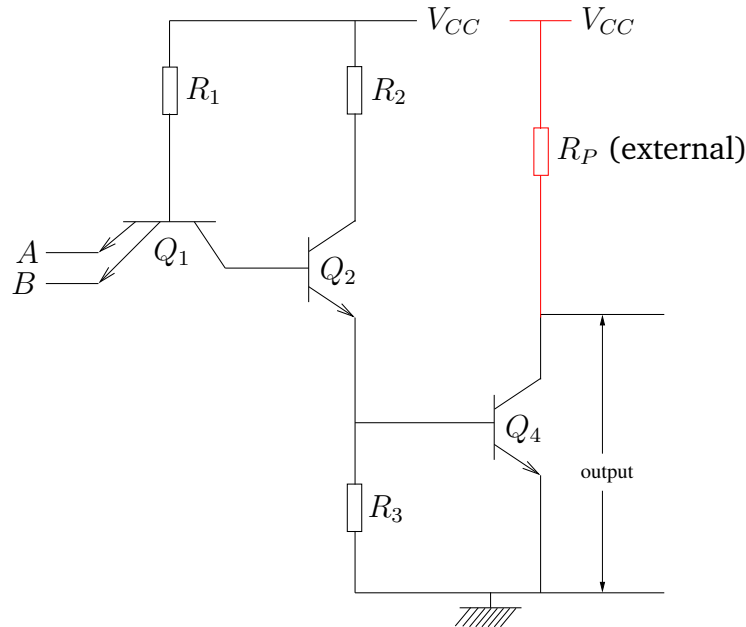


Figure 5.17: TTL open-collector NAND gate

When the output is LOW, the transistor Q_4 is in saturation mode hence $V_{out} = V_{CE(sat)} \simeq 0$. When the output is HIGH, Q_4 is cut-off. For a HIGH level voltage to appear, an external resistor R_P (known as a pull-up resistor) is connected to the collector of Q_4 as shown in the diagram. Without R_P , there would be no output voltage when Q_4 is cut-off. (Refer to the references at the end of this handout for methods of determining the value of R_P .)

Open-collector outputs allow us to perform the wired-AND operation, which is not possible with totem-pole outputs. As an example, if we were to implement the function:

$$x = \overline{AB} \cdot \overline{CD} \cdot \overline{EF}$$

using open-collector TTL gates, the circuit would look as shown on Figure 5.18.

The AND gate shown in dotted in Figure 5.18 is just symbolic – it means that connecting the three (open-collector) outputs together is equivalent to an AND gate. The pull-up resistor R_P is necessary to observe the output x . The open-collector gate circuit design therefore generally results in the use of fewer gates than totem-pole gates design, as we can directly connect open-collector outputs together, which is not possible with TTL outputs. The open-collector gates are also designed to handle a relatively large current and are used for driving devices like LEDs and relays.

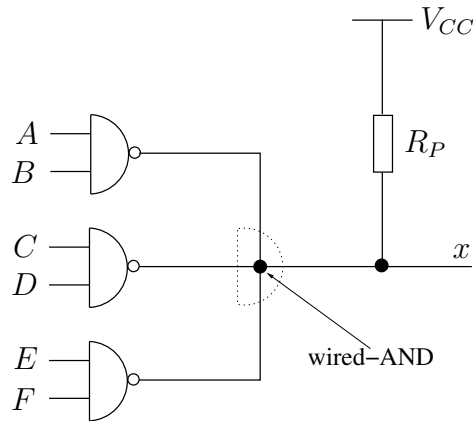


Figure 5.18: Wired-AND operation using open-collector gates

5.6.8 Tristate Devices

Tristate devices have outputs that can be get to three possible states: HIGH, LOW and High Impedance (Hi-Z). An example of a tristate TTL inverter is shown on Figure 5.19.

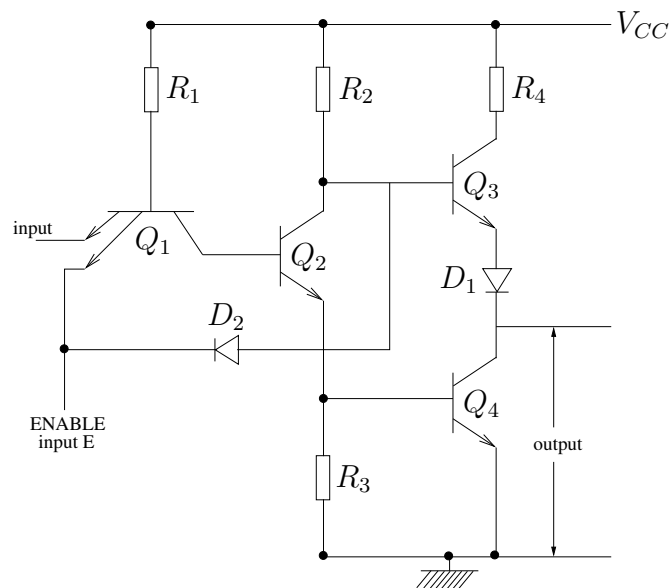


Figure 5.19: Tristate TTL inverter

When $E = 1$, the circuit operates as an inverter since E has no effect on Q_1 or Q_3 . We say that the gate is enabled when $E = 1$.

When $E = 0$, Q_1 is in saturation mode, hence Q_2 and Q_4 are cut-off. At the same time, Q_3 is also in cut-off mode. Since both transistors on the totem-pole output are in cut-off mode, the output is essentially an open circuit. We say that when $E = 0$, the gate is disabled, and the output is in the High Impedance state, hence the input to the gate has no effect on the output. These situations are illustrated on Figure 5.20.

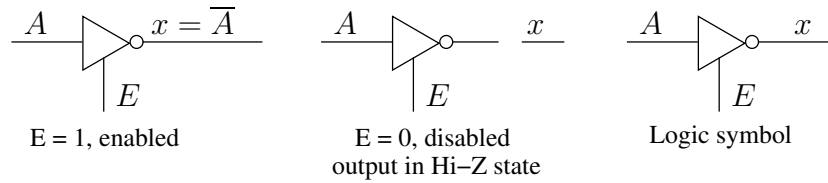


Figure 5.20: Tristate inverter

Note from the figure that when $E = 0$, the output of the gate is in the Hi-Z state, which is equivalent to having the output being disconnected from the input. *Tristate ability allows many devices to be connected on the same output line, with one device enabled at a time.*

5.6.9 TTL 74 series

- a) 74XX e.g. 7400, 7432 e.t.c. - This is the standard TTL. In this case, $R1$ (Figure 5.6) = $4K$, $R2 = 1K6$, $R3 = 1K$ and $R4 = 130\Omega$.
- b) 74LXX e.g. 74L00, 74L32 e.t.c. - Low power TTL - Uses resistors of higher values than standard TTL. $R1 = 40K$, $R2 = 20K$, $R3 = 12K$ and $R4 = 5K$.
- c) 74HXX e.g. 74H00, 74H32 e.t.c. - High speed TTL - Uses resistors of lower values than standard TTL, has a higher power consumption than standard TTL. $R1 = 2K8K$, $R2 = 760\Omega$, $R3 = 470\Omega$ and $R4 = 58\Omega$.
- d) 74SXX e.g. 74S00, 74S32 e.t.c. - Uses Schottky transistors for high-speed switching - can work up to speeds of 100MHz.
- e) 74LSXX e.g. 74LS00, 74LS32 e.t.c. - Low power Schottky - low power version of 74SXX series.
- f) 74ALSXX e.g. 74ALS00, 74ALS32 e.t.c. - Advanced Low Power Schottky - has a lower power consumption than 74LSXX series and can operate at higher speeds than 74SXX series.

5.6.10 Comparison between TTL and CMOS

- a) **Power Consumption:** In TTL devices, power is consumed at constant rate. CMOS devices have a low power consumption since they only consume a significant amount of power when switching (HIGH to LOW or LOW to HIGH) - very little power is consumed when the gate is in a stable HIGH or stable LOW state. (*Note that among all logic families, ECL has the highest power consumption.*)

- b) **Noise Margin:** With a supply voltage of $V_{CC} = 5V$, TTL devices have a noise margin of about $0.8V$. CMOS devices have a noise immunity $\simeq \frac{1}{3}V_{CC}$, which is generally higher than that of TTL devices. (*ECL has the lowest noise margin among the logic families.*)
- c) **Power Supply Voltage:** TTL devices should be powered with a supply voltage of $5V \pm 5\%$. The 4XXX and 74C series of CMOS devices can be powered with voltages between $3V$ and $15V$, while the 74HC and the 74HCT CMOS series can operate with power supply voltages of $2V$ to $6V$. CMOS devices are therefore less sensitive to power supply voltages.
- d) **Speed:** TTL devices have higher speed than CMOS devices. TTL devices have an average propagation delay of about $10ns$, compared to about $50ns$ for the 4XXX CMOS series. (*ECL has the highest speed among the logic families - used where speed is the main consideration.*)
- e) **Fan-out** CMOS devices have a higher fan-out than TTL devices.

Exercise

Figure 5.21 shows a CMOS inverter. Explain its operation.

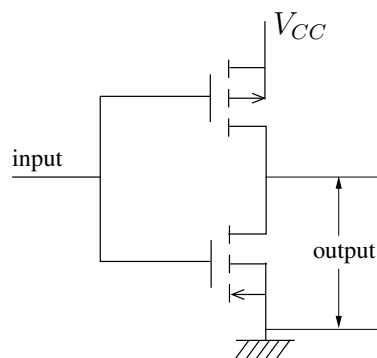


Figure 5.21: CMOS inverter

(Refer to the references at the end of this handout for more CMOS gates)

5.6.11 CMOS handling precautions

CMOS devices are easily damaged by static charges, making them electrically delicate and hence the following precautions should be taken when working with these devices:

1. Do not handle CMOS ICs unless your body is connected with a wrist strap to earth ground.

2. Tools used to straighten pins should be grounded.
3. Do not remove or replace a CMOS IC in a socket with power ON
4. All unused CMOS gate inputs must be connected to an active gate input, V_{CC} or ground. If allowed to float, an input can take on enough static charges to damage the IC.

5.6.12 CMOS logic sub-families

1. CD 4XXX e.g. CD 4001 Quad 2-input NOR gate IC - this is the oldest and it is no longer manufactured. It is also the slowest among the CMOS logic families.
2. 74CXX - Standard CMOS - these are pin-compatible with TTL devices (e.g. the CMOS 74C08 and the TTL 7408 are both AND gate ICs and have the same pin configuration. *However, they are not electrically compatible.*
3. 74HCXX - High Speed CMOS - has a speed comparable to the TTL 74LSXX series. This is also pin compatible but not electrically compatible with TTL devices.
4. 74HCTXX - High Speed, TTL compatible CMOS - This is pin comparable and electrically compatible with TTL devices. This means that you can construct a circuit having TTL and 74HCT ICs.

Chapter 6

MSI COMBINATIONAL LOGIC CIRCUITS AND THEIR APPLICATIONS

6.1 Introduction

Digital ICs are often categorized according to their circuit complexity as measured by the number of gates they contain:

Complexity	Number of gates
Small Scale Integration (SSI)	Less than 12 gates
Medium Scale Integration (MSI)	12 to 99 gates
Large Scale Integration (LSI)	100-9999 gates
Very Large Scale Integration (VLSI)	More than 10000 gates

6.2 Medium Scale Integrated (MSI) devices

MSI devices are used as building blocks for more complex digital circuits. They perform specific digital functions commonly needed in the design of combinational logic circuits and they are available in IC form.

Digital circuit design using MSI devices results in significant reduction in the number of ICs required to implement logic circuits as compared to circuit implementation using SSI devices. As an example, suppose we would like to implement the Boolean function

$$f(A, B, C, D) = \sum m(4, 5, 7, 9, 11, 15). \quad (6.1)$$

To implement this function using SSI devices, it has to be simplified first, and this results in the Boolean expression

$$f(A, B, C, D) = \bar{A}B\bar{C} + BCD + A\bar{B}D. \quad (6.2)$$

Using AND-OR-NOT logic, this circuit requires three 3-input AND gates, one 3-input OR gate and 3 inverters, a minimum of 3 SSI ICs. If the circuit were to be implemented using NAND gates only, three 2-input NAND gates and four 3-input NAND gates would be required, again requiring a minimum of 3 SSI ICs. However, as we shall later see, it is possible to implement this function using a single multiplexer IC without any additional components, which considerably reduces the circuit complexity, power consumption and increases circuit reliability. Furthermore, for implementation using a multiplexer, the Boolean function does not need to be simplified first.

Examples of MSI devices are decoders, encoders, multiplexers, demultiplexers, adders, subtractors and magnitude comparators.

6.3 Decoders

6.3.1 The decoder function

A decoder is a logic circuit which converts an N -bit binary input into a maximum of M output lines such that each output line will be activated for only one of the possible combination of inputs.

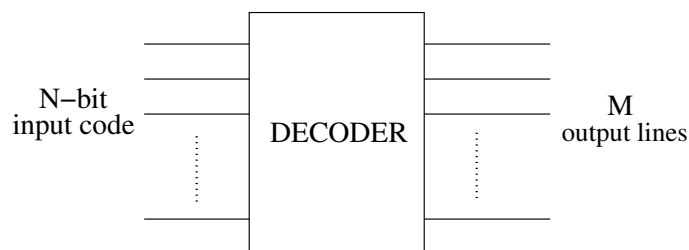


Figure 6.1: A decoder

In general for a decoder, $M \leq 2^N$. For each combination of inputs, only one of the output lines will be activated. The activated output can be HIGH (Logic 1) while other outputs are LOW (Logic 0), or the activated output can be LOW while all other outputs are HIGH. If the activated output of the decoder is HIGH while other outputs are LOW, the decoder is said to have active-HIGH outputs, and if the activated output is LOW with all other outputs HIGH, the decoder is said to have active-LOW outputs. The logic symbols for decoders with active-HIGH outputs and active-LOW outputs are shown in Figure 6.2.

Figure 6.3 shows an example of a decoder.

This decoder generates all the minterms for the input variables A (MSB), B and C

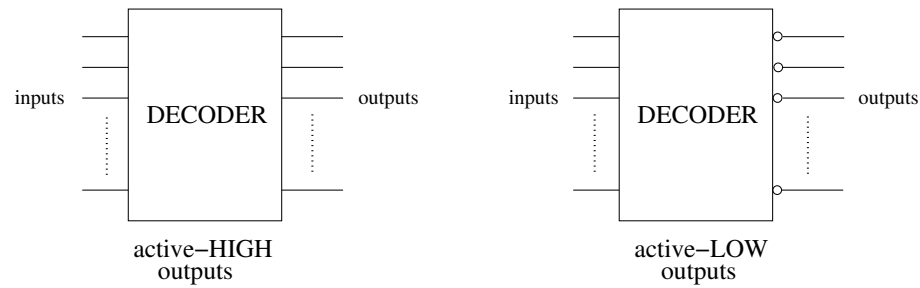


Figure 6.2: Decoders with active-HIGH and active-LOW outputs

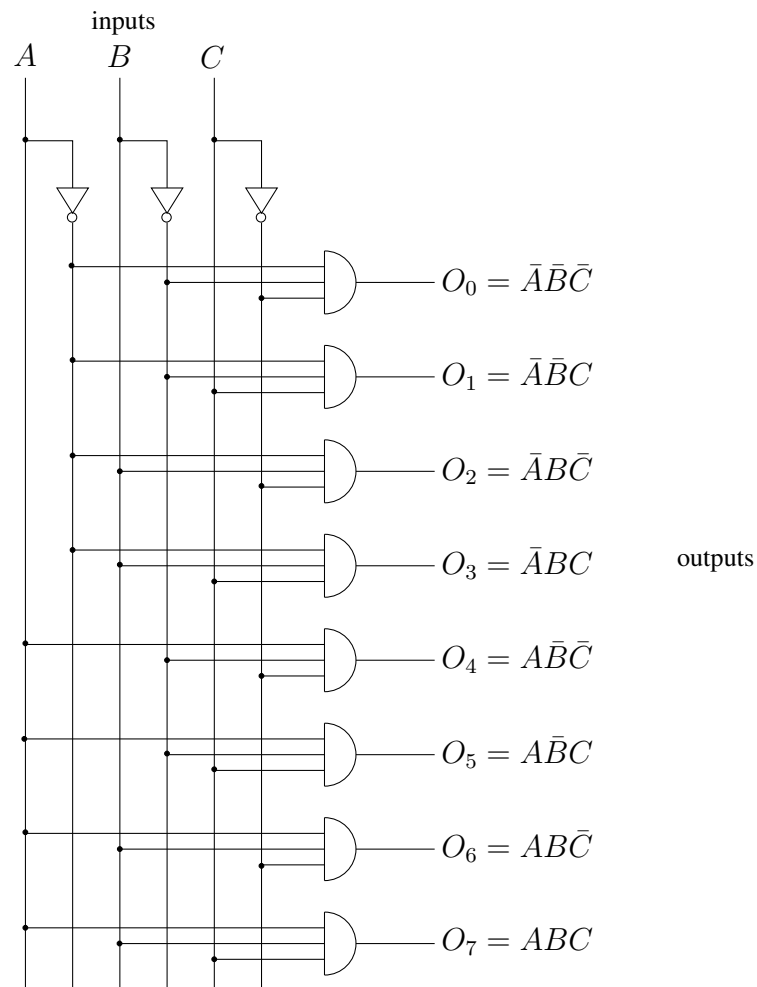


Figure 6.3: 3 line to 8 line decoder

(LSB). This decoder can thus be thought of as a minterm generator. This makes it useful in the design of combinational logic circuits. The truth-table for this decoder is given below.

<i>A</i>	<i>B</i>	<i>C</i>	<i>O</i> ₀	<i>O</i> ₁	<i>O</i> ₂	<i>O</i> ₃	<i>O</i> ₄	<i>O</i> ₅	<i>O</i> ₆	<i>O</i> ₇
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

The decoder shown in Figure 6.3 is referred to as

- 3 line to 8 line decoder (since it has 3 inputs and 8 outputs), or
- binary to octal decoder (since the input is a 3-bit binary code and at any time, only one of 8 outputs (octal) is activated), or
- 1 of 8 decoder (since only one of the eight outputs is activated at a time).

6.3.2 ENABLE inputs

Some decoders have one or more ENABLE inputs that are used to control the operation of the decoder. If a decoder has ENABLE inputs, it will only function normally if the ENABLE inputs are at the right logic levels. Most of the decoders which are available in IC form have ENABLE inputs.

The ENABLE inputs are used for two reasons:

- The ENABLE inputs make it possible to connect several decoders together to form a larger decoder circuit.
- The ENABLE inputs make it possible to use a decoder as a demultiplexer.

6.3.3 The 74138 1 of 8 decoder

The circuit diagram of the 74138 1 of 8 decoder IC is shown in Figure 6.4.

The truth table showing how the decoder responds to the enable inputs E_0 , E_1 and E_2 is shown below.

E_0	E_1	E_2	decoder outputs
1	X	X	decoder disabled – all outputs HIGH
X	1	X	decoder disabled – all outputs HIGH
X	X	0	decoder disabled – all outputs HIGH
0	0	1	decoder enabled – outputs respond to inputs A , B , C

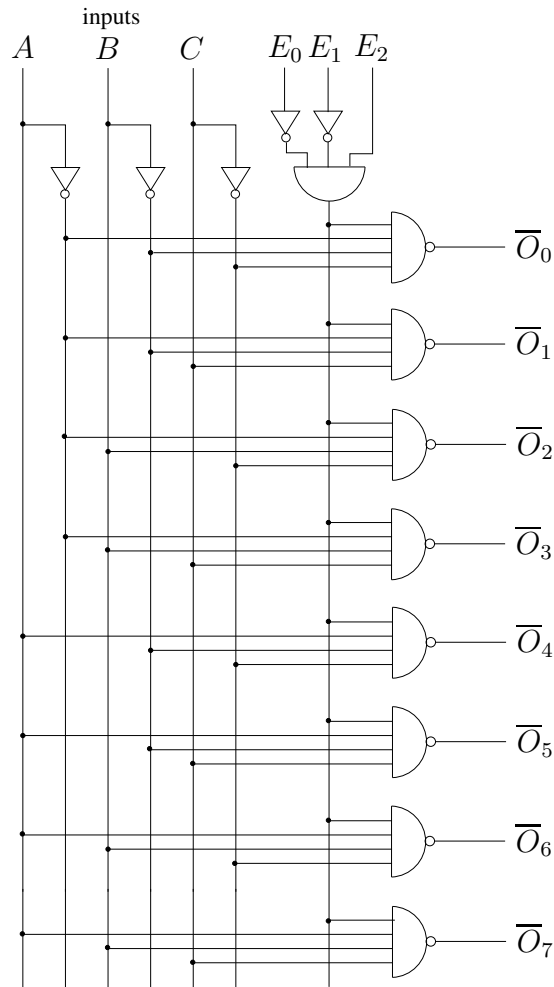


Figure 6.4: Circuit diagram for the 74138 3 line to 8 line decoder

The logic symbol for the 74138 decoder is shown in Figure 6.5. (A_0 is the LSB while A_2 is the MSB.)

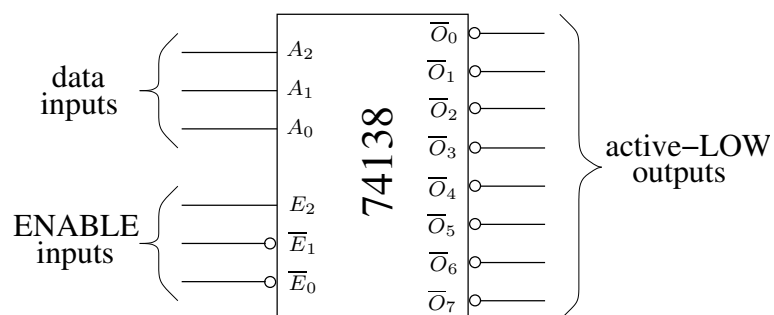


Figure 6.5: The 74138 decoder

Example

Design a 4-line to-16 decoder based on the 74138 decoder.

Solution

The 74138 decoder has 3 data inputs and 8 outputs. To build a 4-line-to-16-line decoder, two 74138 decoders are needed. The 4-line-to-16-line decoder has 4 data inputs A_3 (MSB), A_2 , A_1 and A_0 (LSB) but since the 74138 decoder has only 3 data inputs A_2 (MSB), A_1 and A_0 (LSB), the inputs A_2 , A_1 and A_0 are connected directly to the data inputs of the 74138 decoders while the data input A_3 is connected to the ENABLE inputs of the decoders in a suitable manner such that when $A_3 = 0$, one of the decoders is selected, and when $A_3 = 1$, the other decoder is selected.

Figure 6.6 shows the 4-line-to-16 line decoder.

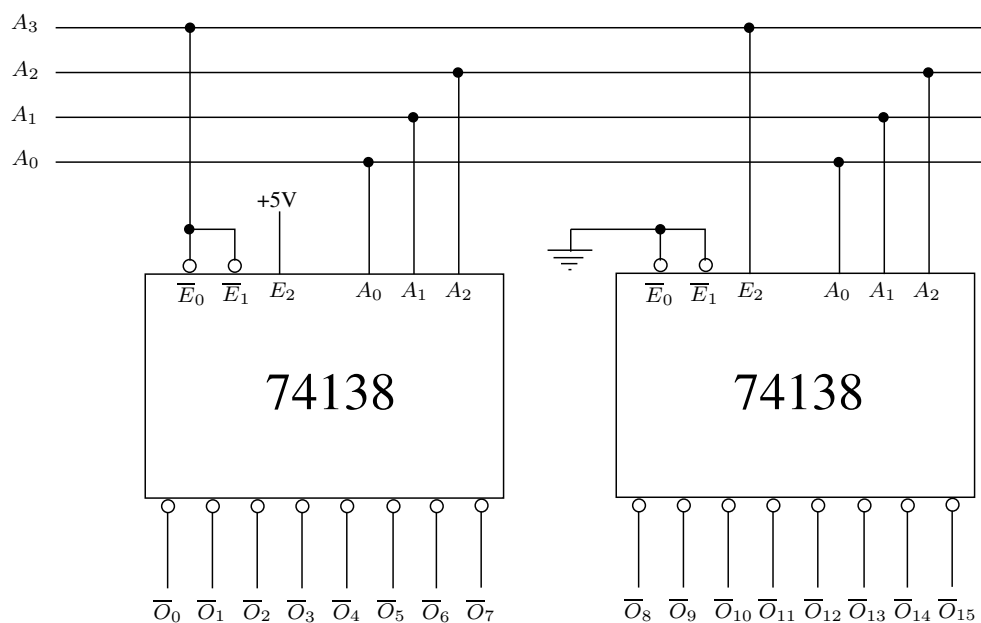


Figure 6.6: 4-line-to-16-line decoder constructed two 74138 decoders

Exercise

You are provided with 2-line-to-4-line decoders of the form shown in Figure 6.7. (A_0 is the LSB while A_1 is the MSB.) Using as many of these decoders as you need and no other components, design a 4-line-to-16-line decoder.

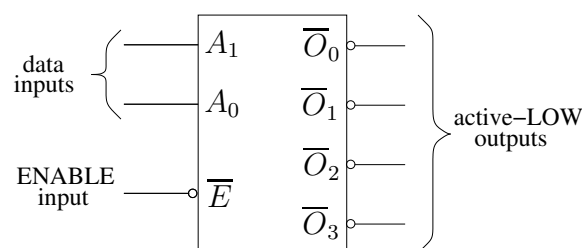


Figure 6.7: 2-line-to-4-line decoder

6.3.4 BCD to decimal decoder

The 7442 is a 4-line-to-10-line decoder (or a 1 of 10 decoder). This decoder does not have an enable input. This decoder is shown in Figure 6.8.

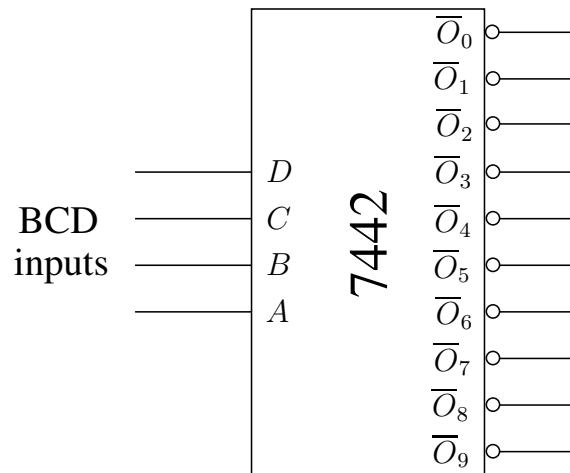


Figure 6.8: The 7442 BCD to decimal decoder

The truth-table for the 7442 decoder is given below:

<i>D</i>	<i>C</i>	<i>B</i>	<i>A</i>	active output
0	0	0	0	\overline{O}_0
0	0	0	1	\overline{O}_1
0	0	1	0	\overline{O}_2
0	0	1	1	\overline{O}_3
0	1	0	0	\overline{O}_4
0	1	0	1	\overline{O}_5
0	1	1	0	\overline{O}_6
0	1	1	1	\overline{O}_7
1	0	0	0	\overline{O}_8
1	0	0	1	\overline{O}_9
1	0	1	0	None
1	0	1	1	None
1	1	0	0	None
1	1	0	1	None
1	1	1	0	None
1	1	1	1	None

6.3.5 Decoder applications

Applications of decoders include

- ◇ Implementation of combinational logic circuits
- ◇ Selection of memory chips (ICs) when the input code is an address from the central processing unit of a computer.
- ◇ Displaying of data on decimal readouts.

- ◇ Sequencing operations (when the input code is from a counter).
- ◇ Used as demultiplexers.

Implementation of combinational logic circuits

In general, a decoder provides 2^N minterms for N input variables. Since any Boolean function can be expressed as a sum of minterms, a decoder can be used to generate the minterms and an external OR gate used to form the sum. (If the decoder has active-LOW outputs, the outputs are connected using a NAND gate).

Example

Implement the Boolean function

$$S(X, Y, Z) = \sum m(1, 2, 4, 7) \quad (6.3)$$

$$C(X, Y, Z) = \sum m(3, 5, 6, 7) \quad (6.4)$$

using

- a) a 1 of 8 decoder shown in Figure 6.9,
- b) a 74138 decoder.

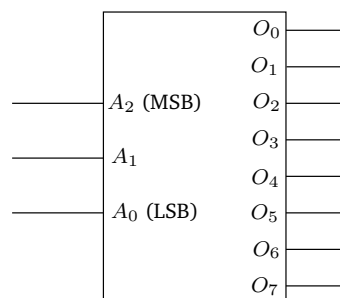


Figure 6.9:

Solution

a)

b)

$$\begin{aligned} S(X, Y, Z) &= \sum m(1, 2, 4, 7) = m_1 + m_2 + m_4 + m_7 \\ &= \overline{\overline{m_1 + m_2 + m_4 + m_7}} = \overline{\overline{m_1} \cdot \overline{m_2} \cdot \overline{m_4} \cdot \overline{m_7}}. \end{aligned}$$

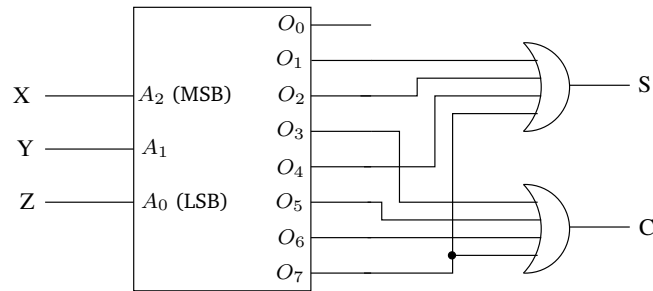


Figure 6.10:

Similarly,

$$C(X, Y, Z) = \overline{\overline{m_3} \cdot \overline{m_5} \cdot \overline{m_6} \cdot \overline{m_7}}.$$

The circuit implementation using a 74138 decoder is shown in Figure 6.11. Note that the decoder is permanently enabled.

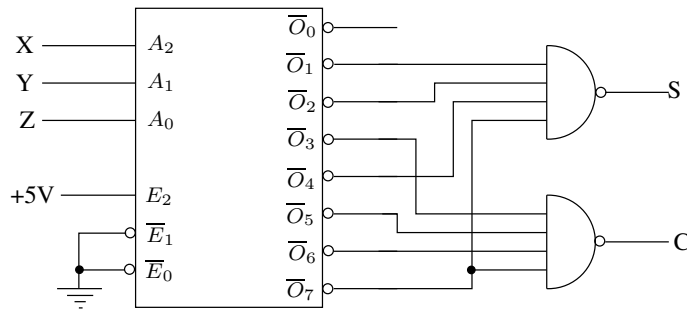


Figure 6.11:

Exercise

Design a 3-bit binary to Gray code converter using a 74138 decoder and a minimum of logic gates.

Displaying of data on decimal readouts

In this case, a decoder is used to provide outputs to drive a display so that the binary number is displayed as a decimal digit. When a decoder is combined with a high current driving output, it is called a decoder/driver. The high current driving output is required because LEDs used in many decimal displays require a high current which an ordinary decoder cannot provide.

A commonly used display is the 7-segment display shown in Figure 6.12. The segments are either LEDs or Liquid Crystal Displays (LCDs). The 7-segment LED displays are available in two configurations: the common-anode connection or the common cathode connection. These configurations are illustrated in Figure 6.13.

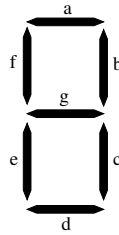


Figure 6.12: A 7-segment display

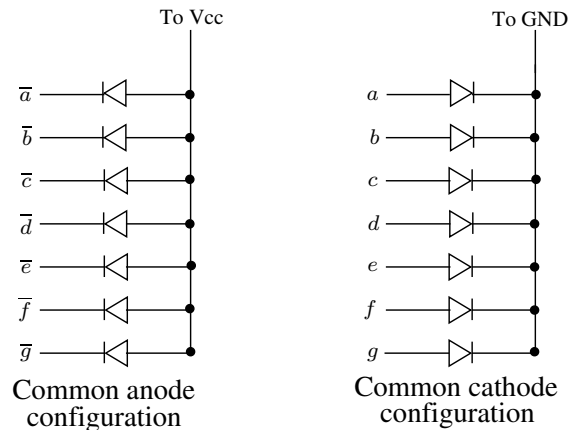


Figure 6.13: Common-anode and common-cathode configurations of 7-segment displays

A BCD-to-7-segment decoder/driver is used to take a 4-bit BCD input and provide the outputs that will pass current through the appropriate segments of a 7-segment display to display a decimal digit. One such a device is the 7447, illustrated in Figure 6.14. The decoder has active-LOW outputs.

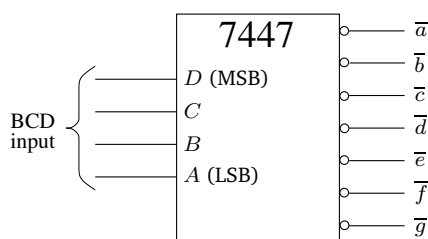


Figure 6.14: A 7447 decoder/driver

The truth-table for the 7447 BCD-to-7-segment decoder/driver is given below.

D	C	B	A	\bar{a}	\bar{b}	\bar{c}	\bar{d}	\bar{e}	\bar{f}	\bar{g}
0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	1	0	0	1	1	1	1
0	0	1	0	0	0	1	0	0	1	0
0	0	1	1	0	0	0	0	1	1	0
0	1	0	0	1	0	0	1	1	0	0
0	1	0	1	0	1	0	0	1	0	0
0	1	1	0	0	1	0	0	0	0	0
0	1	1	1	0	0	0	1	1	1	1
1	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	1	0	0
1	0	1	0	1	1	1	1	1	1	1
1	0	1	1	1	1	1	1	1	1	1
1	1	0	0	1	1	1	1	1	1	1
1	1	0	1	1	1	1	1	1	1	1
1	1	1	0	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
1	1	1	0	1	1	1	1	1	1	1

Note from the truth table above that if the input code is not a valid BCD code, none of the decoder outputs is activated (all outputs are HIGH). Also note that for valid BCD codes, more than one output is activated for each binary code, but nevertheless, the 7447 is still referred to as a decoder.

Figure 6.15 shows how a 7447 can be connected to a 7-segment display. Since this decoder has active-LOW outputs, it is connected to a common-anode 7-segment display.

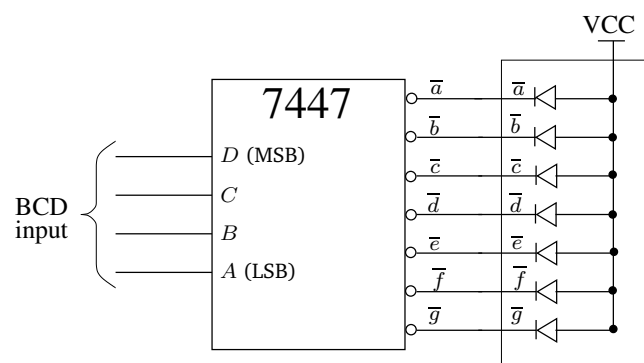


Figure 6.15: A 7447 decoder/driver connected to a common-anode 7 segment display

6.4 Encoders

An encoder has a number of input lines, only one of which is activated at a given time, and produces an N -bit output code, which depends on which input is activated.

An encoder is functionally the opposite of an encoder. As an example, a binary-to-octal decoder accepts a 3-bit binary input code and activates only one of 8 output lines. An octal to binary encoder operates in the opposite manner in that it has 8

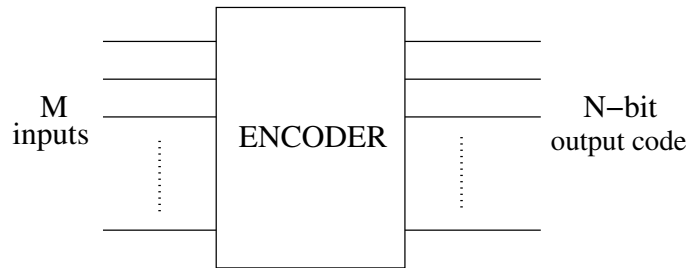


Figure 6.16: An encoder

input lines, only one of which is activated at a given time, and it produces a 3-bit binary output code which depends on which input line is activated. An octal-to-binary encoder is shown in Figure 6.17

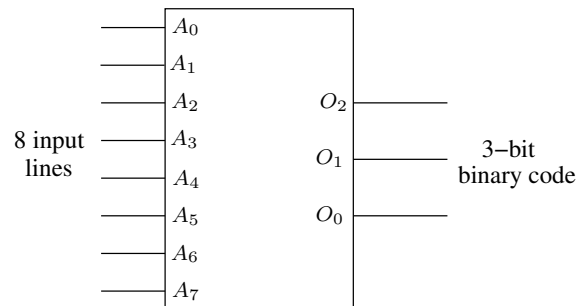


Figure 6.17: Octal to binary encoder

The truth-table for this encoder is given below:

A_0	A_1	A_2	A_3	A_4	A_5	A_6	A_7	O_2	O_1	O_0
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

From the truth-table above, we can write that

$$O_2 = A_4 + A_5 + A_6 + A_7,$$

$$O_1 = A_2 + A_3 + A_6 + A_7,$$

$$O_0 = A_1 + A_3 + A_5 + A_7.$$

The above equations can be implemented using the circuit shown in Figure 6.18.

From the truth-table above, we can see that not all the possible combinations of the input variables are considered hence the circuit shown in Figure 6.18 will work properly only if only one output is activated at a time. This problem is solved by use of a priority encoder. A priority encoder works in such a way that if more than one

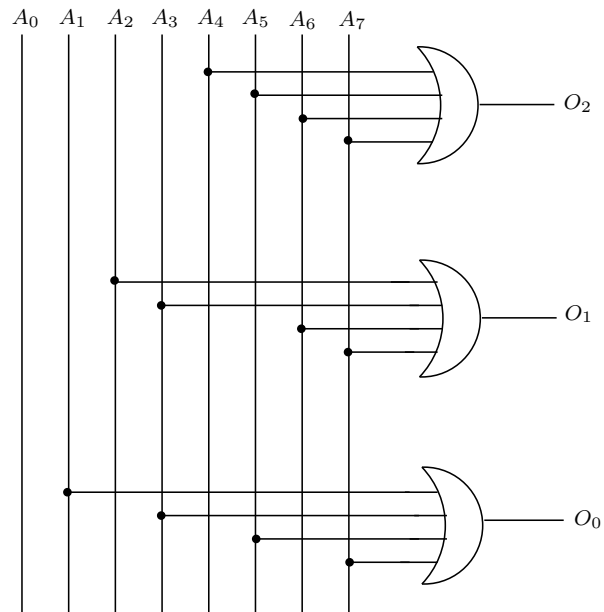


Figure 6.18:

input is activated, the output code will correspond to the highest numbered of the activated inputs. (For more information on how to design priority encoders, see the references at the end of this handout.) Examples of priority encoders are the 74148 8-line-to-3-line priority encoder (with active-HIGH outputs), and the 74147 decimal-to-BCD priority encoder (with active-LOW outputs). The 74147 encoder is shown in Figure 6.19.

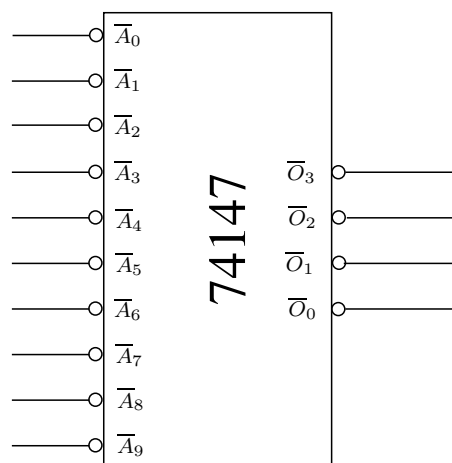


Figure 6.19: The 74147 decimal to BCD priority encoder

The truth-table for the 74147 is given below:

\bar{A}_0	\bar{A}_1	\bar{A}_2	\bar{A}_3	\bar{A}_4	\bar{A}_5	\bar{A}_6	\bar{A}_7	\bar{A}_8	\bar{A}_9	\bar{O}_3	\bar{O}_2	\bar{O}_1	\bar{O}_0	
1	1	1	1	1	1	1	1	1	1	1	1	1	1	all outputs inactive A_0 activated
0	1	1	1	1	1	1	1	1	1	1	1	1	1	
X	0	1	1	1	1	1	1	1	1	1	1	1	0	
X	X	0	1	1	1	1	1	1	1	1	1	0	1	
X	X	X	0	1	1	1	1	1	1	1	1	0	0	
X	X	X	X	0	1	1	1	1	1	1	0	1	1	
X	X	X	X	X	0	1	1	1	1	1	0	1	0	
X	X	X	X	X	X	0	1	1	1	1	0	0	1	
X	X	X	X	X	X	X	0	1	1	1	0	0	0	
X	X	X	X	X	X	X	X	0	1	0	1	1	1	
X	X	X	X	X	X	X	X	X	0	0	1	1	0	

The 74147 is commonly used as a switch encoder. Consider the circuit shown in Figure 6.20.

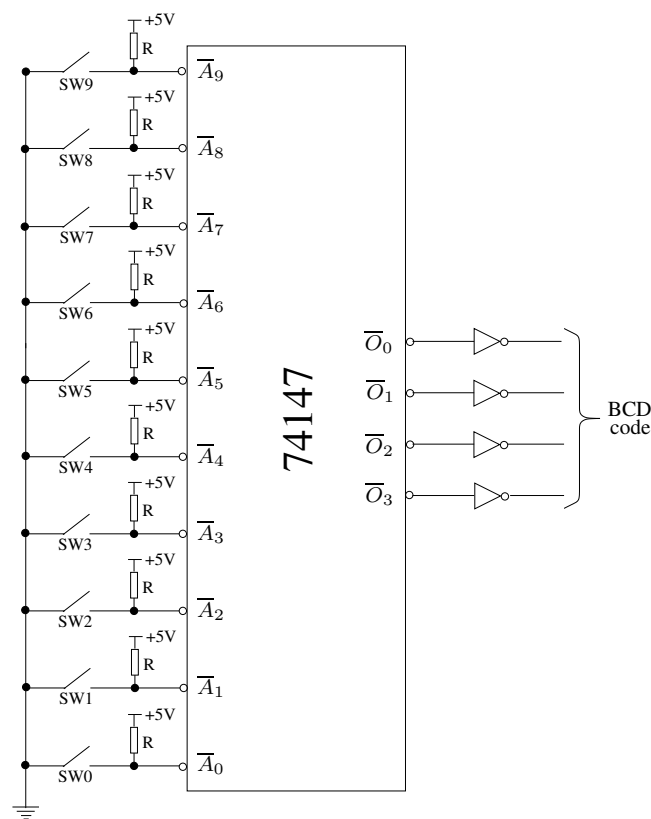


Figure 6.20: Switch encoder

The switches may be keyboard switches on a calculator representing digits 0 to 9. When no key is pressed, the output is 0000. When a digit is pressed, the output will be the BCD code for that digit. The 74147 is a priority encoder, so pressing two keys simultaneously will produce the BCD code for the bigger digit.

6.5 Multiplexers

6.5.1 The multiplexer function

Multiplexing means transmitting a large number of information units over a smaller number of channels or lines. A multiplexer is a logic circuit that accepts several data inputs and allows only one of them at a time to get to the output. The routing of the desired data input to the output is controlled by SELECT inputs, also known as ADDRESS inputs. A multiplexer is analogous to a multi-position switch.

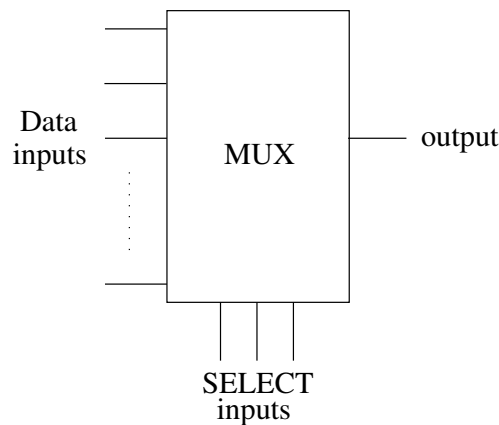


Figure 6.21: A multiplexer

6.5.2 2-input multiplexer

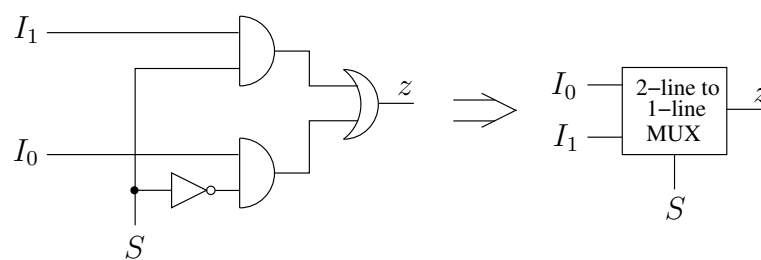


Figure 6.22: 2-line to 1-line multiplexer

Consider the circuit shown in Figure 6.22. The output is given by

$$z = I_1 S + I_0 \bar{S}.$$

The truth-table for the circuit is given below:

S	z
0	I_0
1	I_1

It is clear from the truth-table that the input signals are routed to the output depending on the SELECT input S .

6.5.3 4-input multiplexer

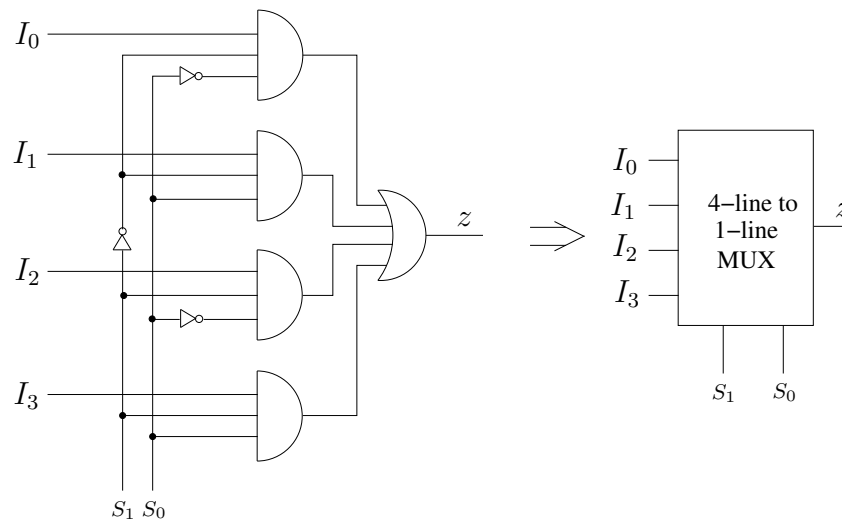


Figure 6.23: 4-line to 1-line multiplexer

From the circuit shown in Figure 6.23, the output is given by

$$z = I_0 \bar{S}_1 \bar{S}_0 + I_1 \bar{S}_1 S_0 + I_2 S_1 \bar{S}_0 + I_3 S_1 S_0.$$

The truth-table for the circuit is given below:

S_1	S_0	z
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

6.5.4 4-input multiplexer with ENABLE input

Figure 6.24 shows a 4-line to 1-line multiplexer with an ENABLE input. The truth-table for this multiplexer is given below:

E	S_1	S_0	z
0	X	X	0
1	0	0	I_0
1	0	1	I_1
1	1	0	I_2
1	1	1	I_3

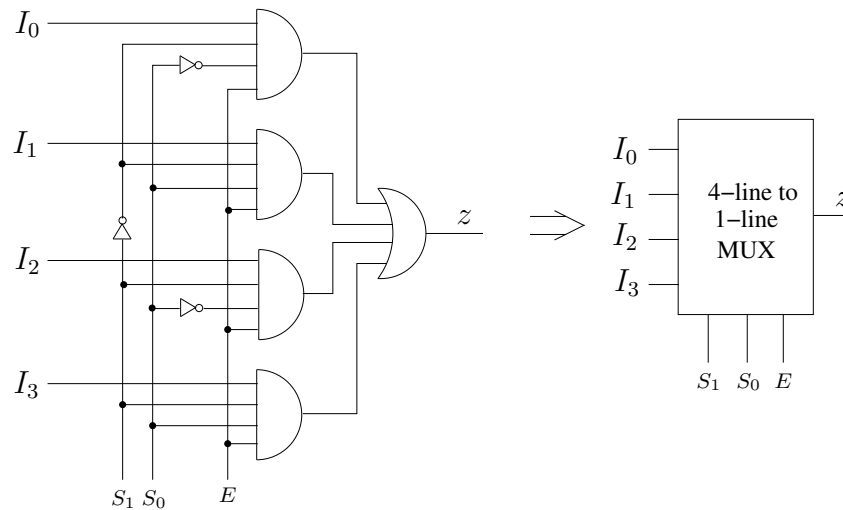


Figure 6.24: 4-line to 1-line multiplexer with ENABLE input

6.5.5 Examples of IC multiplexers

An example of an IC multiplexer with an ENABLE input is the 8-input 74151 multiplexer shown in Figure 6.25.

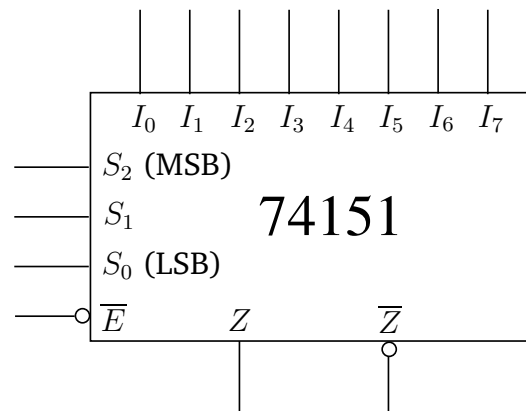


Figure 6.25: The 74151 8-input multiplexer

The truth-table for the 74151 multiplexer is given below:

\overline{E}	S_2	S_1	S_0	Z	\overline{Z}
1	X	X	X	0	1
0	0	0	0	I_0	\overline{I}_0
0	0	0	1	I_1	\overline{I}_1
0	0	1	0	I_2	\overline{I}_2
0	0	1	1	I_3	\overline{I}_3
0	1	0	0	I_4	\overline{I}_4
0	1	0	1	I_5	\overline{I}_5
0	1	1	0	I_6	\overline{I}_6
0	1	1	1	I_7	\overline{I}_7

Another example is the 74157 quad 2-input multiplexer IC (has four 2-input multiplexers). The circuit diagram for this multiplexer is shown in Figure 6.26.

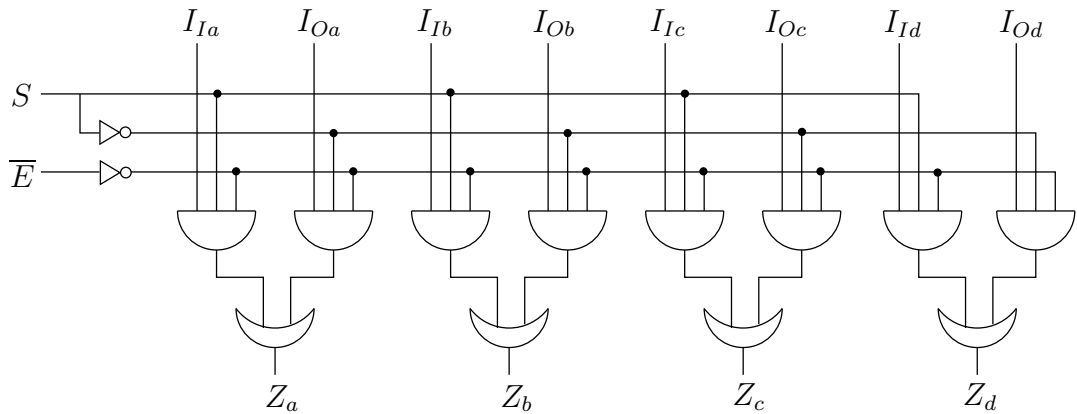


Figure 6.26: Circuit diagram for the 74157 Quad 2-input multiplexer

The symbol for the 74157 multiplexer is shown in Figure 6.27.

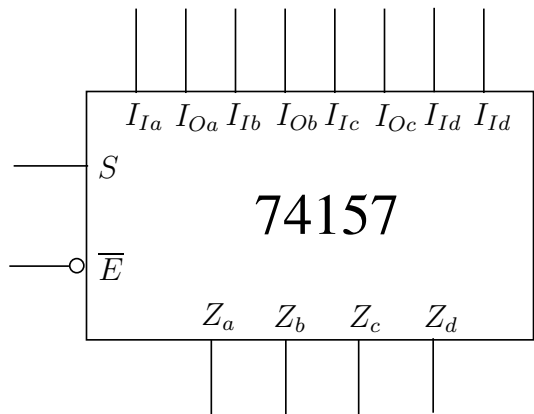


Figure 6.27: The 74157 Quad 2-input multiplexer

The truth-table for the 74157 is given below:

\overline{E}	S	Z_a	Z_b	Z_c	Z_d
1	X	0	0	0	0
0	0	I_{Oa}	I_{Ob}	I_{Oc}	I_{Od}
0	1	I_{Ia}	I_{Ib}	I_{Ic}	I_{Id}

6.5.6 The ENABLE input(s) in multiplexers

The ENABLE input(s) in multiplexers make it possible to connect two or more multiplexer ICs to form a multiplexer with a large number of inputs.

Exercise

An application requires a 16-input, 1-output multiplexer but the only multiplexer IC available is the 8-input 74151. Using two 74151 multiplexers and a minimum of logic gates, show how the required multiplexer can be realized.

6.5.7 Multiplexer applications

Data Routing: Multiplexers are used to route data from one of several sources to one destination. In the example shown in Figure 6.28, there are two BCD counters and one seven-segment display and the 74157 multiplexer is used to route the data from the counters to the display depending on the SELECT input.

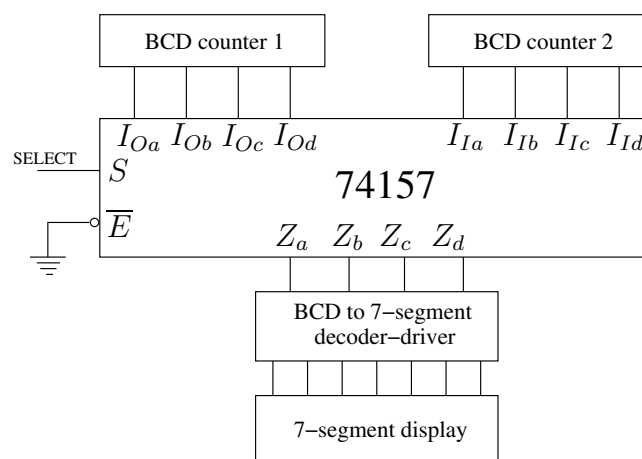


Figure 6.28: An example illustrating data routing

SELECT	Display
0	Contents of BCD counter 1 displayed
1	Contents of BCD counter 2 displayed

This kind of data routing is commonly used in digital watches/clocks to display the status of different counters using a single display.

Parallel to serial conversion: Figure 6.29 shows how a multiplexer can be used to convert data from parallel to serial form.

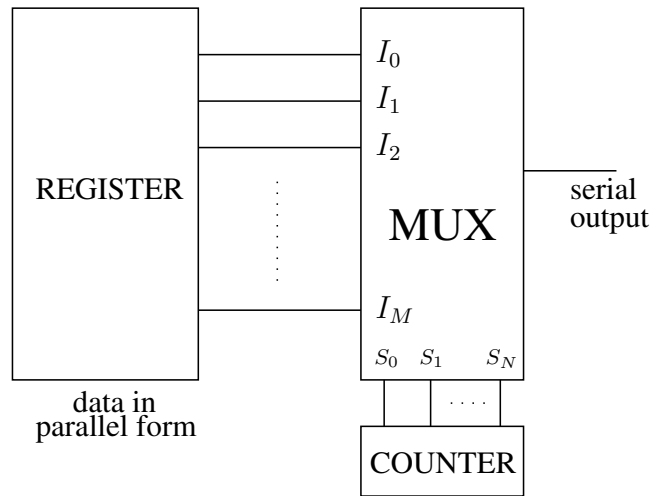


Figure 6.29: Parallel to serial conversion

In this application, the register where data is in parallel form is connected to the data inputs of a multiplexer. A counter is connected to the SELECT inputs. The contents of the register will appear at the multiplexer output in serial form.

Waveform generation: An arbitrary waveform can be generated by the circuit shown in Figure 6.29 by setting the desired bit pattern in the register.

Implementation of logic functions: Multiplexers can be used to implement logic functions directly from truth-tables without simplification.

Type 0 MUX design

The SELECT inputs are used as the logic variables and each data input is connected permanently HIGH or LOW as necessary to satisfy the truth-table.

Example

Implement the Boolean function

$$f(C, B, A) = \sum m(1, 2, 7) \quad (6.5)$$

using a 74151 multiplexer.

Solution

The truth-table corresponding to function (6.5) is as follows:

<i>C</i>	<i>B</i>	<i>A</i>	<i>f</i>
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Figure 6.30 shows the circuit.

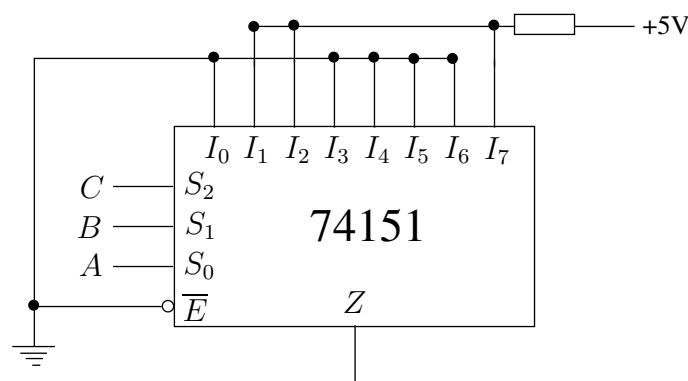


Figure 6.30:

Type 1 MUX design

This type of design is carried out when the number of logic variables exceeds the number of SELECT inputs by one. As an example, recall the Boolean function (6.1) given in Section 6.2, and reproduced here for convenience:

$$f(A, B, C, D) = \sum m(4, 5, 7, 9, 11, 15). \quad (6.6)$$

This function has 4 logic variables *A*, *B*, *C* and *D*, and to implement this function as described in the previous section, a multiplexer with 16 data inputs and 4 SELECT inputs is required. To implement this function using an 8-input multiplexer (which has 3 SELECT inputs), type 1 MUX design is applied. The least significant input bit is partitioned off in the truth-table as shown below. A new column is then created where the output is expressed as 0, 1, and the LSB.

Input Variables				Output	
<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>f</i>	<i>f</i>
0	0	0	0	0	0
0	0	0	1	0	
0	0	1	0	0	0
0	0	1	1	0	
0	1	0	0	1	1
0	1	0	1	1	
0	1	1	0	0	D
0	1	1	1	1	
1	0	0	0	0	D
1	0	0	1	1	
1	0	1	0	0	D
1	0	1	1	1	
1	1	0	0	0	0
1	1	0	1	0	
1	1	1	0	0	D
1	1	1	1	1	

The implementation of the Boolean function using a 74151 multiplexer is shown in Figure 6.31.

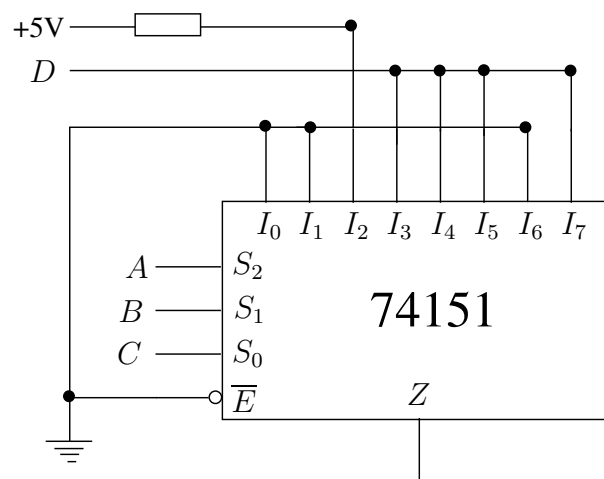


Figure 6.31:

Exercise

Implement the Boolean function

$$f(A, B, C, D) = \sum m(4, 5, 6, 7, 10, 14)$$

using a single 74151 MUX and a minimum of logic gates.

6.6 Demultiplexers

A demultiplexer takes one input data source and selectively distributes it to 1 of N output channels. For that reason, demultiplexers are sometimes referred to as data distributors. A demultiplexer is functionally the opposite of a multiplexer.

The circuit diagram shown in Figure 6.32 is an example of a demultiplexer.

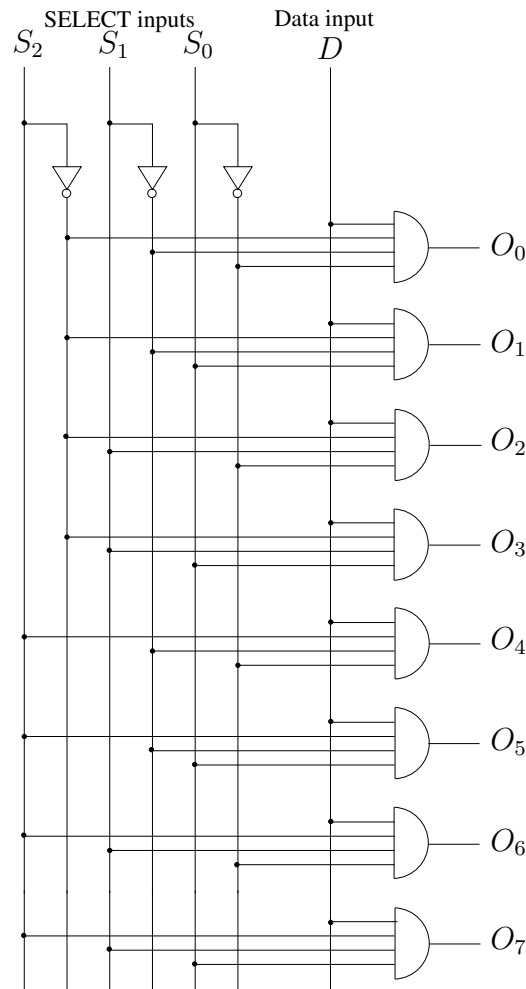


Figure 6.32: An example of a demultiplexer

The truth-table for the demultiplexer shown in Figure 6.32 is as follows:

S_2	S_1	S_0	O_0	O_1	O_2	O_3	O_4	O_5	O_6	O_7
0	0	0	D	0	0	0	0	0	0	0
0	0	1	0	D	0	0	0	0	0	0
0	1	0	0	0	D	0	0	0	0	0
0	1	1	0	0	0	D	0	0	0	0
1	0	0	0	0	0	0	D	0	0	0
1	0	1	0	0	0	0	0	D	0	0
1	1	0	0	0	0	0	0	0	D	0
1	1	1	0	0	0	0	0	0	0	D

The truth-table above is similar to that of a 3-line-to-8-line decoder. A closer look at

the circuit of Figure 6.32 shows that the circuit is similar to that of a 3-line-to-8-line decoder where the data inputs of the decoder are used the SELECT inputs and the ENABLE input used as the data input. As such, it is possible to use a decoder as a demultiplexer: the data inputs of the decoder are used as the SELECT inputs and the ENABLE input serves as the data input.

In general:

- ◇ If the decoder has active-HIGH outputs, one of the active-HIGH ENABLE inputs should be used as the data input.
- ◇ If the decoder has active-LOW outputs, one of the active-LOW ENABLE inputs should be used as the data input.

Exercise

Verify the validity of the above statements.

Exercise

Show how the 74138 decoder can be used as a demultiplexer.

6.7 Adders and Subtractors

6.7.1 The half adder

Given two bits X and Y , the truth table below shows the arithmetic sum S and the carry C resulting from the addition of the two bits.

X	Y	sum S	carry C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

From the truth-table above,

$$S = \bar{X}Y + X\bar{Y} = X \oplus Y$$

and

$$C = XY.$$

The Boolean function given in the truth-table above is implemented as shown in Figure 6.33. This circuit provides the sum and carry for two bits but does not take into consideration the carry from a lower significant bit position. Such a circuit is known as a *half-adder* (HA).

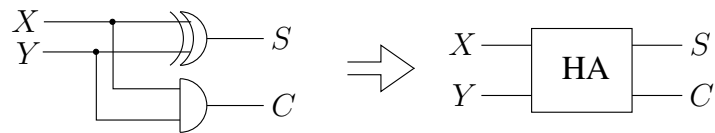


Figure 6.33: A half adder

6.7.2 The full adder

The full adder takes in bits X and Y and a carry from the lower significant bit position C_{in} , and gives the arithmetic sum S of the three bits, and a carry C_{out} to the next higher bit position. The truth-table for a full adder (FA) is given below:

X	Y	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$S = \bar{X}\bar{Y}C_{in} + \bar{X}Y\bar{C}_{in} + XYC_{in} + X\bar{Y}\bar{C}_{in} = X \oplus Y \oplus C_{in}$$

and

$$C_{out} = XY + YC_{in} + XC_{in}.$$

Figure 6.34 shows the schematic diagram for a full adder.

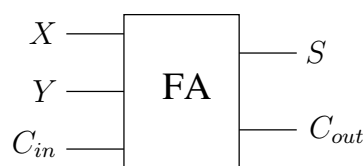


Figure 6.34: A full adder

6.7.3 The half subtractor

Given two bits X and Y , the truth table below shows the arithmetic difference D and the borrow B resulting from the subtraction of Y from X ($X - Y$).

X	Y	difference D	borrow B
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

From the truth-table above,

$$D = \bar{X}Y + X\bar{Y} = X \oplus Y$$

and

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

The function given in the truth-table above is implemented as shown in Figure 6.35.

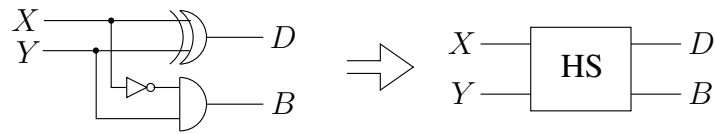


Figure 6.35: A half subtractor

This circuit provides the difference and borrow for two bits but does not take into consideration the borrow from a lower significant bit position. Such a circuit is known as a *half subtractor* (HS).

6.7.4 The full subtractor

The full subtractor carries out the arithmetic operation $(X - Y - B_{in})$, where B_{in} is a borrow from a lower significant bit position, and gives the difference D and the borrow B_{out} . The truth-table for a full subtractor (FS) is given below:

X	Y	B_{in}	D	B_{out}
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

$$D = \bar{X}\bar{Y}B_{in} + \bar{X}Y\bar{B}_{in} + XYB_{in} + X\bar{Y}\bar{B}_{in} = X \oplus Y \oplus B_{in}$$

and

$$B_{out} = \bar{X}Y + \bar{X}B_{in} + YB_{in}.$$

Figure 6.36 shows the schematic diagram for a full subtractor.

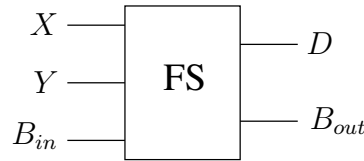


Figure 6.36: A full subtractor

6.7.5 Serial and parallel addition

In general, two binary numbers can be added in serial form or in parallel form. Serial addition uses only one full adder and a storage device to hold the generated output carry. Starting with the least significant bit position, a pair of bits in the binary numbers $\mathbf{A} = A_N A_{N-1} A_{N-2} \dots A_1 A_0$ and $\mathbf{B} = B_N B_{N-1} B_{N-2} \dots B_1 B_0$ are transferred to the full adder, one bit position at a time, until the sum of the two numbers \mathbf{A} and \mathbf{B} is generated.

In contrast, parallel addition produces the arithmetic sum of two binary numbers in parallel. An N -bit parallel adder requires N full adders.

In general, serial addition is slower than parallel addition, and the design of serial adders is more complicated than the design of parallel adders, hence parallel adders are much more widely used.

Figure 6.37 shows a 4-bit parallel full adder constructed by cascading 4 full adders.

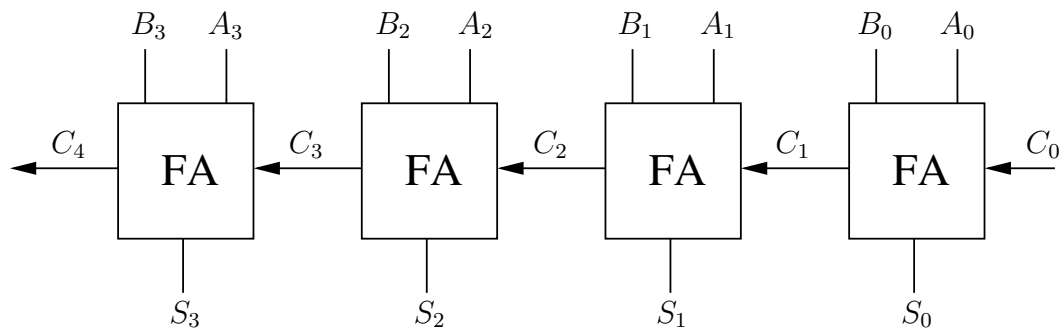


Figure 6.37: A 4-bit parallel full adder

The 4-bit parallel adder takes in two 4-bit numbers $A_3 A_2 A_1 A_0$ and $B_3 B_2 B_1 B_0$ and a carry bit C_0 from the lower bit position, and generates a sum $S_3 S_2 S_1 S_0$ and a carry out bit C_4 to the next higher bit position.

6.7.6 Examples of IC adders

The 7483 is an MSI 4-bit full adder and its schematic diagram is shown in Figure 6.38. It consists of four full adders connected as shown in Figure 6.37.

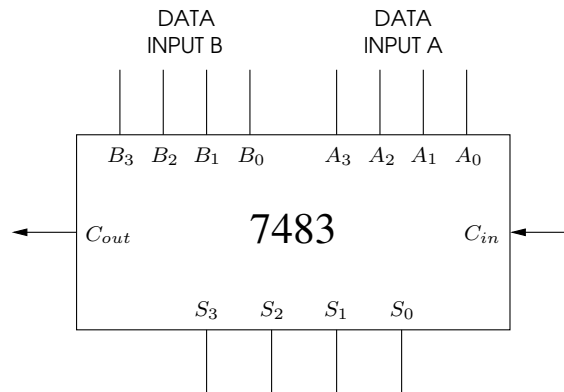


Figure 6.38: The 7483 4-bit full adder

Several 7483 adders can be cascaded using the C_{in} and C_{out} terminals to form larger adder units.

Another example of an MSI adder is the 74283 4-bit full adder IC.

6.7.7 The 2s complement adder/subtractor

Exercise

Recall that in 2s complement binary, subtraction can be performed just like addition, which makes it possible to use the same circuit for addition and subtraction. Figure 6.39 shows circuit which can be used for 2s complement addition and subtraction. Explain the concept behind this 7483-IC-based circuit.

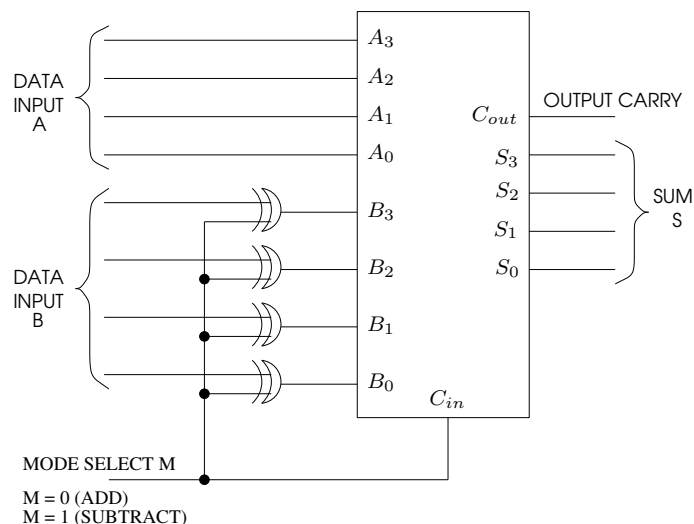


Figure 6.39: A 4-bit 2s complement adder/subtractor

6.7.8 Carry propagation

For parallel adders such as the one shown in Figure 6.37 (or the 7483), the carry propagation produces a propagation delay, i.e. the sum from the most significant bit cannot be decided until the carries from all the other bit positions have been taken into account. The propagation delay increases with the number of bits. This propagation delay limits the speed with which two numbers can be added. This problem is solved by use of a *look-ahead carry generator*, a device which generates all carry outputs simultaneously. The 74283 4-bit full adder IC has a built in look-ahead carry generator, so the 74283 is sometimes referred to as a fast adder. Look-ahead carry generators are also available in IC form, e.g. 74182.

Exercise Explain the principle of operation of the look-ahead carry generator.

Chapter 7

SEMICONDUCTOR MEMORY DEVICES

7.1 Introduction

Memory devices are used in digital systems to store digital information (1's and 0's). The ability of digital systems to store large quantities of information is one major advantage over analog systems. It makes digital systems very versatile and adaptable to many situations e.g. in a digital computer, the memory stores instructions to tell the computer what to do so that the computer can carry out a large variety of tasks with a minimum amount of human intervention.

The digital information stored is usually:

- data - data to be operated on, intermediate and final results.
- instructions - An instruction tells the digital system (computer) to perform a specific operation, such as adding two numbers together, comparing one number to another number, or moving a number to a specific memory location. (a series of instructions is usually referred to as a program)

As an example, in an expression like $3 + 4$, 3 and 4 is data while + is the instruction. (The data and instruction in this example would have to be binary-coded first before being stored in a memory system).

There are many different types of semiconductor memory devices and they are suited for different applications.

7.2 Memory Terminology

Memory Cell A device or an electrical circuit used to store a single bit (0 or 1) e.g. a flip-flop. A capacitor can also be used to store 1 bit of information (presence

of charge representing one logic level, absence of charge signifying the other logic level).

Memory Word A group of 1's and 0's which may represent a number, an instruction, one or more alphanumeric characters, or any other binary coded information.

Byte Special term for an 8-bit word.

capacity A way of specifying how many bits can be stored in a particular memory device or a complete memory system. E.g. suppose a certain memory device can store 4096 20-bit words.

$$\text{Capacity} = 4096 \times 20 (= 81920) \text{ bits}$$

The first number represents the *number of words* which can be stored in a particular memory device, while the second number represents *number of bits per word*.

In memory devices, $2^{10} = 1024 = 1K$ (1 Kilo), so we can write that the capacity of the memory device above as $4K \times 20$. Further, $2^{20} = 1M$ (1 Mega), $2^{30} = 1G$ (1 Giga), $2^{40} = 1T$ (1 Tera) e.t.c.)

Address A number that identifies the location of a word in memory. Each word stored in a memory device has a unique address.

Read Operation Operation whereby a binary word is stored in a specific memory is sensed and then transferred to another device. Also known as the **fetch** operation.

Write Operation The operation whereby a new word is placed into a particular memory location. Also known as the **store** operation. Whenever a new word is written into a particular memory location, it replaces the word previously stored there.

Access Time Time required to perform a read operation. It is a measure of speed for memory devices - the higher the access time, the slower the memory and vice-versa. A more specific definition of access time will be given when we cover Read Cycle Timing.

Volatile Memory A type of memory that requires the application of electrical power in order to store information. If electrical power is removed, all the information stored in the memory will be lost. An example of a volatile memory device is a flip-flop: if the power supply to a flip-flop is interrupted, the data stored will be lost.

Random Access Memory (RAM) A type of memory where the access time is the same for all locations i.e. access time does not depend on the address of the word. Example: Semiconductor memories.

Sequential Address Memory A type of memory where the access time varies with the location of the word. A particular word is found by sequencing through all address locations until the desired address is reached. Sequential Access Memories generally have larger access than RAMs. Example of a Sequential Access Memory device: Magnetic tape

Read Write Memory (RWM) Any type of memory device that can be written into or read from with equal ease.

Read Only Memory (ROM) A type of memory where the ratio of READ to WRITE operations is very high. Some ROMs can be written into only once after which information can only be read from them. Other types of ROM can be written into more than once but the WRITE operation is more complicated than the READ operation and is not performed very often.

Static Memory Devices Semiconductor memory devices in which data will remain permanently stored as long as power is supplied, without the need for periodically rewriting the data into memory e.g. flip-flops.

Dynamic Memory Devices Semiconductor memory devices in which the stored data will not remain permanently stored, even with power applied, unless data is periodically rewritten into memory. The rewriting of data is called a **refresh** operation.

7.3 Semiconductor Memory Technologies

On the basis of their manufacturing technologies, semiconductor memory devices are divided into two categories: bipolar memories (made using bipolar transistors) and MOS memories (made using MOSFETS). The two categories may be subdivided as shown on Figure 7.1.

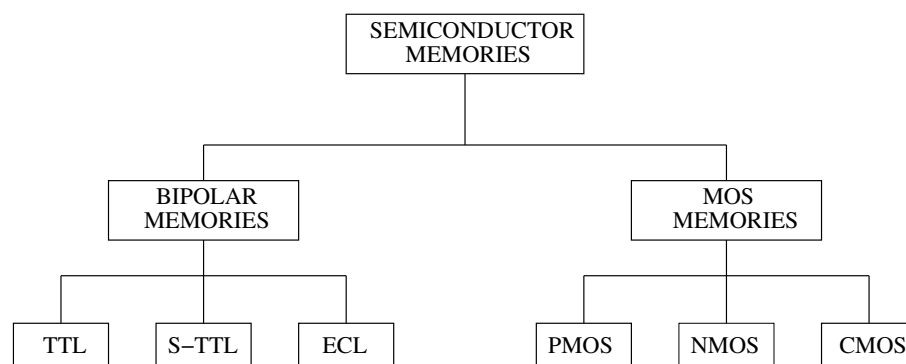


Figure 7.1: Semiconductor Memory Technologies

KEY:

TTL = Transistor Transistor Logic

S-TTL = Schottky TTL

ECL = Emitter Coupled Logic

PMOS = p-channel enhancement-mode MOSFET

NMOS = n-channel enhancement-mode MOSFET

CMOS = Complementary metal-oxide semiconductor

The memory devices manufactured using various technologies are usually compared according to the following criteria:

- Density - Capacity in bits per chip.
- Speed - usually specified using access time - the smaller the access time, the faster the memory device.
- Power requirements (μ W or mW per bit)- Power required to store data/operate.
- Cost of fabrication per bit.
- Noise immunity - Ability of a memory device to tolerate noise voltages at its inputs.

The technologies shown on Figure 7.1 compare as shown on the table below:

PARAMETER	Speed	Power Consumption	Capacity	Noise Immunity	Cost per bit
TTL/S-TTL	fast	high	low	low	high
ECL	fastest	highest	lowest	lowest	highest
NMOS/PMOS	medium	low	highest	high	lowest
CMOS	slowest	lowest	high	highest	low

- TTL memories are the most commonly used. They are suitable for applications that demand relatively high speed and medium capacity and where power consumption is not a critical factor
- ECL memories are used only where very high operating speed is required without regard to cost and power consumption
- CMOS memories are best suited for applications in high noise environments or where power consumption is extremely critical e.g. in battery operated systems

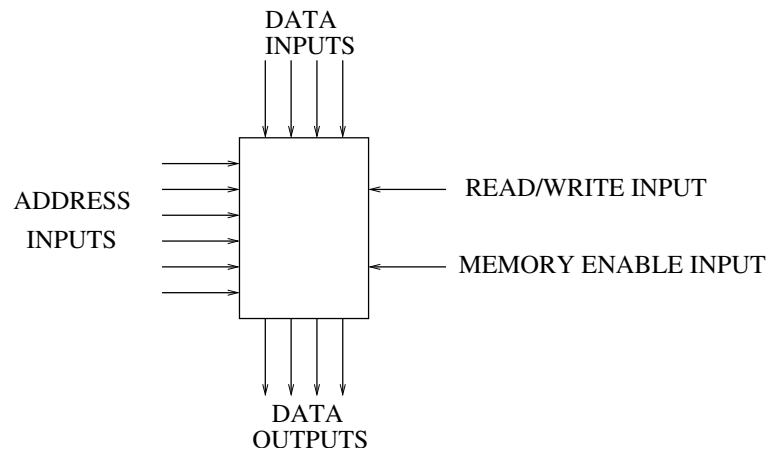


Figure 7.2: Semiconductor Memory – General case

7.4 Semiconductor Memory Operation

A semiconductor device can be represented as shown on Figure 7.2.

Address Inputs The address of the location being read from or being written into is entered through this inputs. N address inputs address 2^N memory locations. (For the memory device shown, the number of locations that can be addressed is $2^6 = 64$).

Data inputs Used to enter data during a WRITE (or STORE) operation

Data outputs Data appears on this lines during a READ operation. The data output lines are usually tristate i.e. have the states HIGH, LOW and High-Impedance (Hi-Z) states. This enables many memory devices to share a common data bus.

Memory Enable input (Usually abbreviated ME) Used to enable or disable the memory chip. If the ME input is not at the correct logic level, the memory chip will not respond to address and READ/WRITE inputs and the data output lines will be in the Hi-Z state.

Note that the memory enable input comes under several names e.g. CHIP ENABLE (CE) or CHIP SELECT (CS). Note also that some chips may have more than one enable input, and that some enable inputs may be active-LOW or active-HIGH.

READ/WRITE input (Usually abbreviated R/\overline{W}). Determines whether a READ or WRITE operation will take place.

$R/\overline{W} = 1 \longrightarrow$ READ OPERATION

$R/\overline{W} = 0 \longrightarrow$ WRITE OPERATION

Note that some of the older semiconductor devices had two separate inputs for READ and WRITE operations.

NOTE: Modern semiconductor memories use the same set of data pins for writing and reading the device. This is done to save the number of pins in the IC package. When $R/\overline{W} = 1$ (READ OPERATION), the data pins act as outputs, while $R/\overline{W} = 0$ (WRITE OPERATION) sets the pins to act as inputs.

Generally speaking, semiconductor memory devices operate as follows:

WRITE operation:

- i) Using the address input pins, apply the address of the location to be written into.
- ii) Apply the data to be written on the data input lines
- iii) Enable the memory chip by setting the Memory Enable Input(s) to the appropriate logic levels.
- iv) Set the R/\overline{W} input to LOW for a WRITE operation.

READ operation

- i) Using the address input pins, apply the address of the location to be read from.
- ii) Enable the memory chip by setting the Memory Enable Input(s) to the appropriate logic levels.
- iii) Set the R/\overline{W} input to HIGH for a READ operation.
- iv) Data is read from the data output lines

7.5 Read-Only Memories (ROMs)

ROMs are used to store data that is not to change during the operation of a system. One use of such memories is in the start-up programs of digital computers, printers, electronic photocopiers etc. A ROM IC does not have a R/\overline{W} input since the ROM cannot be written into under normal operating conditions.

7.5.1 ROM Timing

A ROM READ cycle is shown on Figure 7.3.

- t_0 : New address is applied to the ROM. The ROM circuitry begins to decode the address inputs to select the register which is to send data to the outputs.

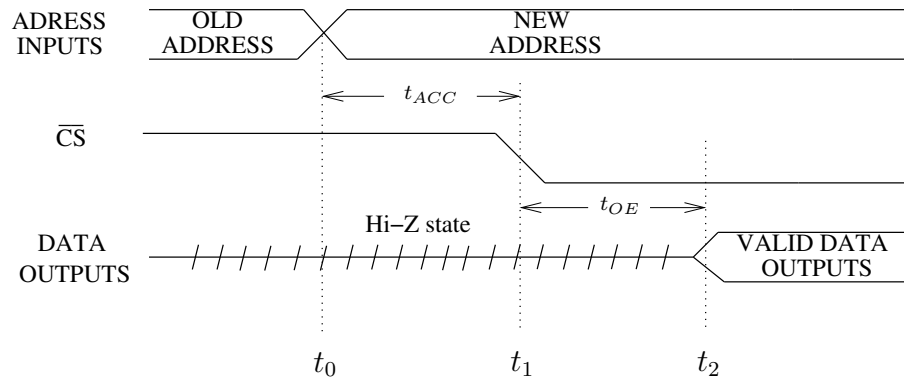


Figure 7.3: ROM READ operation timing diagram

- t_1 : The Chip select input \overline{CS} is activated to enable the output buffers.
- t_2 : Data outputs change from high impedance state to the valid data stored at the specified address.
- t_{OE} : Output Enable Time - Delay time between the chip being enabled and valid data appearing at the data outputs.
- t_{ACC} : Access Time - Time interval between the address being valid and the data outputs being valid.

Exercise:

A ROM has the following timing parameters: $t_{ACC} = 250ns$, $t_{OE} = 120ns$. The ROM has an active-LOW chip select input \overline{CS} . Assume that a new address is applied to the ROM x nanoseconds before \overline{CS} is driven LOW. Determine the minimum duration for which \overline{CS} must be kept LOW for a reliable READ operation if

- i) $x = 50ns$ ii) $x = 100ns$ iii) $x = 500ns$
 (200ns 150ns 120ns)

7.5.2 Mask-Programmed ROM (MROM)

This is programmed by the manufacturer according to the customer's specifications (usually in the form of a truth table). A photographic negative called a mask is used to control the electrical interconnections in the chip to produce 0's and 1's to satisfy the given truth table. Since the masks are expensive, this type of ROM is only economical if a large quantity of the same ROM is needed.

The main disadvantage of this ROM is that it cannot be reprogrammed. A new ROM would have to be programmed if the original program needs a modification. This type of ROM is generally used for data that does not change e.g. math tables, and character generator for CRT displays.

7.5.3 (User) Programmable ROM (PROM)

As the name suggests, this type of ROM is user programmable. The memory consists of an array of cells such as the one shown on Figure 7.4.

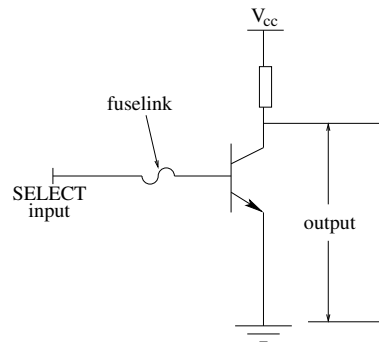


Figure 7.4: PROM cell

Each of the connections between the SELECT inputs and the transistor bases is made with a thin fuselink that comes intact from the manufacturer. By applying a certain voltage, the user can selectively blow any of the fuselinks to produce the desired truth table. Once a fuselink is broken, it cannot be reprogrammed. Special kits are available for programming PROMs.

To read a cell, the SELECT input is set HIGH. If the fuselink is intact, output voltage = 0, hence the cell is storing a 0. If the fuselink has been broken, the transistor will be in cut-off mode hence output voltage will be approximately equal to V_{CC} , meaning the cell is storing a 1.

7.5.4 Erasable Programmable ROM (EPROM)

An EPROM is user programmable, and it can also be erased and reprogrammed as often as desired. To see how an EPROM cell works, let us first look at the n-channel enhancement type MOSFET shown on Figure 7.5.

The drain is biased positive with respect to the source. No conduction occurs if no voltage is applied to the gate.

By applying a voltage such that the gate is positive with respect to the substrate, electrons in the substrate are attracted towards the gate and the substrate near the gate effectively changes from p-type to n-type. A low impedance channel is thus formed between the gate and the source and the drain and current flows from the drain to the source. (The minimum voltage which causes a drain-source current flow is known as the threshold voltage).

Figure 7.6 shows the EPROM cell.

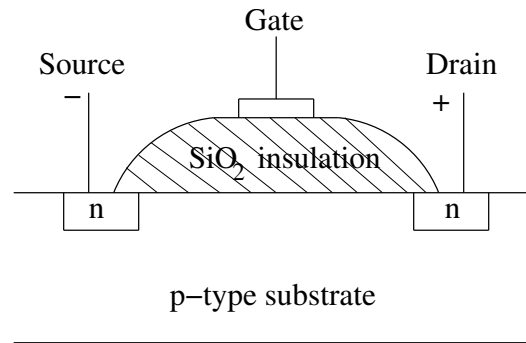


Figure 7.5: MOSFET

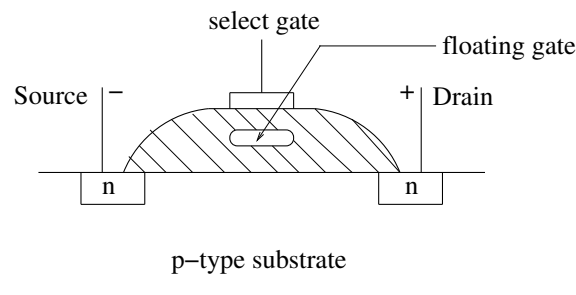


Figure 7.6: EPROM cell

The storage cell in an EPROM is a field-effect transistor with a silicon gate that has no electrical connections i.e. a floating gate. In the unprogrammed state, the memory cell has no charge stored in the floating gate and this state represents one logic state (usually logic 1 state). In the programmed state, the cell has charge stored in the floating gate and this represents the other logic state (logic 0).

Programming a '0' in the cell involves the application of about +20V between the drain and the source (the drain +) and about +25V on the select gate. The drain source potential causes electrons to flow from the source to the drain. Due to the high select gate voltage, some of the electrons crossing between the source and the drain acquire sufficient energy to pass through the silicon dioxide insulation to get to the floating gate. When the programming voltages are removed, the charge in the floating gate remains trapped since silicon dioxide is a good insulator. The charge can remain trapped for 10-20 years.

For an EPROM to be programmed, it has to be removed from the circuit board and placed in an EPROM programming kit. The chip's address and data pins are used to determine which memory cells will be affected by the programming voltages.

To read an EPROM cell, the select gate is set to a voltage slightly higher than the threshold voltage for the MOSFET. In the unprogrammed state (no charge stored in the floating gate), the cell will conduct like a MOSFET. When charge is stored in the floating gate, the threshold voltage for the MOSFET increases and the transistor will

not conduct when a voltage slightly greater than the threshold voltage is applied to the select gate.

Once a memory cell has been programmed, it can be erased only by exposing it to ultra-violet (UV) light. For this reason, EPROM ICs have a quartz window through which UV light can be applied. The UV light causes a flow of photocurrent from the floating gate to the p-substrate, thereby restoring the gate to its unprogrammed status. There is no way to expose just a single cell to the UV light without exposing all the cells. The UV light will therefore erase the entire memory. The erasure process requires 15-30 minutes of exposure to the UV rays. (For this reason, the EPROM is sometimes referred to as UV-EPROM).

EXAMPLE: the 2716 $2K \times 8$ EPROM

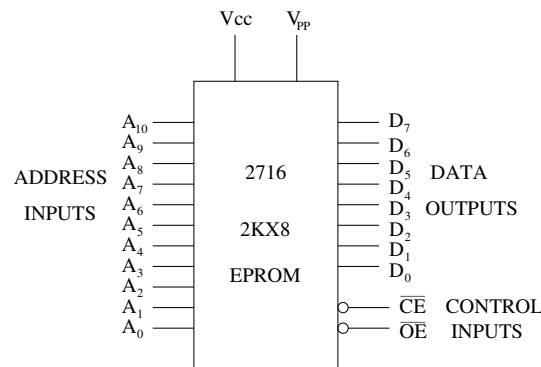


Figure 7.7: 2716 EPROM

During normal operation, both V_{CC} and V_{PP} are connected to a +5V supply. \overline{CE} and \overline{OE} have to be LOW in order for the internal circuitry to select the register which is being addressed and to route the data from that register to the outputs.

Programming the 2716 EPROM

- i) Connect both V_{CC} and \overline{OE} to a +5V supply, and V_{PP} to a +25V supply.
- ii) Apply the desired address to the address inputs.
- iii) Apply the desired 8-bit data word to the data input pins $D_0 - D_7$. Since the \overline{OE} input is HIGH, these data pins function as inputs.
- iv) Apply a 50ms LOW to HIGH pulse at \overline{CE} . At the termination of the pulse, the selected address location should be storing the applied data word.
- v) Disconnect the data inputs from the data output pins.
- vi) To verify if data has been stored correctly, the address location should be read. This is done by connecting V_{CC} and V_{PP} to +5V supply, applying the address of the location to be read, setting \overline{OE} and \overline{CE} LOW and then reading the data appearing at the data output pins.

Other examples of EPROM ICs are the $4K \times 8$ 2732, $8K \times 8$ 2764 and the $32K \times 8$ 27256.

7.5.5 Electrically Erasable Programmable ROM (EEPROM or E^2 PROM)

(Some books refer to this as Electrically Alterable Programmable ROM).

Some disadvantages of the UV-EPROM are:

- The device must be removed from the circuit to be programmed and to be erased.
- Erasure removes the entire memory contents
- The quartz window package is expensive.

These problems were solved by the development of the EEPROM in the early 1980s.

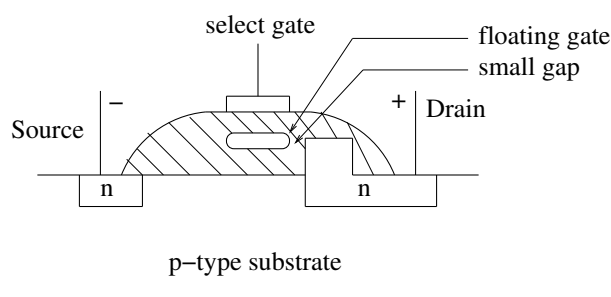


Figure 7.8: EEPROM cell

The EEPROM is a modified version of the UV-EPROM to allow electrons to tunnel through the silicon dioxide insulation in both directions depending on the applied voltages.

By applying about +21V between the cell's select gate and the drain, charge can be induced in the floating gate, where it will remain even when power is removed. Reversal of the same voltage will cause the removal of the trapped charges from the floating gate and this erases the cell. This process requires very low currents so the EPROM can be programmed in-circuit (without removing it from the circuit where it is being used).

Advantages of EEPROM over UV-EPROM include:

- With EEPROMs, it is possible to delete individual words in the memory array, which is not possible with UV-EPROMs.
- The EEPROM can be programmed in-circuit.

- The complete EEPROM can be erased in 10ms compared to about 30 minutes required for the UV-EPROM.

7.5.6 ROM Applications

Microcomputer Program Storage: Microcomputers use ROMs to store start-up programs. These microcomputer programs that are stored in ROM are called *firmware* because they are not subject to change. Products that include a microprocessor to control their operation use ROMs to store their control programs e.g. electronic cash registers, electronic photocopiers, printers, electronic games e.t.c.

Storage of data tables: ROMs are used to store tables of data that does not change e.g. trigonometric tables.

Character generators: Alphanumeric characters displayed on cathode ray tubes are made up of dots. Each character is made to fit into a pattern of dot positions, usually arranged as a 5×7 or a 7×9 matrix. To display a character, some dot positions are made bright and others dark. A character generator ROM stores the dot pattern for each character.

Implementing combinational logic functions: When used to implement combinational logic functions, the ROM acts like a combinational circuit that has a number inputs equal to its address inputs and the number of outputs equal to its data lines. The ROM is programmed so that each data output represents a specific function of the inputs.

As an example, suppose we would like to implement the following functions using a ROM:

$$f_1(A, B, C) = \bar{A}\bar{B}\bar{C} + AB$$

$$f_2(A, B, C) = AB\bar{C} + AC + B$$

$$f_3(A, B, C) = A\bar{B}\bar{C} + ABC + \bar{A}B$$

$$f_4(A, B, C) = C$$

First, these Boolean expressions have to be converted to the truth-table format, and the truth-table is given below:

A	B	C	f_1	f_2	f_3	f_4
0	0	0	1	0	0	0
0	0	1	0	0	0	1
0	1	0	0	1	1	0
0	1	1	0	1	1	1
1	0	0	0	0	1	0
1	0	1	0	1	0	1
1	1	0	1	1	0	0
1	1	1	1	1	1	1

There are three variables A , B and C and 4 outputs f_1 , f_2 , f_3 and f_4 . To implement the truth-table, we therefore need an 8×4 ROM IC (which has 3 address lines). The variables A , B and C are connected to the address lines. The address 000 is then programmed to store 1000, address 001 to store 0001 e.t.c. You can think of this process as the ROM being made to “memorize” the truth-table.

7.6 Programmable Logic Devices (PLDs)

A PLD is an integrated circuit (IC) with internal logic gates that are connected through electronic fuses. Programming the device involves blowing of the fuses along the paths that must be disconnected so as to obtain a particular function. Like a ROM, once a PLD has been programmed for a particular purpose, it cannot be erased and reprogrammed. An example of a PLD is a Programmable Logic Array (PLA).

A PLA is a logic circuit made up of AND and OR gates which can be interconnected to generate one or more outputs that are sum of product functions of several inputs. By selectively blowing the fusible links that interconnect the logic inputs to the AND gates, and the And gates to the OR gates, any desired sum-of-products output function can be generated. In block diagram form, a PLA looks as shown on Figure 7.9.

The symbols shown in Figure 7.10 are used in PLAs.

An example of a 3-input, 2-output PLA is shown in Figure 7.11.

Example: Use the PLA shown on Figure 7.11 to implement the truth-table shown below:

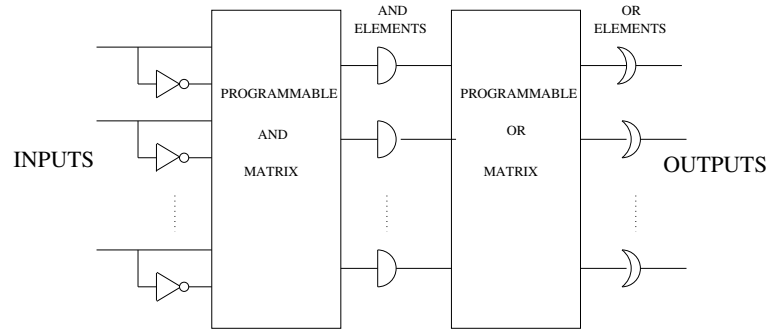


Figure 7.9: Block diagram representation of a PLA

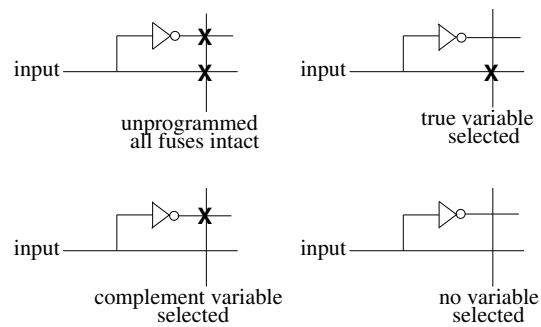


Figure 7.10: Symbols used in PLAs

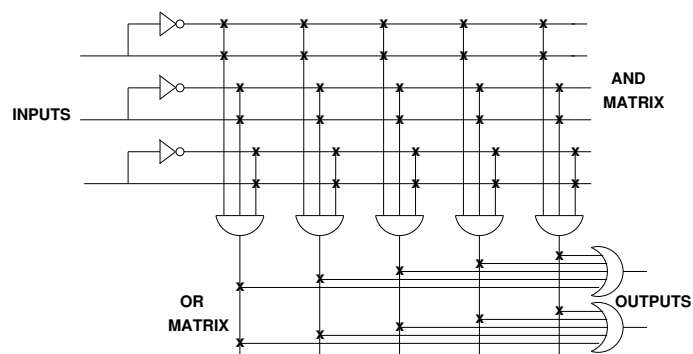


Figure 7.11: 3-input 2-output PLA

A	B	C	f_1	f_2
0	0	0	1	1
0	0	1	0	0
0	1	0	0	0
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	1	0
1	1	1	0	1

Solution: From the given truth-table, we can write that:

$$f_1 = \bar{A}\bar{B}\bar{C} + \bar{A}BC + AB\bar{C}$$

$$f_2 = \bar{B}\bar{C} + ABC$$

These functions can be implemented as shown in Figure 7.12.

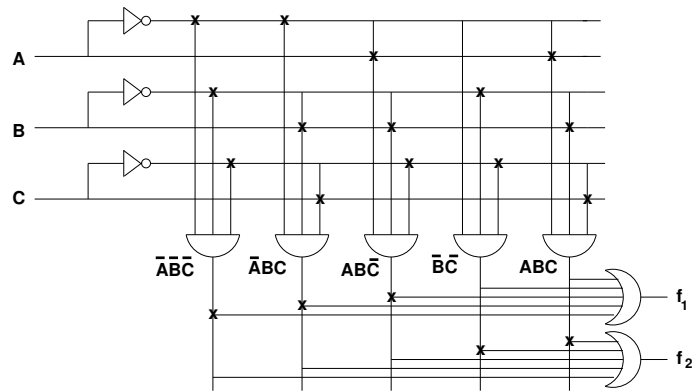


Figure 7.12: Example: using a PLA used to implement a combinational logic circuit

A shorthand way of drawing Figure 7.12 is shown in Figure 7.13.

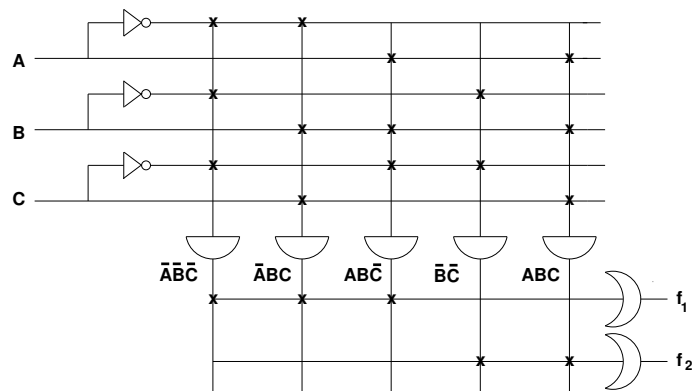


Figure 7.13: Shorthand notation of Figure 7.12

In the implementation of large combinational logic circuits, PLDs are preferred to SSI or MSI devices for the following reasons:

- PLDs use less circuit board area since one PLD package is equivalent to several packages of SSI/MSI devices.
- PLDs shorten design time since less packages are used.
- Design changes can be done by reprogramming new PLDs, which is less time consuming than redesigning a circuit board made up of SSI/MSI devices.
- Since PLD-based circuits use fewer ICs, there are fewer interconnections to be made and this makes PLD-based circuits more reliable than SSI/MSI circuits.

7.7 Semiconductor RAMs

This refers to random-access read-write memory devices. RAMs are used in microcomputers for the temporary storage of programs and data. RAMs are volatile and will lose all information stored if power is turned off.

There are two categories of RAM: static RAM (SRAM) and dynamic RAM (DRAM). STATIC RAM is made up of memory cells which are capable of retaining stored information indefinitely, as long as power supply is maintained. DYNAMIC RAM memory is only capable retaining the stored information for a few milliseconds (typically 2ms). The dynamic memory cell is a capacitor which stores charge, and this charge inevitably leaks with time. The DRAM must therefore be refreshed periodically by recharging the capacitors in order to retain the information.

7.7.1 Static RAM Architecture

This is illustrated using a 64×6 RAM shown on Figure 7.14.

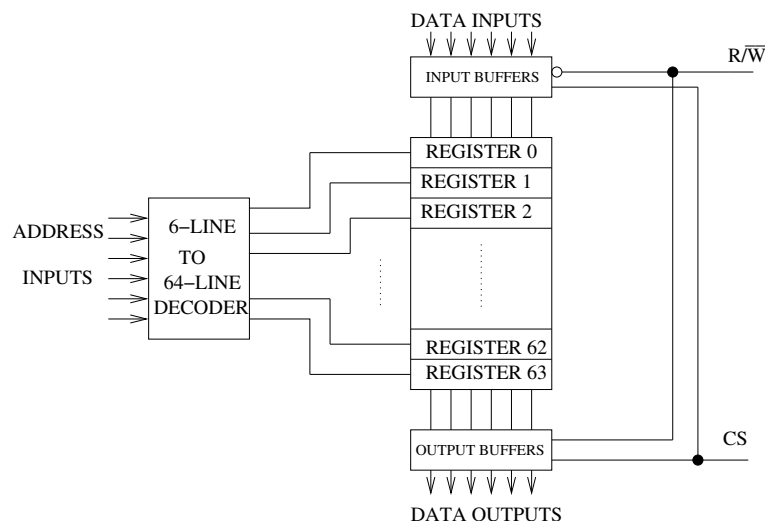


Figure 7.14: SRAM Architecture

READ OPERATION: The address code selects one register for READ or WRITE operation. For READ operation, the R/\overline{W} input and the CS input must be HIGH. (Chip Select input is active-HIGH in this example). This combination enables the output buffers so that the contents of the selected register will appear at the data outputs. Setting the R/\overline{W} signal HIGH disables the input buffers so that the data inputs do not affect the memory during a READ operation.

WRITE OPERATION $R/\overline{W} = 0$ and $CS = 1$. This combination enables the input buffers so that the 6-bit word applied to the data inputs will be loaded into the

selected register. Setting R/\overline{W} LOW disables the output buffers so that the data outputs are in their High-Impedance State during a write operation.

NOTE: Since the READ and WRITE operations are not carried out simultaneously, most RAM ICs use the same data pins for data input and output (such pins are known as data input/output lines, abbreviated data I/O lines). The R/\overline{W} input controls the functions of these pins i.e. $R/\overline{W} = 0$, data I/O lines act as inputs and for $R/\overline{W} = 1$, data I/O lines act as outputs. This helps to reduce the number of pins on an IC package.

For larger RAMS, the registers are arranged in a matrix similar to that of DRAM architecture (Figure 7.20).

7.7.2 Static RAM memory Cell

Consider the circuit shown on Figure 7.15 where Q_1 , Q_2 , Q_3 and Q_4 are field-effect transistors. (In the figure S = Source, D = Drain and G = Gate).

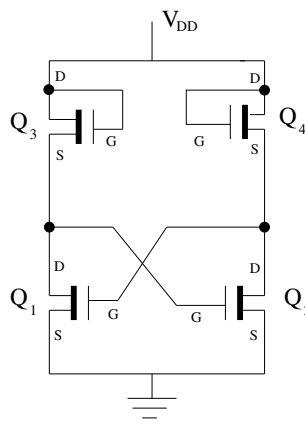


Figure 7.15: Basic SRAM cell

Q_3 and Q_4 are connected such that they act as high-value resistors (typically 10^{12} ohms - this type of connection is preferred to using resistors as it takes much less space than resistors of the same value, and the very high resistance reduces power consumption).

Transistors Q_1 and Q_2 are cross-coupled, and act like a flip-flop i.e. suppose Q_1 is fully conducting. Then its drain falls to nearly 0V and this causes Q_2 to become non-conducting and its drain rises to the supply voltage. This voltage is applied to the gate of Q_1 and since it is above the threshold voltage of Q_1 , Q_1 is maintained in the conducting state. The circuit can be maintained in one of two states: Q_1 fully conducting and Q_2 off, or Q_1 off and Q_2 fully conducting. The complete memory cell is shown on Figure 7.16.

Data is written into the cell by enabling Q_5 and Q_6 (by setting ROW SELECT line

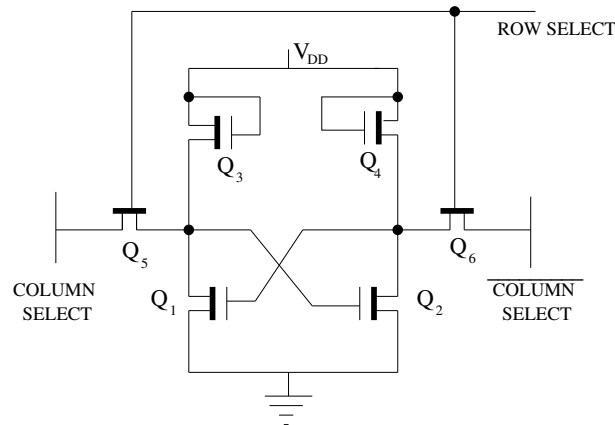


Figure 7.16: Complete SRAM cell

HIGH), applying the desired data to the *COLUMN SELECT* line and the complement of the data on the *COLUMN* SELECT line.

Data is read by enabling Q_5 and Q_6 and reading the *COLUMN SELECT* line.

7.7.3 Static RAM Timing

READ Cycle

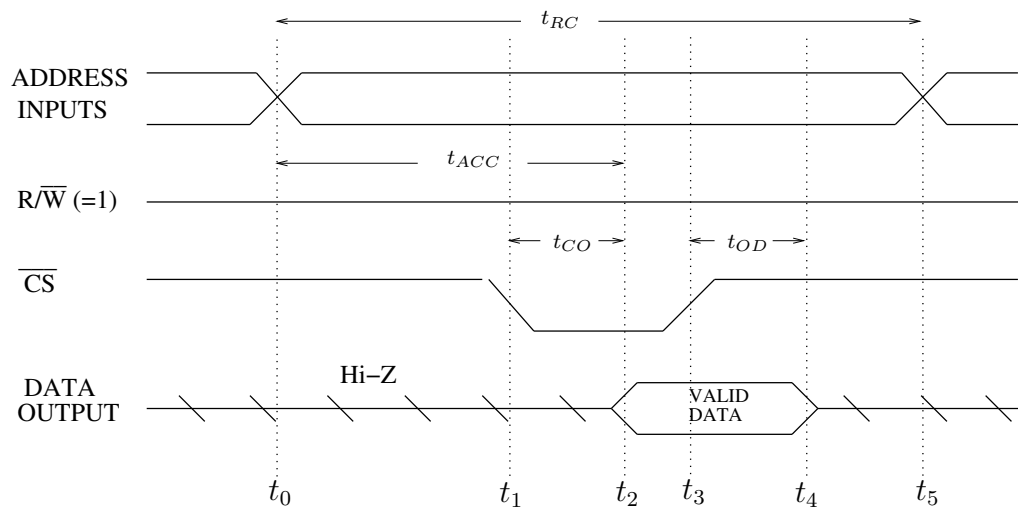


Figure 7.17: SRAM READ cycle

- t_0 : Address inputs change to a new address from which data is to be read. This is the beginning of a READ cycle.
- t_1 : Chip Select input activated
- t_2 : Data outputs change from High Impedance State to valid data outputs.

- t_3 : Chip Select input is de-activated.
- t_4 : Data outputs change from valid data outputs to the High Impedance State in response to the de-activation of the Chip Select signal. Any device that needs to read data from the memory device should do so between t_2 and t_4 .
- t_5 : End of READ cycle. Address inputs change to a different address for a another READ or WRITE cycle.
- t_{CO} : Minimum time taken for memory outputs to go from High Impedance state to valid data outputs after Chip Select input has been activated.
- t_{OD} : Minimum time taken for memory outputs to go from valid data outputs to High Impedance state after Chip Select input has been de-activated.
- t_{RC} : READ cycle time.

WRITE cycle

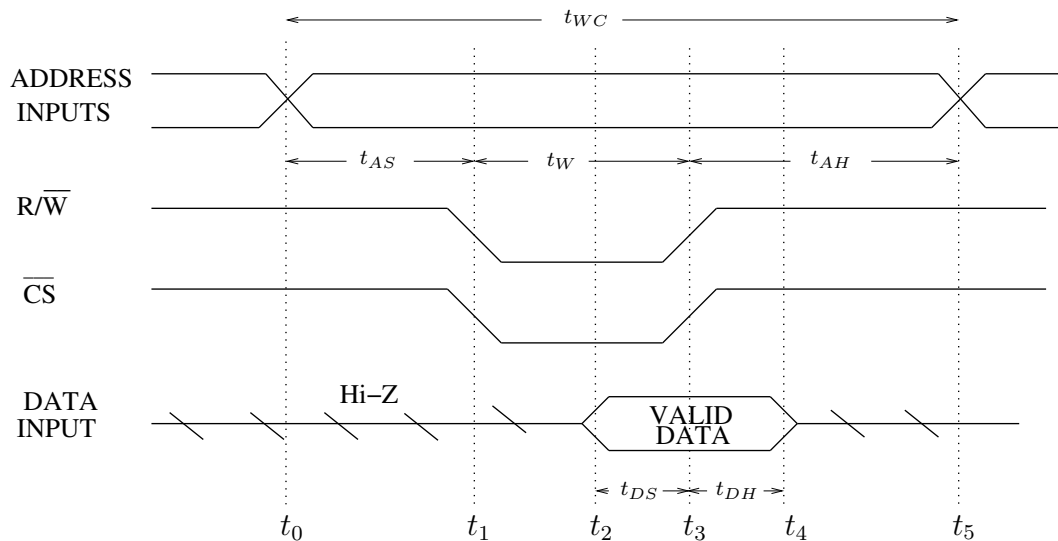


Figure 7.18: SRAM WRITE cycle

- t_0 : Address inputs change to a new address to which data is to be written. This is the beginning of a WRITE cycle.
- t_1 : Chip Select input is activated and at the same time, the R/\overline{W} input is set LOW for a WRITE operation.
- t_{AS} : Address set-up time - time taken by the RAM's address decoders to respond to the new address.
- t_2 : Data to be written into the addressed memory location is applied. The data has to be held stable for at least a time t_{DS} , data set-up time, before R/\overline{W}

and \overline{CS} are returned HIGH. The data also has to be held stable for at least a time t_{DH} , data hold time, after R/\overline{W} and \overline{CS} have returned HIGH. Similarly, the address inputs have to be maintained for a time t_{AH} , address hold time after R/\overline{W} and \overline{CS} have returned HIGH. If any of these set-up or hold-time requirements are not met, the WRITE operation will not occur reliably.

- t_{WC} : WRITE cycle time.

7.8 Dynamic RAM (DRAM)

7.8.1 DRAM cell

A DRAM cell is shown on Figure 7.19.

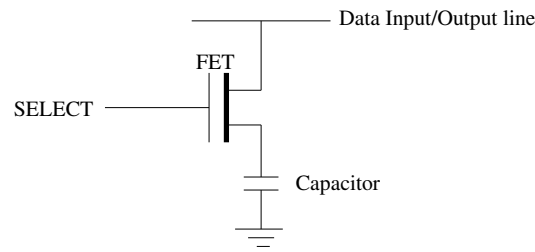


Figure 7.19: DRAM cell

The FET acts as a switch while the capacitor is the actual storage element.

WRITE operation: The SELECT line is held HIGH to turn on the transistor. The voltage applied to the data input/output line charges or discharges the capacitor to store either a 1 or a 0. The SELECT line is then made LOW, turning off the transistor and opening the path to the capacitor. In practice, the charge in the capacitor leaks off after about 2ms.

READ operation: The SELECT line is made HIGH, connecting the capacitor to the data input/output line. The charge stored in the capacitor determines the voltage that appears on the data input/output line.

Advantages of DRAMs over SRAMs include:

- High capacity due to its simple cell structure.
- Low cost per bit
- Power consumption is low because the DRAM cell only draws a significant amount of current when it is being charged.

Disadvantages of DRAM over SRAM include:

- Slower than SRAMs
- DRAMs require more external support circuitry than SRAMs because of the refreshing and address multiplexing operations that must take place.

7.8.2 DRAM structure

A DRAM is usually arranged as a matrix of single bit registers as shown on Figure 7.20.

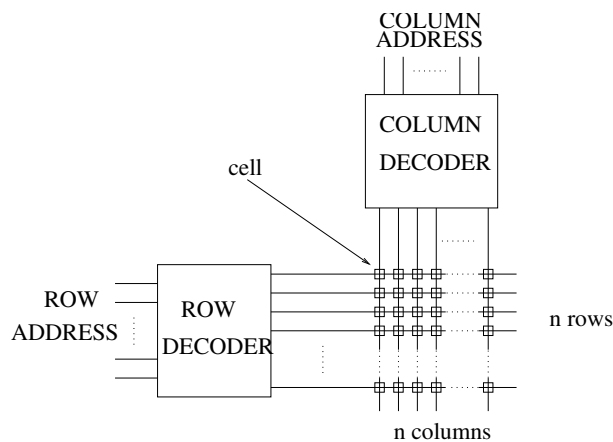


Figure 7.20: DRAM structure

In order to reduce the required number of address pins, a DRAMs memory address is split into two parts: a row address and a column address corresponding to the row and column of the required cells. The row address is first applied to the DRAM chip and then the column address is applied using the same address pins as the row address. This is known as *address multiplexing*. This concept is illustrated using a 64×1 DRAM on Figure 7.21.

In the case shown on Figure 7.21, we have saved three address pins but we have an additional two pins, \overline{CAS} and \overline{RAS} , so overall, we save only one pin by using address multiplexing. The saving is higher with larger DRAMs, e.g. using address multiplexing with a $256K \times 1$ DRAM, the saving is seven pins.

The timing diagram corresponding to Figure 7.21 is shown on Figure 7.22.

- t_{RS} : Row Address set-up time
- t_{CS} : Column Address set-up time

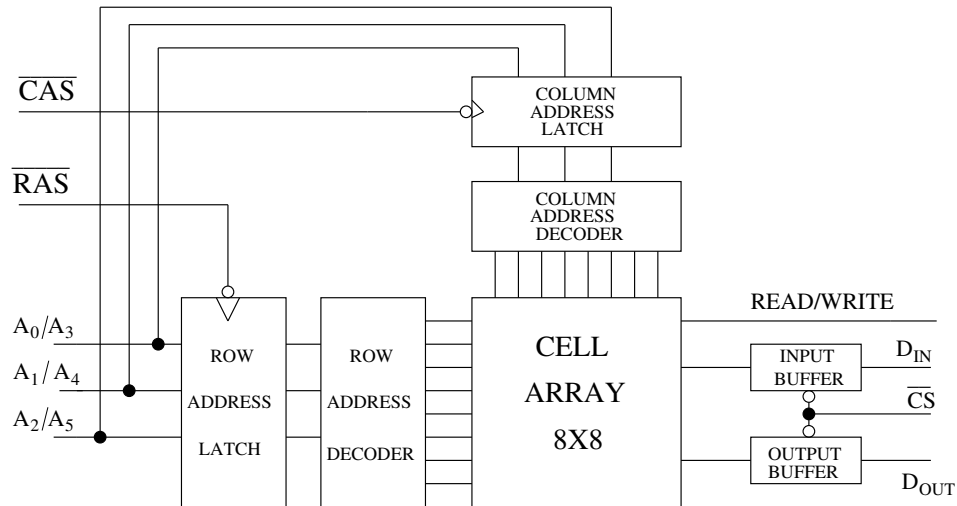


Figure 7.21: DRAM: address multiplexers

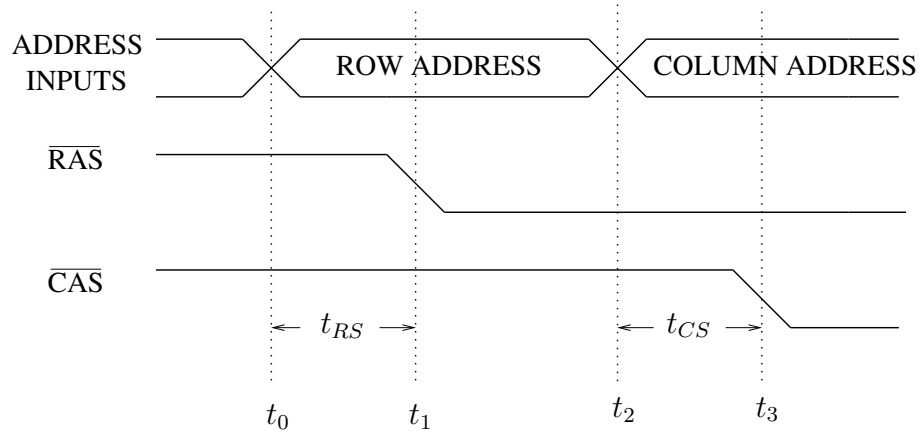


Figure 7.22: DRAM timing

- t_0 : Row address is applied to the address inputs. After allowing time for the address inputs to stabilize, the Row Address Strobe \overline{RAS} is driven LOW at t_1 . \overline{RAS} clocks the row address latch to store the row address.
- t_2 : Column address is applied to the address input pins.
- t_3 : Column Address Strobe, \overline{CAS} is driven LOW and the column address is loaded into the column address latch.

When both portions of the address are in their respective latches, the decoders will select the memory cell that is being accessed, and a READ or WRITE operation will take place just as in a static RAM.

7.8.3 DRAM refreshing

DRAMs are designed so that each time a READ or WRITE operation is performed on a memory cell, all the cells on the same row will be refreshed. However, it cannot be guaranteed that every row of a DRAM will be read from or written into every 2ms so the refresh operation has to be performed by other means.

Typical refresh circuitry contains a refresh counter that generates sequential row addresses for the refresh operation. The addresses from the refresh counter are used to access the various rows of the DRAM to perform a row refresh operation once in every 2ms.

7.9 Memory Expansion

Memory expansion refers to the combination of two or more memory ICs to:

- i) form longer words
- ii) store more words
- iii) both to form longer words and to store more words.

7.9.1 Expanding word size

This involves the combination of several ICs to form longer words.

Example 1

Suppose we need to store 16 8-bit words but we only have 16×4 RAM ICs as shown on Figure 7.23.

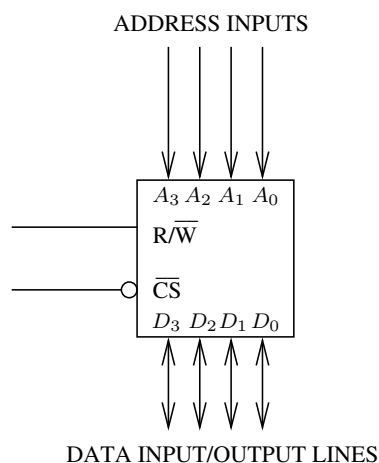


Figure 7.23: 16×4 RAM

The objective here is to create a 16×8 memory module using 16×4 ICs. Number of

ICs needed $= (16 \times 8) / (16 \times 4) = 2$. Two 16×4 ICs can be connected as shown on Figure 7.24 to form a 16×8 memory module.

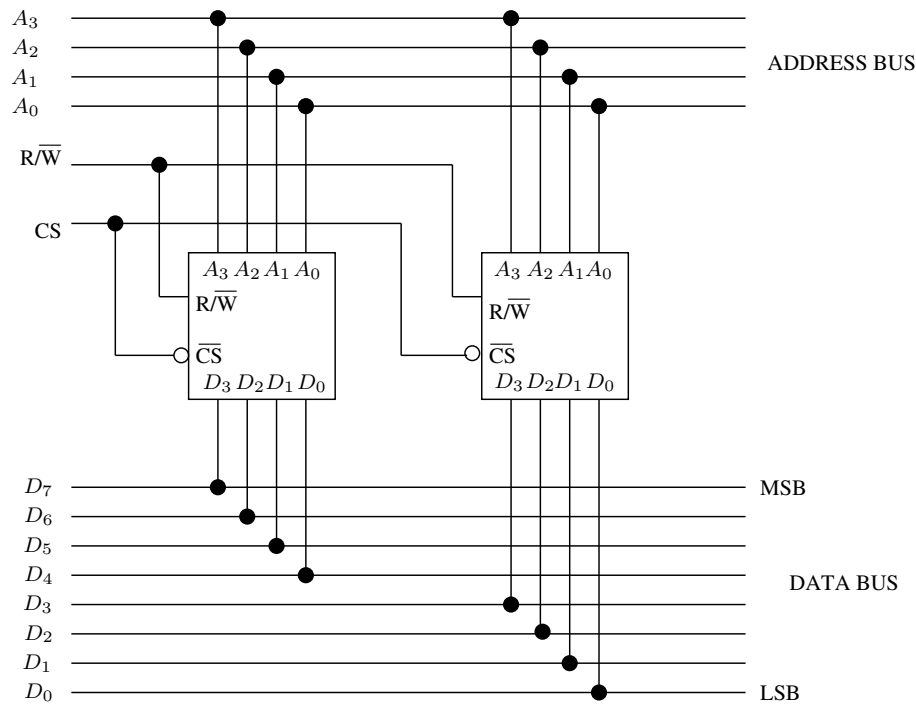


Figure 7.24: 16×8 RAM module

Any one of the 16 words is selected by applying the appropriate address code to the 4-line address bus. Since each address bus line is connected to the corresponding address input of each IC, the same address code is applied to both chips so that the same location in each IC is accessed at the same time. The combination of the two ICs acts as a single 16×8 memory chip. READ operation: $R/\overline{W} = \text{HIGH}$, $\overline{CS} = \text{LOW}$. WRITE operation: $R/\overline{W} = \text{LOW}$, $\overline{CS} = \text{LOW}$.

Example 2

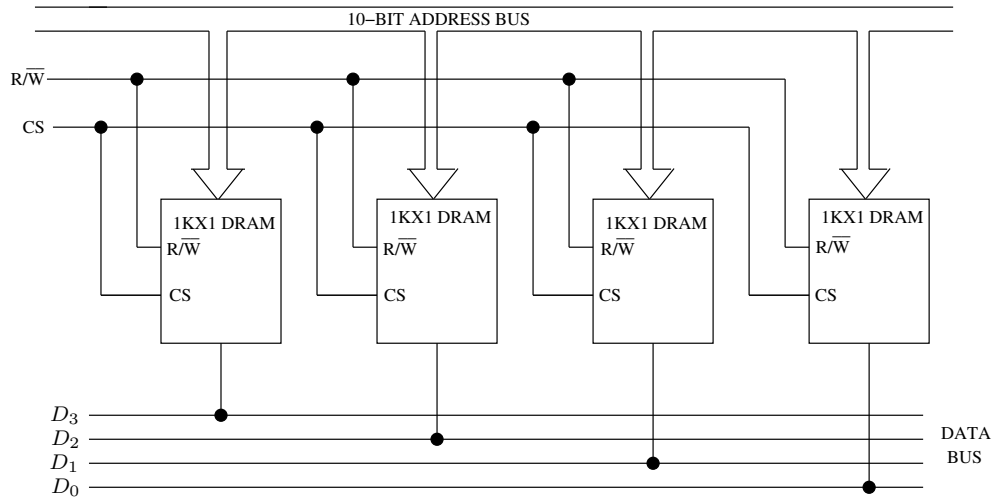
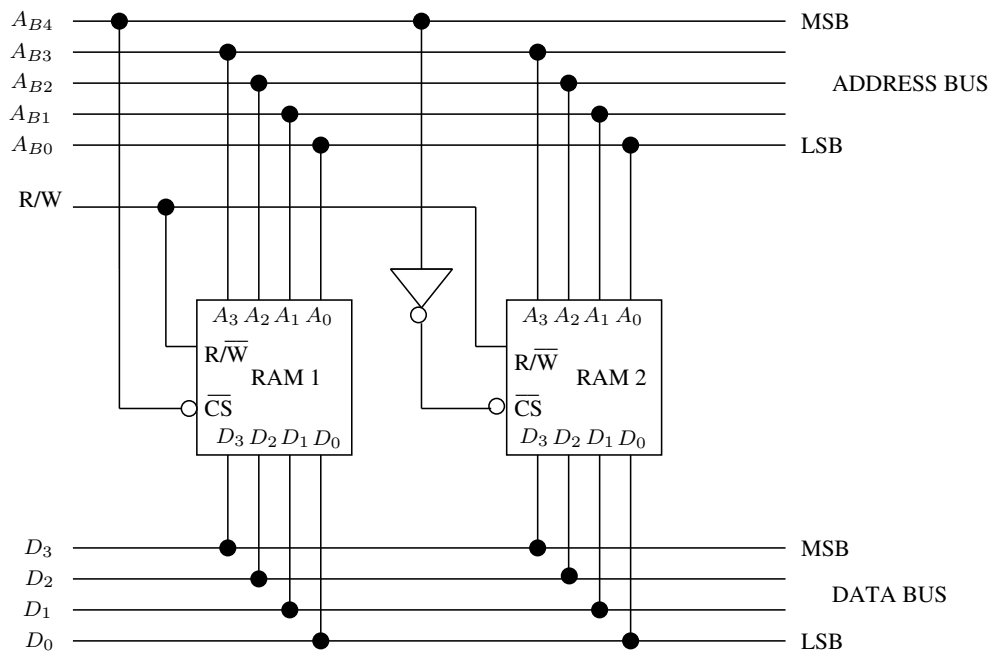
To construct a $1K \times 4$ DRAM module using $1K \times 1$ DRAM ICs. This is constructed as shown on Figure 7.25.

7.9.2 Expanding Capacity

This refers to creating a memory module that has a bigger number of words than the ICs used to make it.

Example 1

Suppose we need a memory that can store 32 4-bit words we only have 16×4 RAM ICs as shown on Figure 7.23. Number of ICs needed $= (32 \times 4) / (16 \times 4) = 2$. Two 16×4 ICs can be connected as shown on Figure 7.26 to form a 32×4 memory module.

Figure 7.25: $1K \times 4$ DRAM moduleFigure 7.26: 32×4 RAM module

Each RAM is used to hold 16 4-bit words. Since the capacity of the memory is 32×4 , 5 address lines are required.

When $A_{B4} = 0$, the \overline{CS} of RAM 1 enables this IC for a READ or WRITE operation. Then any address location in RAM 1 can be accessed by address bits $A_{B3}A_{B2}A_{B1}A_{B0}$. The range of addresses representing locations in RAM 1 are $A_{B4}A_{B3}A_{B2}A_{B1}A_{B0} = 00000$ to 01111 (00 - 0F Hex).

When $A_{B4} = 1$, RAM 2 is enabled. The range of addresses located in RAM 2 is $A_{B4}A_{B3}A_{B2}A_{B1}A_{B0} = 10000$ to 11111 (10 - 1F Hex).

In this example, only two ICs were used so it was convenient to use an inverter to

select between the two ICs. When more than two ICs are used, it is more convenient to use a decoder, as the next example illustrates.

Example 2 To construct a $1K \times 8$ ROM using 256×8 ROM ICs.

The 256×8 ROMs are connected as shown on Figure 7.27 to achieve this purpose.

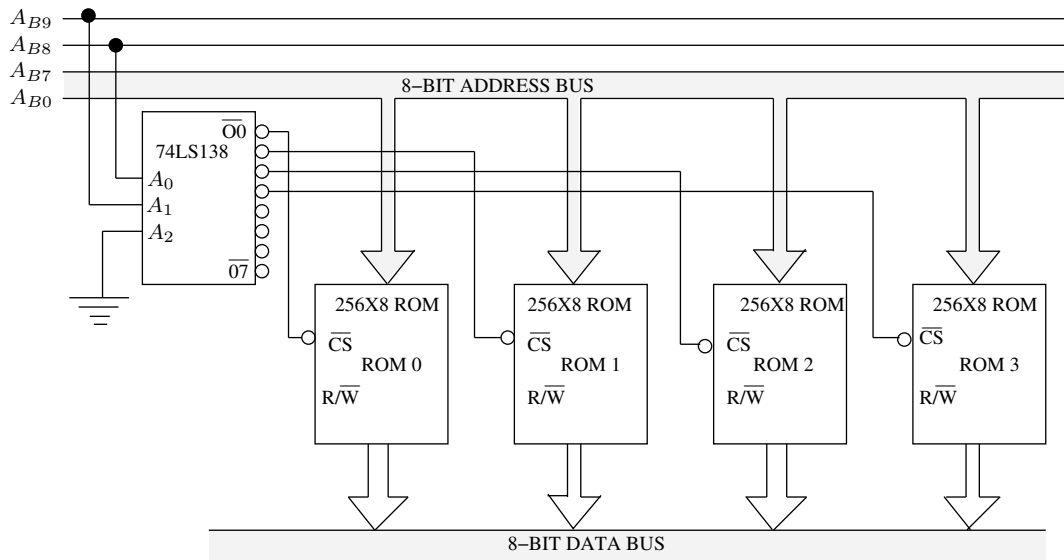


Figure 7.27: $1K \times 8$ ROM module

The range of addresses in each ROM is as follows:

ROM 0: 0000000000 to 0011111111 (000 to 0FF Hex)

ROM 1: 0100000000 to 0111111111 (100 to 1FF Hex)

ROM 2: 1000000000 to 1011111111 (200 to 2FF Hex)

ROM 3: 1100000000 to 1111111111 (300 to 3FF Hex)

NOTE: It is also possible to increase both word size and capacity.

Exercise

Using an IC such as the one shown on Figure 7.23, construct a 32×8 memory module.

Chapter 8

SEQUENTIAL NETWORKS AND TIMING CIRCUITS

8.1 Analysis of clocked sequential networks

8.1.1 Introduction

A sequential circuit can be represented by the block diagram on Figure 8.1.

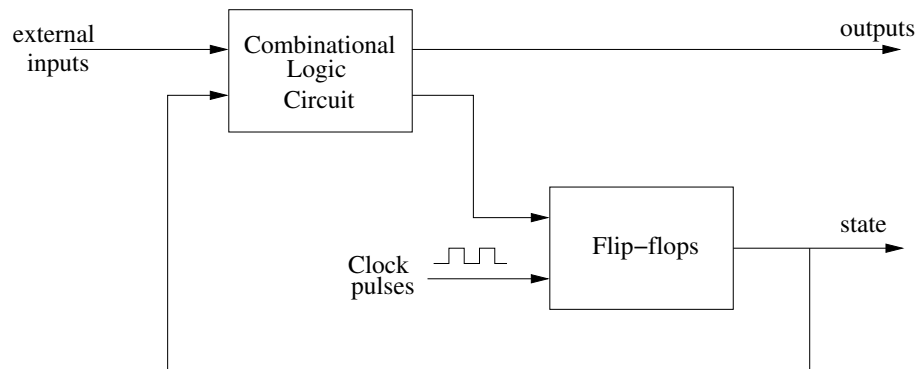


Figure 8.1: Block diagram of a clocked sequential logic circuit

As can be seen from the figure, the behaviour of the circuit is determined from the inputs, the outputs and the state of the flip-flops. The outputs and the next state of the flip-flops are functions of both the inputs and the current state of the flip-flops. The behaviour of a clocked sequential circuit can be described in the following ways:

- State equations, flip-flop input equations and output equations.
- State transition table
- State diagram

Flip-flop input equations: The flip-flop input equations are:

$$D_A = Q_A x + Q_B x = (Q_A + Q_B) x \quad (8.1)$$

and

$$D_B = \overline{Q}_A x. \quad (8.2)$$

To emphasize the fact that these are the inputs to the flip-flops at a time t , these equations are sometimes written as

$$D_A(t) = Q_A(t)x(t) + Q_B(t)x(t) = (Q_A(t) + Q_B(t))x(t) \quad (8.3)$$

and

$$D_B(t) = \overline{Q}_A(t)x(t). \quad (8.4)$$

State equations: Recall the truth-table for the D-flop given below:

D	Q_{n+1}
0	0
1	1

It can be seen from the table that

$$Q_{n+1} = D. \quad (8.5)$$

Using these relation, we can then write the state equations for the circuit shown in Figure 8.2 as

$$Q_{A+1} = Q_A x + Q_B x = (Q_A + Q_B) x \quad (8.6a)$$

$$Q_{B+1} = \overline{Q}_A x. \quad (8.6b)$$

Sometimes these equations are written as

$$Q_A(t+1) = Q_A(t)x(t) + Q_B(t)x(t) = (Q_A(t) + Q_B(t))x(t) \quad (8.7a)$$

$$Q_B(t+1) = \overline{Q}_A(t)x(t). \quad (8.7b)$$

The state equations give the next state of the circuit, i.e. the state of the circuit after the next clock edge – in this case the next positive going transition of the clock waveform.

Output equation: The output equation is

$$y = (Q_A + Q_B) \overline{x} \quad (8.8)$$

which can also be written as

$$y(t) = (Q_A(t) + Q_B(t)) \overline{x}(t). \quad (8.9)$$

The output equations are used to determine the current output of the circuit.

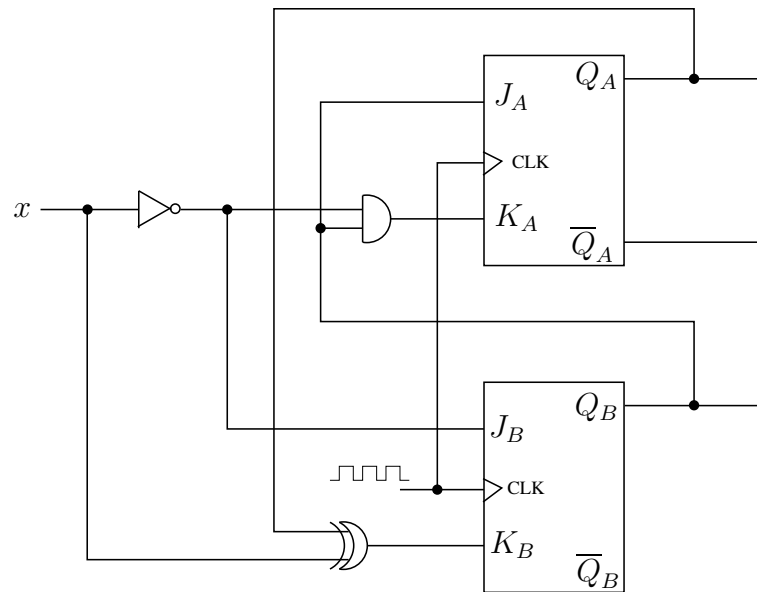


Figure 8.3: Example

Example

Consider the circuit shown in Figure 8.3.

The flip-flop input equations for this circuit are

$$J_A = Q_B \quad (8.10a)$$

$$K_A = Q_B \bar{x} \quad (8.10b)$$

$$J_B = \bar{x} \quad (8.10c)$$

$$K_B = Q_A \oplus x = Q_A \bar{x} + \bar{Q}_A x \quad (8.10d)$$

To write the state equations, consider the truth-table for a JK flip-flop below.

J	K	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	\bar{Q}_n

This truth-table can be expanded as shown below:

J	K	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Using algebraic means or a Karnaugh map, a simplified expression for Q_{n+1} is obtained as

$$Q_{n+1} = J\overline{Q}_n + \overline{K}Q_n \quad (8.11)$$

which is the state equation for the JK flip-flop. Substituting (8.10) into (8.11) gives the state equations for the circuit shown in Figure 8.3 as

$$Q_{A+1} = \overline{Q}_A Q_B + (\overline{Q_B x}) Q_A = \overline{Q}_A Q_B + Q_A \overline{Q}_B + Q_A x \quad (8.12a)$$

$$Q_{B+1} = \overline{x} \overline{Q}_B + (\overline{x \oplus Q_A}) Q_B = \overline{Q}_B \overline{x} + Q_A Q_B x + \overline{Q}_A Q_B \overline{x}. \quad (8.12b)$$

8.1.3 State transition table

The time sequence of inputs, outputs and flip-flop states can be enumerated in a state transition table (sometimes referred to simply as a *state table*). In general, the state transition table consists of five sections, namely the *present state*, *input*, *next state*, *output* and *flip-flop inputs*. Note that in some cases, not all the five sections are present in a state transition table, e.g. in the design of synchronous counters, the only columns necessary are the present state, next state and flip-flop inputs.

The present state section shows the states of the flip-flops at any time t . The input section gives the values of the external inputs for each possible present state. The next-state section shows the state of the flip-flops after an active clock transition. The output section of the state transition table gives the values of the outputs for each present state and input condition. The section on flip-flop inputs is strictly speaking not necessary in a state table, but it is usually included for purposes of evaluating the next state.

Example

Consider the circuit shown in Figure 8.2. The state transition table for this circuit is given below. The next state is determined using equations (8.6), while the output is determined using equation (8.8). Since the circuit uses D flip-flops, the flip-flop inputs column is not necessary.

8.1.4 State diagram

A state diagram is a graphical representation of the information in a state table. It shows the states and the possible transitions between the states.

States: A state is represented by a circle. A binary number in the circle identifies the state of the circuit.

State transitions: Transition between the states are indicated by directed lines connecting the circles. The directed lines are labelled with two binary numbers separated by a slash. The number before the slash is the input value during the present state, and the number after the slash is the output during the present state with the given input. It is important to remember that the bit value listed for the output along the directed line *occurs during the present state with the given input, and has nothing to do with the transition to the next state.*

There is no difference between a state table and a state diagram except in the manner of representation. The state table is easier to derive from a given circuit diagram. The state diagram gives a pictorial view of the state transitions and is more suitable for human interpretation of the circuit operation.

Example

Consider the circuit shown in Figure 8.2. The state table for this circuit is given in Section 8.1.3. The state diagram for this circuit is shown in Figure 8.4.

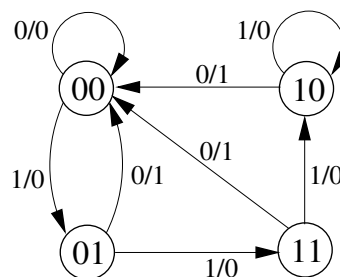


Figure 8.4: State diagram for circuit shown in Figure 8.2

Example

Figure 8.5 shows the state diagram of the circuit shown in Figure 8.3.

Exercise

Figure 8.6 shows a circuit made using T flip-flops. Write down the flip-flop input equations and hence determine the state equations for the circuit. Work out the state table and draw the state diagram of the circuit.

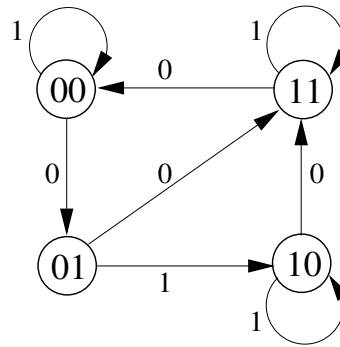


Figure 8.5: State diagram for circuit shown in Figure 8.3

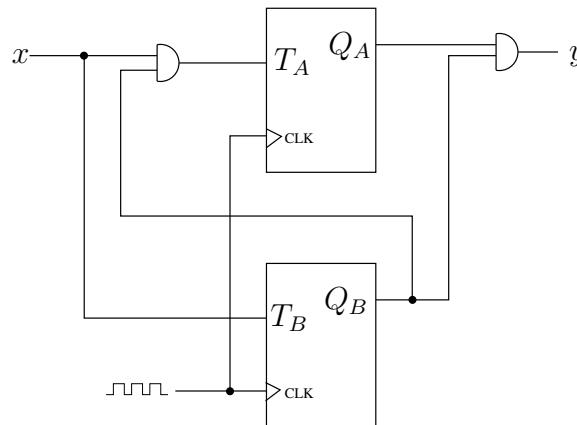


Figure 8.6: Exercise

8.2 State reduction

State reduction algorithms are concerned with procedures for reducing the number of states in a state table while keeping the input-output relationship unchanged. Since the input-output relationship remains unchanged, an input sequence applied to the original circuit and to the reduced circuit results in the same output sequence. State reduction may result in the reduction in the number of flip-flops.

Consider the state diagram shown in Figure 8.7. For state reduction, only the input-output relationship is important so the states are marked with letter symbols.

Suppose an input sequence 01010110100 is applied to the circuit, with the circuit initially in state *a*. For the given input sequence, the sequence of states and outputs that the circuit goes through is given below. Each column gives the present state, the input value and the current output.

state	<i>a</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>	<i>f</i>	<i>g</i>	<i>f</i>	<i>g</i>	<i>a</i>
input	0	1	0	1	0	1	1	0	1	0	0	
output	0	0	0	0	0	1	1	0	1	0	0	

To perform state reduction, the information in the state diagram is used to construct

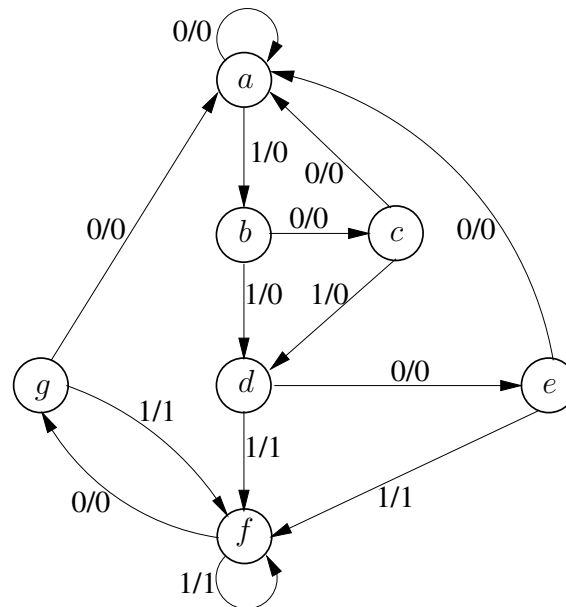


Figure 8.7: State diagram with states marked using letter symbols

a state table. A state table corresponding to the state diagram shown in Figure 8.7 is given below. The input is denoted as x while the output is denoted as y .

PRESENT STATE	NEXT STATE		OUTPUT y	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	f	0	1
e	a	f	0	1
f	g	f	0	1
g	a	f	0	1

An algorithm of state reduction, given here without proof, states that *two states are said to be equivalent if similar inputs give similar outputs and send the circuit to the same next-state or to an equivalent state*. When two states are equivalent, one can be removed without altering the input-output relationship.

To perform state reduction, we look at the state table and look for two present states that go to the same next state and have the same output for both input combinations. Looking at the state table above, we note that states e and g both go to the same next states, and they give the same outputs for both input combinations. State g can therefore be removed, and wherever g appears, it is replaced with e . A further look at the table shows that states f and d are similar and state f is removed and all occurrences of f are replaced with d . No further reduction is possible. The reduced state table is shown below:

PRESENT STATE	NEXT STATE		OUTPUT y	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	d	0	1
e	a	d	0	1

Note that states c and e have similar next states but since they give different outputs when the input $x = 1$, they are not equivalent and hence one of them cannot be removed.

The reduced state diagram is shown in Figure 8.8.

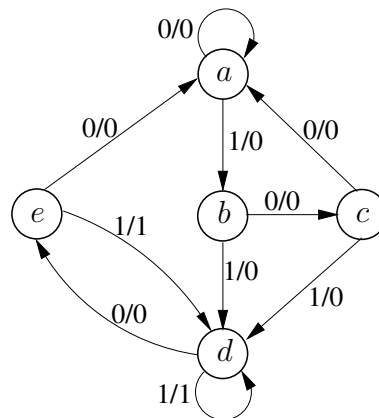


Figure 8.8: Reduced state diagram

If the input sequence 01010110100 is applied to the reduced circuit, with the circuit initially in state a , the sequence of states and outputs that the circuit goes through is given below:

state	a	a	b	c	d	e	d	d	e	d	e	a
input	0	1	0	1	0	1	1	0	1	0	0	
output	0	0	0	0	0	1	1	0	1	0	0	

Note that the input-output relationship is similar to the one obtained earlier.

8.3 Design procedure for clocked sequential circuits

1. From the word description and specifications of the desired operation, derive a state diagram for the circuit.
2. Reduce the number of states if possible.
3. Obtain a binary coded state transition table.

4. Choose the type of flip-flops to be used.
5. Using the flip-flop excitation tables of the flip-flops being used, compute the flip-flop inputs for the various states. Also determine the corresponding outputs.
6. Using a K-map or any other simplification method, derive the circuit output functions and the flip-flop input functions.
7. Draw the circuit diagram and implement the circuit.

Example

Design a JK flip-flop-based circuit with the state diagram shown in Figure 8.9.

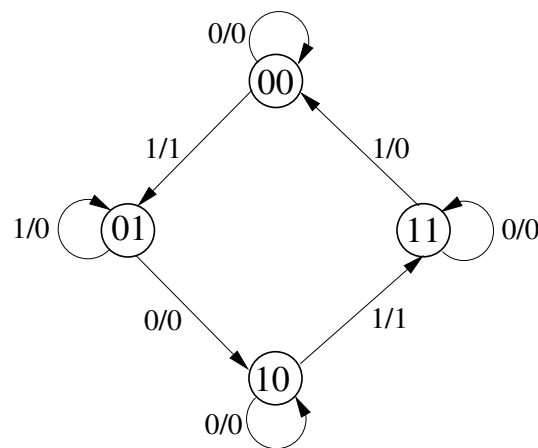


Figure 8.9:

Solution

Since the state diagram is already given, we go straight to the state reduction stage. To perform state reduction, the states are assigned letter symbols: state 00 is denoted as state *a*, state 01 as *b*, state 10 as *c* and state 11 as state *d*. The state table is shown below, with the input denoted as *x* and the output denoted as *y*.

PRESENT STATE	NEXT STATE		OUTPUT <i>y</i>	
	<i>x</i> = 0	<i>x</i> = 1	<i>x</i> = 0	<i>x</i> = 1
<i>a</i>	<i>a</i>	<i>b</i>	0	1
<i>b</i>	<i>c</i>	<i>b</i>	0	0
<i>c</i>	<i>c</i>	<i>d</i>	0	1
<i>d</i>	<i>d</i>	<i>a</i>	0	0

State reduction is not possible.

To determine the flip-flop inputs, the JK flip-flop excitation table below is used.

Q_n	Q_{n+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

The complete state transition table is given below:

PRESENT STATE		INPUT	NEXT STATE		OUTPUT	FLIP-FLOP INPUTS			
Q_A	Q_B	x	Q_{A+1}	Q_{B+1}	y	J_A	K_A	J_B	K_B
0	0	0	0	0	0	0	X	0	X
0	0	1	0	1	1	0	X	1	X
0	1	0	1	0	0	1	X	X	1
0	1	1	0	1	0	0	X	X	0
1	0	0	1	0	0	X	0	0	X
1	0	1	1	1	1	X	0	1	X
1	1	0	1	1	0	X	0	X	0
1	1	1	0	0	0	X	1	X	1

K-map simplification gives

$$J_A = Q_B \bar{x} \quad (8.13a)$$

$$K_A = Q_B x \quad (8.13b)$$

$$J_B = x \quad (8.13c)$$

$$K_B = Q_A x + \bar{Q}_A \bar{x} = \overline{Q_A \oplus x} = Q_A \odot x \quad (8.13d)$$

$$y = \bar{Q}_B x \quad (8.13e)$$

Exercise

Design a D flip-flop-based circuit described by the state table below.

PRESENT STATE		INPUT	NEXT STATE		OUTPUT
Q_A	Q_B	x	Q_{A+1}	Q_{B+1}	y
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	1	0	0
0	1	1	0	1	0
1	0	0	1	0	0
1	0	1	1	1	1
1	1	0	1	1	0
1	1	1	0	0	0

Exercise

Design a JK flip-flop-based circuit whose state diagram is shown in Figure 8.10.

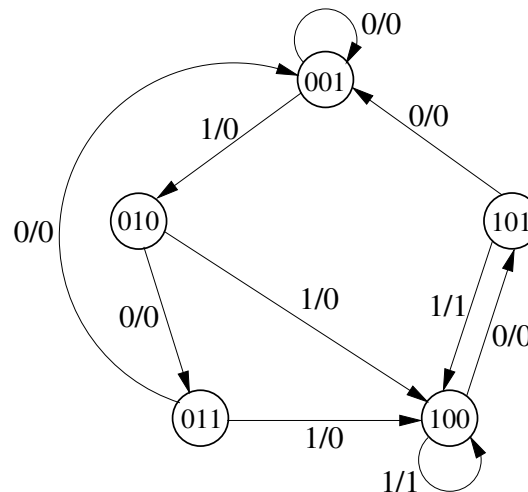


Figure 8.10:

Exercise

Design a JK flip-flop-based mod-4 synchronous counter with one external input x to control the counting sequence such that when $x = 0$, the circuit functions as a modulo-4 up counter, and when $x = 1$, the circuit functions as a modulo-4 down counter.

8.4 Timing circuits

8.4.1 Introduction

Timing circuits are used to produce clock signals for digital systems.

Circuits that generate rectangular waveforms are referred to as multivibrators. There are two types of multivibrators:

- ◇ Astable (or free running) multivibrator
- ◇ Monostable multivibrator (also known as a one shot)

An astable multivibrator is an oscillator which produces a rectangular waveform. It has two quasi-stable states and does not require any triggering and hence it is referred to as a free running multivibrator.

A monostable multivibrator has one stable state and one quasi-stable state. Under normal operating conditions, it is in the stable state (which can be HIGH or LOW). When the circuit is triggered by an externally applied pulse, it goes to the quasi-stable state and it remains in the quasi-stable state for a time duration which depends on the components used in the circuit.

8.4.2 The 555 timer

The 555 timer can function as a monostable multivibrator or as an astable multivibrator. The circuit diagram for the 555 timer is shown in Figure 8.11.

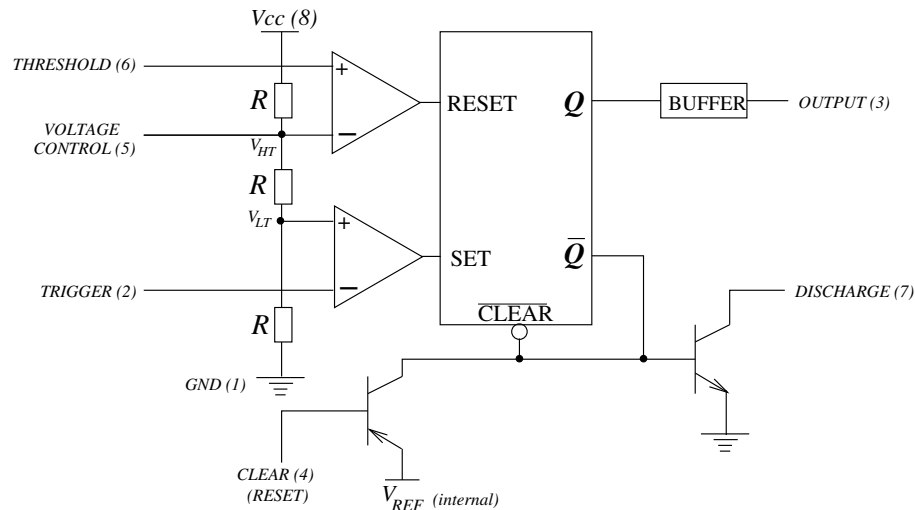


Figure 8.11: Circuit diagram for a 555 timer

Power supply: The IC can be powered by voltages between +4.5V and 15V. If the IC is to be used with TTL devices, it should be powered with $5V \pm 5\%$.

Output terminal: This is output pin of the IC. This output is TTL compatible. The output is buffered which makes it handle relatively large currents. The output can source or sink current.

Reset Terminal: When the RESET terminal is LOW, the 555 is disabled and does not respond to the voltages applied at the trigger terminal. Under such conditions, the asynchronous \overline{CLEAR} of the flip-flop is activated hence the output of the 555 is LOW and the discharge terminal is at a voltage $\approx 0V$.

Under normal operating conditions, the RESET terminal is kept in the HIGH state.

Discharge terminal: Used to discharge an external timing capacitor during the time when the output is LOW. When the output is HIGH, the discharge terminal acts as an open circuit and allows the capacitor to charge at a rate determined by an external resistor and capacitor. This is illustrated in Figure 8.12.

Voltage control terminal: This terminal can be used to change the threshold and trigger voltage levels. When this terminal is not used, it should be connected to 0V through a $0.01\mu F$ capacitor to prevent it from picking up noise voltages (in this case, $V_{LT} = \frac{1}{3}V_{CC}$ and $V_{HT} = \frac{2}{3}V_{CC}$).

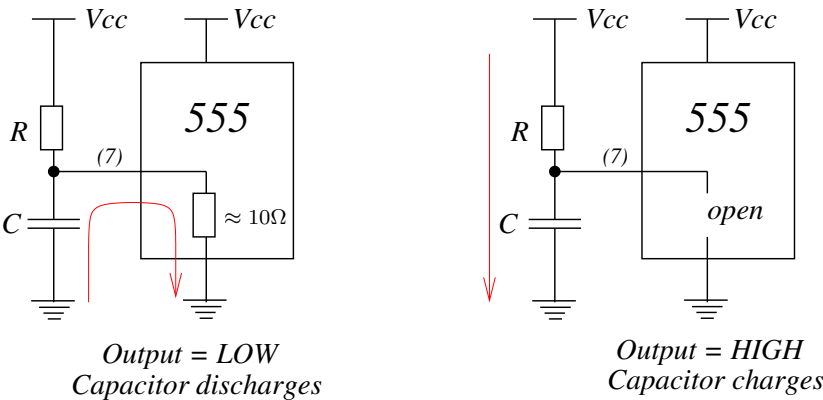


Figure 8.12: Charging and discharging of external capacitor

Trigger and threshold terminals: The table below shows the operation of the trigger and threshold terminals.

Operating Mode	Trigger	Threshold	Output	Discharge	Comment
A	Below V_{LT}	Below V_{HT}	HIGH	open	NOT USED (SET=RESET=1) Memory State
B	Below V_{LT}	Above V_{HT}			
C	Above V_{LT}	Below V_{HT}			
D	Above V_{LT}	Above V_{HT}	LOW	$\approx 0V$	

Suppose a 555 timer is connected as shown in Figure 8.13.

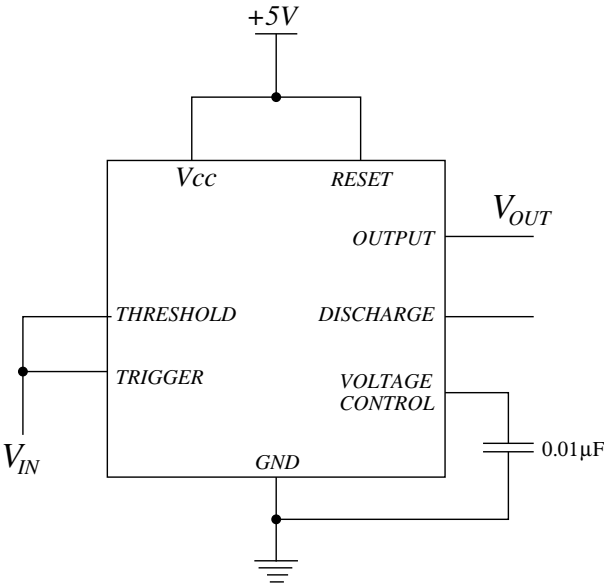


Figure 8.13: Input and output waveforms

Figure 8.14 shows an input waveform V_{IN} applied to the trigger and threshold terminals, and the output V_{OUT} observed at the output terminal.

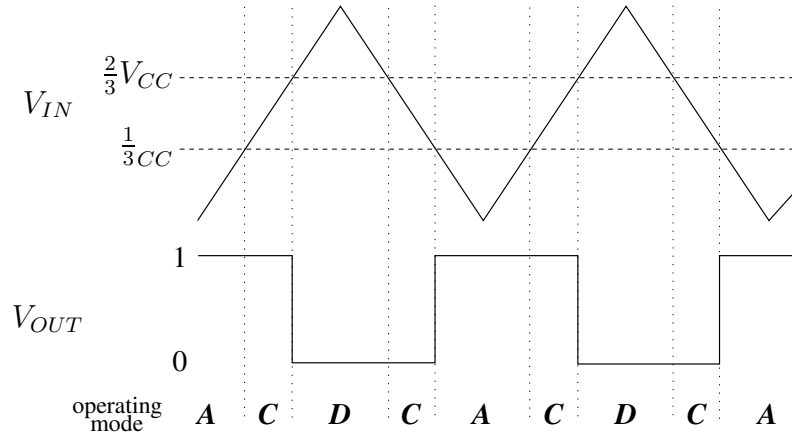


Figure 8.14:

8.4.3 Astable operation of the 555 timer

For astable operation, the 555 timer is connected as shown in Figure 8.15.

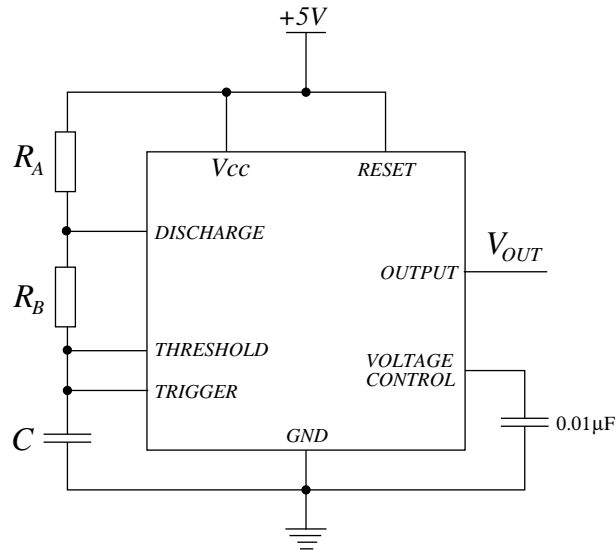


Figure 8.15: 555 timer connected as an astable multivibrator

When the output is HIGH, the capacitor charges through resistors R_A and R_B . When the output is LOW, the capacitor discharges through resistor R_B via the discharge terminal. Figure 8.16 shows the capacitor voltage v_c and the output voltage.

When the capacitor is charging through R_A and R_B ,

$$v_c(t) = V_{CC} \left(1 - e^{-\frac{t}{(R_A + R_B)C}} \right). \quad (8.14)$$

Solving for t gives

$$t = -(R_A + R_B) C \ln \left(1 - \frac{v_c(t)}{V_{CC}} \right). \quad (8.15)$$

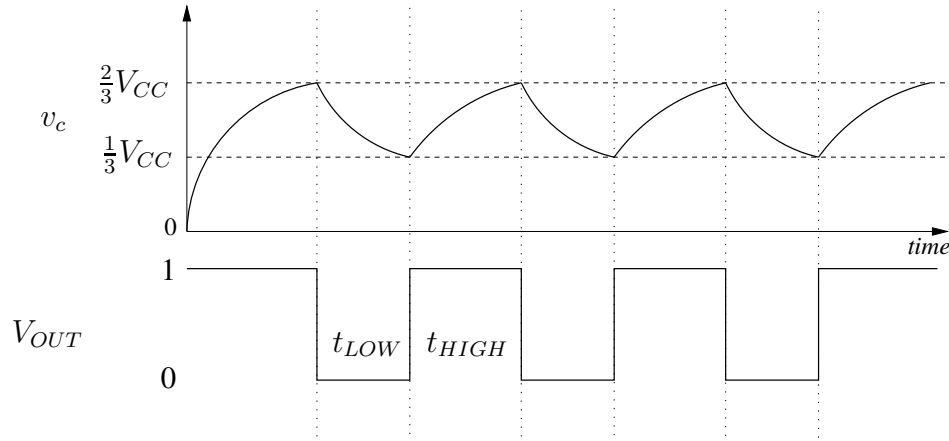


Figure 8.16:

Substituting $v_c(t) = \frac{1}{3}V_{CC}$, we can write

$$t_1 = -(R_A + R_B)C \ln \left(1 - \frac{V_{CC}}{3V_{CC}} \right) = -(R_A + R_B)C \ln \left(\frac{2}{3} \right). \quad (8.16)$$

Similarly, when $v_c(t) = \frac{2}{3}V_{CC}$,

$$t_2 = -(R_A + R_B)C \ln \left(\frac{1}{3} \right). \quad (8.17)$$

The time taken by the capacitor to charge from $\frac{1}{3}V_{CC}$ to $\frac{2}{3}V_{CC}$ is the clock HIGH time t_{HIGH} , given by

$$t_{HIGH} = t_2 - t_1 = -(R_A + R_B)C \left(\ln \frac{1}{3} - \ln \frac{2}{3} \right) = 0.693(R_A + R_B)C. \quad (8.18)$$

Similarly, it can be shown that the clock LOW time t_{LOW} , which is the time the capacitor takes to discharge from $\frac{2}{3}V_{CC}$ to $\frac{1}{3}V_{CC}$ through resistor R_B is given by

$$t_{LOW} = 0.693R_B C. \quad (8.19)$$

The period of the clock T is given by

$$T = t_{HIGH} + t_{LOW} = 0.693(R_A + 2R_B)C. \quad (8.20)$$

The frequency of the clock waveform is

$$f = \frac{1}{T} = \frac{1.44}{(R_A + 2R_B)C} \quad (8.21)$$

and the duty cycle is

$$= \frac{t_{HIGH}}{t_{HIGH} + t_{LOW}} \times 100\% = \frac{R_A + R_B}{(R_A + 2R_B)} \times 100\%. \quad (8.22)$$

8.4.4 Adjusting the duty cycle of a 555 timer

Suppose we would like to make the duty cycle to be 50%. From a mathematical viewpoint, this can be achieved by setting R_A to zero (equation (8.22)). However, this is not a good engineering solution since setting R_A to zero would cause a large current to flow from the supply to the discharge terminal when the output is LOW (at which time the discharge terminal offers a low resistance path to ground), and this could damage the IC. (The minimum recommended value of R_A is $1K\Omega$. Also, $R_A + R_B < 6.6M\Omega$ and $C > 500pF$). The practical way to get a 50% duty cycle (or close to that) is to set $R_B \gg R_A$.

To achieve more flexibility in setting of the duty cycle, a diode can be connected in parallel with R_B as shown in Figure 8.17.

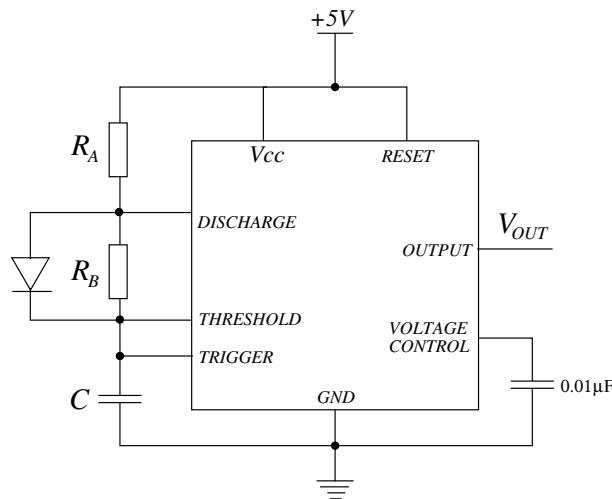


Figure 8.17:

In this case, the capacitor charges through resistor R_A and it discharges through resistor R_B . It then follows that

$$t_{HIGH} = 0.693R_A C \quad (8.23a)$$

$$t_{LOW} = 0.693R_B C. \quad (8.23b)$$

The diode makes it possible to set a duty cycle where $t_{LOW} > t_{HIGH}$.

8.4.5 Monostable operation of the 555 timer

For monostable operation, the 555 timer is connected as shown in Figure 8.18.

The trigger input is held at the supply voltage. Under these conditions, the stable output of the circuit is LOW. If the trigger input is momentarily connected to ground,

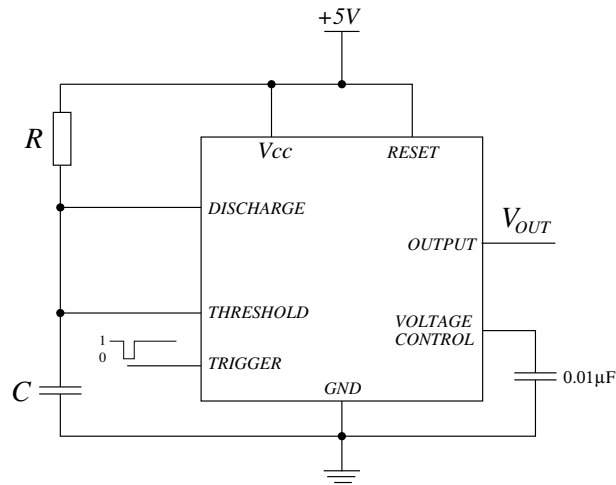


Figure 8.18: 555 timer connected as a monostable multivibrator

the output goes HIGH and the capacitor C starts charging through resistor R . The capacitor charges to a voltage of $\frac{2}{3}V_{CC}$ at which point the output goes LOW and the capacitor discharges. The circuit remains in this state (i.e. with the output in LOW state and the capacitor discharged) until the trigger input is pulsed LOW again.

The duration of the LOW to HIGH pulse at the output of the 555, denoted as t_P , is the time taken by the capacitor C to charge from $0V$ to $\frac{2}{3}V_{CC}$, and it can be shown that

$$t_P = 1.1RC. \quad (8.24)$$

Note that the circuit shown in Figure 8.18 will only work properly if the duration of the HIGH to LOW trigger pulse applied to the trigger input is shorter than t_P .

Exercise

For the circuit shown in Figure 8.18, show that if the trigger input is held HIGH, the stable output of the 555 is LOW.

Exercise

Prove equation (8.24).

8.4.6 Other monostable multivibrators

IC monostable multivibrators (or one-shots) are available in two forms: retriggerable and non-retriggerable.

Non-retriggerable: Once a non-retriggerable monostable multivibrator is in the quasi-stable state, it does not respond to triggering pulses. Figure 8.19 shows an example of triggering pulses and the corresponding outputs for this type of a monostable multivibrator.

An example of a non-retriggerable monostable multivibrator is the 74121.

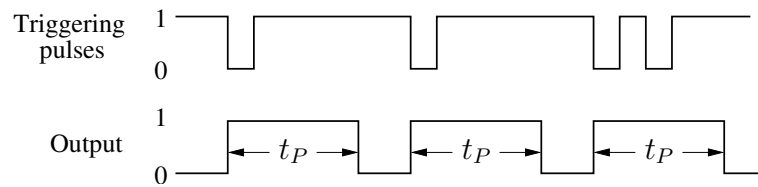


Figure 8.19: Waveforms for a non-retriggerable monostable multivibrator

Retriggerable: This type of monostable multivibrator can be retriggered when it is in the quasi-stable state. Figure 8.20 shows an example of triggering pulses and the corresponding outputs for this type of a monostable multivibrator.

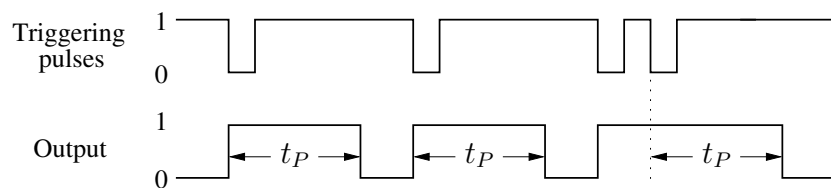


Figure 8.20: Waveforms for a retriggerable monostable multivibrator

Examples of retriggerable monostable multivibrators are the 74122 and the 74123.

The circuit diagram for a 74121 is shown in Figure 8.21.

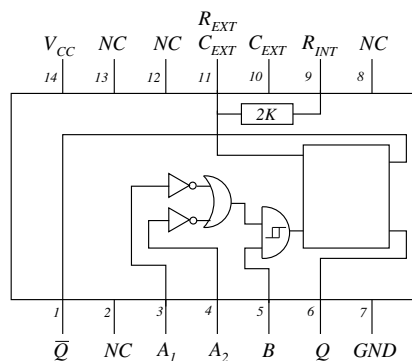


Figure 8.21: The 74121 non-retriggerable monostable multivibrator

The 74121 has three trigger inputs, A_1 , A_2 and B . Depending on the circuit design, any of these three pins may be connected to the input trigger signals. Inputs A_1 and A_2 are connected to TTL signals while input B is normally used for slowly varying inputs.

The functional truth-table for the 74121 is shown in Figure 8.22. The 74121 is triggered by a negative-going transition on A_1 and/or A_2 or a positive-going transition on the input B .











A_1	A_2	B	Q	\bar{Q}
0	X	1	0	1
X	0	1	0	1
X	X	0	0	1
1	1	X	0	1
1	↓	1		
↓	1	1		
↓	↓	1		
0	X	↑		
X	0	↑		

Figure 8.22: The truth table for the 74121

To use the 74121 as a monostable multivibrator, one can use the internal timing resistor R_{INT} or an external timing resistor R_{EXT} and capacitor C_{EXT} .

- ◇ To use the internal timing resistor, connect R_{INT} (pin 9) to V_{CC} , and leave R_{EXT}/C_{EXT} (pin 11) unconnected. When the 74121 is triggered, a LOW-to-HIGH pulse lasting 30ns-35ns appears at the output Q .
- ◇ To use an external timing resistor and capacitor, connect an external resistor to V_{CC} and R_{EXT}/C_{EXT} (pin 11), and an external capacitor between R_{EXT}/C_{EXT} and C_{EXT} (pins 11 and 10). For a $C_{EXT} > 1000\text{pF}$, a LOW-to-HIGH pulse of duration $t = 0.693R_{EXT}C_{EXT}$ is observed at the output Q when the 74121 is triggered.

More information on the 74121 monostable and other monostables can be obtained from the datasheets.

Chapter 9

ANALOG-TO-DIGITAL (A/D) AND DIGITAL-TO-ANALOG (D/A) CONVERSION

9.1 Introduction

Analog-to-digital conversion is the process of converting analog signals (signals which vary over a continuous range of values) to digital form (a binary code), while digital to analog conversion is the process of converting a digital signals into an analog form.

Many signal in the real world are analog e.g. voltages and currents in many electronic circuits, temperature, speed, flow-rate of a liquid in a pipe, level of liquid in a tank, voice signals etc. To process such information using digital devices, these signals have to be converted to digital form.

As an example, the sound signal from a microphone is analog. To store this information in a digital computer, it has to be converted into digital form using an analog-to-digital converter. For playback, this information is then converted back to analog form using a digital-to-analog converter. For the sound signals, analog-to-digital and digital-to-analog conversion is normally done in the sound card.

Another example is in control systems. The last two decades have seen an exponential increase in the computing power and speed of digital computers as well as the dramatic fall in their prices, factors which have led to their widespread use in process control. Since the systems to be controlled are typically analog systems, analog-to-digital and digital-to-analog converters are required.

This chapter examines the devices used for analog-to-digital and digital-to-analog conversion. Many analog-to-digital converters incorporate a digital-to-analog con-

verter in their circuitry, hence, digital-to-analog converters are considered first.

9.2 Digital-to-Analog Conversion

9.2.1 Introduction

Digital to analog conversion is the process of taking a value represented in a binary code and converting it to a voltage or current that is proportional to the digital value. The voltage or current is analog since it can take on many different values over a given range. The device which performs this conversion is known as a Digital to Analog Converter (DAC).

The binary code at the input to a DAC is usually unsigned binary or BCD code. For an unsigned binary code $b_n b_{n-1} \dots b_1 b_0$, the decimal equivalent is given by

$$b_n \times 2^n + b_{n-1} \times 2^{n-1} + \dots + b_1 \times 2^1 + b_0 \times 2^0. \quad (9.1)$$

Since the output of the DAC is proportional to the binary input, we can write that

$$\text{Output of DAC} \propto (b_n \times 2^n + b_{n-1} \times 2^{n-1} + \dots + b_1 \times 2^1 + b_0 \times 2^0). \quad (9.2)$$

From equation (9.2), it can be seen that the contributions of each digital input are weighted according to their position in the binary number. The output voltage can thus be considered to be the weighted sum of the digital inputs. The weights are successively doubled for each bit, beginning with the Least Significant Bit. This weighting factors are the basis on which Digital-to-Analog Converters (DACs) are made.

9.2.2 DAC using weighted resistors

The weighted resistor converter assigns weights to digital inputs by using appropriately weighted resistors. The resistors are weighted so that the resistors are inversely proportional to the numerical significance of the corresponding numerical digit. The operation of the converter depends on the successive resistances being related by a factor of 2 and does not depend on the absolute values of the resistors. An operational amplifier is employed as a summing amplifier which produces a weighted sum of the input voltages. An example of a 4-bit weighted resistor converter is shown on Figure 9.1.

DCBA is a four-bit digital word. Assuming a HIGH level voltage $= V_{REF}$ and LOW level voltage $= 0$, then we can write:

$$V_{OUT} = -V_{REF} \left(\frac{A}{8R} + \frac{B}{4R} + \frac{C}{2R} + \frac{D}{R} \right) R = -V_{REF} \left(\frac{A}{8} + \frac{B}{4} + \frac{C}{2} + D \right) \quad (9.3)$$

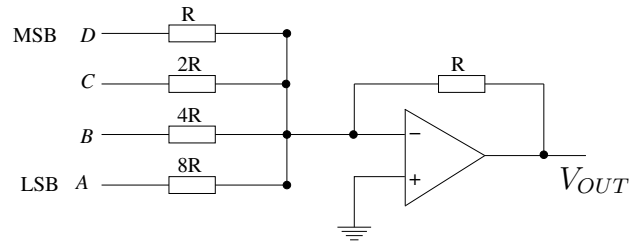


Figure 9.1: Weighted resistor DAC

For $V_{REF} = 5V$, the outputs from the above equation can be tabulated as below:

D	C	B	A	V_{OUT}
0	0	0	0	0
0	0	0	1	-0.625
0	0	1	0	-1.250
0	0	1	1	-1.875
0	1	0	0	-2.500
0	1	0	1	-3.125
0	1	1	0	-3.750
0	1	1	1	-4.375
1	0	0	0	-5.000
1	0	0	1	-5.625
1	0	1	0	-6.250
1	0	1	1	-6.875
1	1	0	0	-7.500
1	1	0	1	-8.125
1	1	1	0	-8.750
1	1	1	1	-9.375

The Resolution of a DAC is defined as the smallest change that can occur in the analog output as a result of a change in the digital input. From the table above, we can see that the resolution of the DAC is 0.625V, while the full scale analog output voltage is -9.375V.

Exercise

The resolution of an 8-bit DAC is 0.2V.

i) What is the output voltage when the input code is 01100100?

ii) What is the maximum analog output voltage?

(20V, 51V)

The value of the feedback resistor in Figure 9.1 can be used to change the range of the output voltage and the resolution e.g. suppose the feedback resistor was changed from R to $0.5R$ then

$$V_{OUT} = -V_{REF} \left(\frac{A}{8R} + \frac{B}{4R} + \frac{C}{2R} + \frac{D}{R} \right) \frac{R}{2} = -V_{REF} \left(\frac{A}{16} + \frac{B}{8} + \frac{C}{4} + \frac{D}{2} \right) \quad (9.4)$$

In this case, the resolution is 0.3125V, and the maximum analog output is -4.6875.

The main disadvantage of this method is the wide spread of resistance values for a large number of bits, e.g. suppose we used a $2K\Omega$ resistor for the MSB, then for a 14-bit DAC, the resistor for the LSB will be $2K\Omega \times 2^{13} = 16.4M\Omega$. Resistors covering such a wide range which have the required precision and which track over a wide

temperature range are difficult to produce.

9.2.3 Modified weighted resistors DAC

The weighted resistor DAC can be modified to accommodate a large number of bits without consequent spread in resistor values. An 8-bit modified weighted resistor DAC is shown on Figure 9.2.

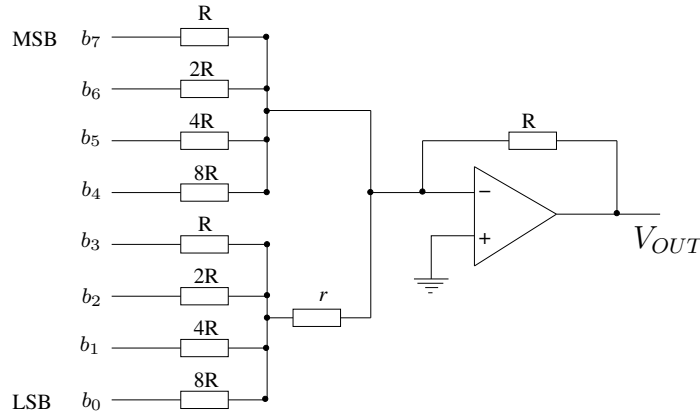


Figure 9.2: Modified weighted resistor DAC

The value of r is worked out in the following way:

Let the b_3 bit be 1 and b_2 , b_1 and b_0 are all 0. The portion of the circuit corresponding to this is shown in Figure 9.3.

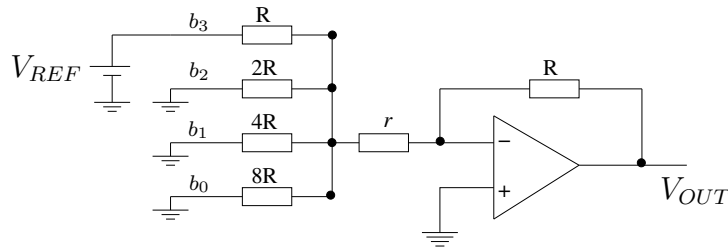


Figure 9.3: Modified weighted resistor DAC: $b_3 = 1, b_2 = b_1 = b_0 = 0$

Resistors $2R$, $4R$ and $8R$ are in parallel so we can redraw Figure 9.3 as shown on Figure 9.4.

Again noting that resistors $\frac{8R}{7}$ and r are in parallel, we can write the expression for the current I as:

$$I = \frac{V_{REF}}{\left(R + r // \frac{8R}{7}\right)} = \frac{V_{REF} \left(\frac{8R}{7} + r\right)}{\left(\frac{15rR}{7} + \frac{8R^2}{7}\right)} \quad (9.5)$$

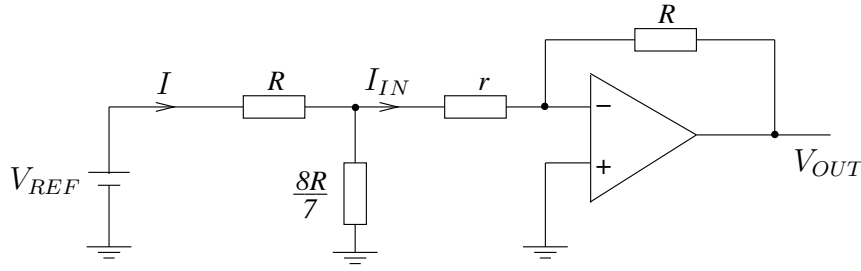


Figure 9.4: Circuit equivalent to Figure 9.3

By current division, we can deduce that:

$$I_{IN} = \frac{V_{REF} \left(\frac{8R}{7} + r \right) \frac{8R}{7}}{\left(\frac{15rR}{7} + \frac{8R^2}{7} \right) \left(\frac{8R}{7} + r \right)} = \frac{V_{REF} 8R}{15rR + 8R^2} \quad (9.6)$$

The current due to b_7 is V_{REF}/R and the current I_{IN} due to b_3 must be $\frac{1}{16}$ of this current due to b_7 . We can therefore write

$$\frac{V_{REF} 8R}{15rR + 8R^2} = \frac{V_{REF}}{16R} \quad (9.7)$$

which simplifies to:

$$r = 8R \quad (9.8)$$

Exercise

Given the circuit shown on Figure 9.2, assuming that the code $b_7b_6b_5b_4b_3b_2b_1b_0$ is in BCD code, show that $r = 4.8R$.

9.2.4 The R-2R Ladder Network

The R-2R ladder network uses resistors of only two sizes, R and 2R. This array requires twice as many resistors for the same number of input bits as the weighted resistor array. Bits are weighted by providing paths for current division with consequent successive attenuations for bits of lower significance. The op-amp is used to sum the current contributions of each bit. An example of a 4-bit R-2R ladder network is shown in Figure 9.5. The switches shown in the figure can be connected to ground (0) or to V_{REF} (1). In the example shown in Figure 9.5, the input code $DCBA = 0001$.

The network works on the principle of the current splitting exactly in half at each node. As such, it is the R-2R ratio of the resistors that is important and not the absolute value of the resistors. The current contribution from each bit is then passed through the op-amp to develop an output voltage V_{OUT} . The op-amp therefore converts binary-scaled currents to an output voltage.

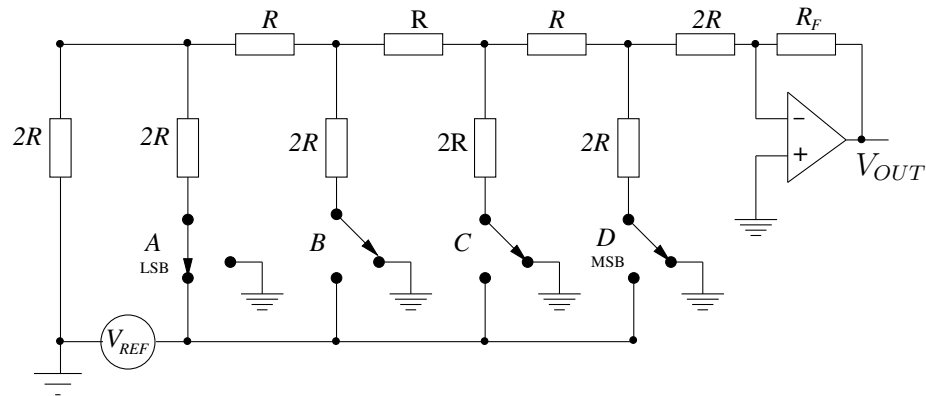


Figure 9.5: R-2R ladder network

Making R_F adjustable allows the output voltage to be set to any desired range within the saturation limits of the operational amplifier.

The circuit shown on Figure 9.5 has an input digital code DCBA = 0001. The equivalent circuit is shown in Figure 9.6.

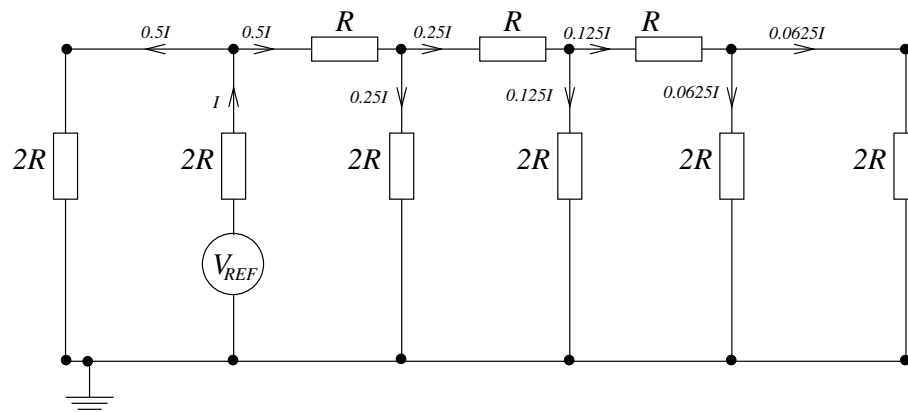


Figure 9.6: R-2R ladder network: equivalent circuit

Figure 9.6 can be simplified to the circuit in Figure 9.7.

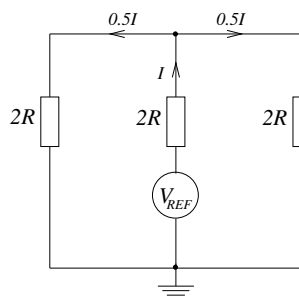


Figure 9.7: R-2R ladder network: simplified circuit

From Figure 9.7, we can write that

$$I = \frac{V_{REF}}{3R} \quad (9.9)$$

From Figure 9.6, we can see that the current contribution by the LSB = $0.0625I$. If the same principle is repeated with DCBA = 0010, DCBA = 0100 and DCBA = 1000, we would find that the respective current contributions would be $0.125I$, $0.25I$ and $0.5I$. Using the principle of superposition, we can therefore write

$$V_{OUT} = - \left(D \times \frac{I}{2} + C \times \frac{I}{4} + B \times \frac{I}{8} + A \times \frac{I}{16} \right) R_F. \quad (9.10)$$

Substituting (9.9) into (9.10) gives

$$V_{OUT} = - \frac{V_{REF}}{3R} \left(\frac{D}{2} + \frac{C}{4} + \frac{B}{8} + \frac{A}{16} \right) R_F. \quad (9.11)$$

9.2.5 DAC symbol

The symbol for a DAC is shown on Figure 9.8.

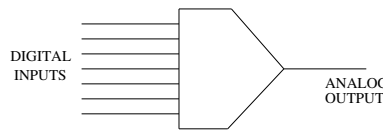


Figure 9.8: DAC symbol

9.2.6 Specifications of DACs

Resolution The smallest change that can occur in the analog output as a result of a change in the digital input. Resolution gets finer with an increase in the number of bits e.g. a 10-bit DAC has a finer resolution than a 4-bit DAC for the same output range. *An ideal DAC should have an infinitely small resolution.*

Linearity In a DAC, equal increments in the numerical significance should result in equal increments in the analog output voltage. In an actual circuit, the input-output relationship is not linear. This is due to errors in the resistor values and voltage drops across the switches. The linearity of a converter is a measure of the precision with which the linear input-output relationship is satisfied. The linearity error is the maximum deviation of the step size from the ideal step size.

Accuracy The accuracy of a DAC is the difference between the actual output voltage and the expected output voltage. It is usually specified as a percentage of the full-scale output voltage.

Settling time This is used to specify the conversion speed of a DAC. It is the time required for the output to go from zero to full-scale as the binary inputs are varied from *all zeros* to *all ones*.

Temperature Sensitivity At any fixed digital input, the analog voltage will vary with temperature. Temperature sensitivity specifies by how much the output changes per unit change in temperature.

Offset Voltage Ideally, the output of a DAC should be zero volts when the binary input is all zeros. In practice, there will be a small output voltage caused by the offset error in the output op-amp. Most DACs have an offset adjustment capability that allows the user to zero the offset error.

Examples of DAC ICs are 12-bit AD767, 8-bit AD557JN (these two ICs are microprocessor-compatible), 8-bit ZN425E, 8-bit ZN435E, 8-bit ZN428E etc. More information about these DACs can be obtained from the data books.

9.3 Analog to Digital Conversion

An analog to digital converter takes an analog input voltage and produces a digital output code which represents the analog input. There are many methods of analog to digital conversion and in this course, we shall look at the most common methods.

9.3.1 Digital Ramp Analog to Digital Converter

A digital ramp analog-to-digital converter is shown in Figure 9.9. (For the comparator shown in the figure, if $V_A > V_B$, $EOC = \text{HIGH}$, and $V_A < V_B$, $EOC = \text{LOW}$.)

Operation of the Digital Ramp ADC

A START pulse is applied and this resets the counter to zero. When the START pulse is HIGH, it inhibits the AND gate so that no clock pulses get through to the counter. With the counter set to zero, $V_B = 0$ so the comparator output is HIGH.

When the START pulse returns LOW, the AND gate is enabled and pulses allowed into the counter. As the counter advances, the DAC's output V_B increases in steps of voltage equal to its resolution. When V_B reaches V_A , the comparator output goes LOW, inhibiting pulses to the counter so that the counter stops and holds the desired digital

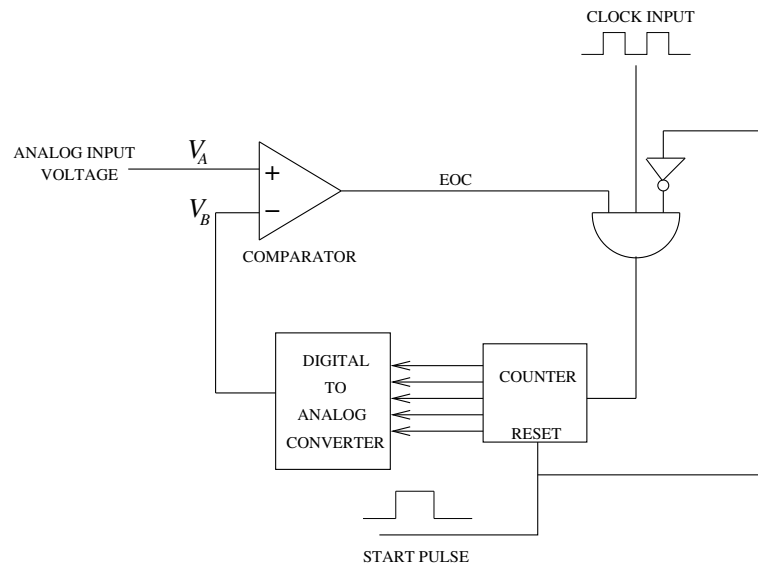


Figure 9.9: Digital Ramp ADC

code representing V_A . The HIGH to LOW transition at the comparator output signals the End Of Conversion (EOC). This is illustrated by Figure 9.10. The conversion time is the interval between t_0 and t_1 .

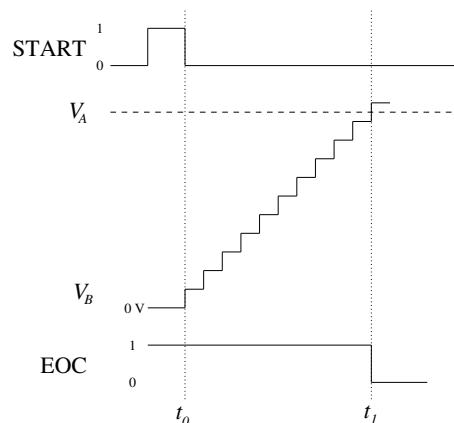


Figure 9.10: Digital Ramp ADC timing

Note that the conversion time for this converter is not constant: the higher the analog input, the higher the conversion time.

Exercise

A ramp ADC has a clock running at a frequency of 1MHz. The input voltage ranges from 0 to 40.95V for a 12-bit output. Determine:

- The resolution of the ADC
- The digital equivalent of 7.828V
- The conversion time for $V_A = 7.828V$
- The maximum conversion time for this converter.

(10mV, 001100001111, 783 μsec , 4095 μsec)

9.3.2 The Successive Approximation Converter

This method is essentially a binary search. It works by trying various binary codes and feeding them into a DAC and comparing the result with the analog input via a comparator. This converter is illustrated in Figure 9.11. (For the comparator shown in the figure, if $V_A > V_B$, COMP = HIGH, and if $V_A < V_B$, COMP = LOW.)

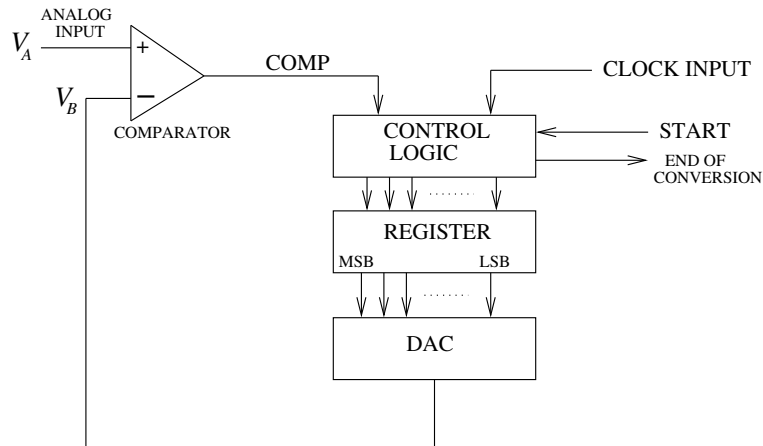


Figure 9.11: Successive Approximation ADC

Operation of the successive approximation ADC

- i) The control logic sets the MSB of the register HIGH and all the other bits LOW. This produces a value of V_B at the DAC output that is equal to the weight of the MSB. If V_B is now greater than V_A , the comparator output goes LOW and causes the control logic to clear the MSB back to LOW. Otherwise the MSB is kept HIGH.
- ii) The control logic sets the next bit of the register to 1. This produces a new value of V_B . If the value of V_B is greater than V_A , the comparator output goes LOW to tell the control logic to clear the bit back to 0. Otherwise the bit is kept at 1.
- iii) The process is repeated for each of the bits in the register. After all the bits have been tried, the register holds the digital equivalent of V_A .

Testing each bit takes one clock cycle so for an N -bit converter, conversion will take N clock cycles regardless of the value of the analog input voltage. (In other words for a given number of bits, the conversion time is constant).

As an example to illustrate this procedure, consider a 6-bit successive approximation converter with a resolution of 1V (it can therefore convert voltages in the range 0-63V). If an input voltage of 37.4V is applied as V_A , the output can be determined using the table below:

STEP	ACTION	REGISTER CONTENTS	V_B	V_A	COMP	COMMENT
	Initial State	000000	0	37.4	HIGH	
1	Set MSB to 1	100000	32	37.4	HIGH	Leave MSB at 1
2	Set 2^{ND} bit to 1	110000	48	37.4	LOW	Reset 2^{ND} bit to 0
3	Set 3^{RD} bit to 1	101000	40	37.4	LOW	Reset 3^{RD} bit to 0
4	Set 4^{TH} bit to 1	100100	36	37.4	HIGH	Leave 4^{TH} bit at 1
5	Set 5^{TH} bit to 1	100110	38	37.4	LOW	Reset 5^{TH} bit to 0
6	Set 6^{TH} bit to 1	100101	37	37.4	HIGH	Leave 6^{TH} bit at 1

The digital equivalent of 37.4V is the final value in the register, which is 100101. The profile of the voltage V_B during the conversion is shown in Figure 9.12.

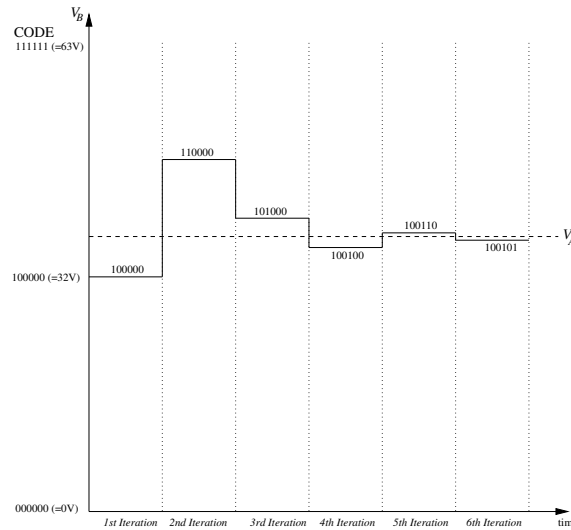


Figure 9.12: Profile of voltage V_B during the conversion

9.3.3 Dual Slope Integration ADC (the integrating converter)

The block diagram of the dual-slope integration converter is shown in Figure 9.13.

The conversion begins with the electronic switch in the T_1 position, with the counter reset to zero and the capacitor fully discharged. The input voltage causes the capacitor connected to the op-amp to charge at a rate dependent on the magnitude of the input voltage. For the integrator, we can write that

$$V_{OUT} = -\frac{1}{RC} \int_0^t v_{in} dt + \text{initial capacitor voltage} \quad (9.12)$$

The capacitor has no initial charge and the input voltage v_{in} is a constant voltage V_{IN} . Substituting these values and integrating gives

$$V_{OUT} = -\frac{V_{IN}t}{RC} \quad (9.13)$$

Equation (9.13) is the equation of a straight line starting at the origin and with a gradient of $-\frac{V_{IN}}{RC}$.

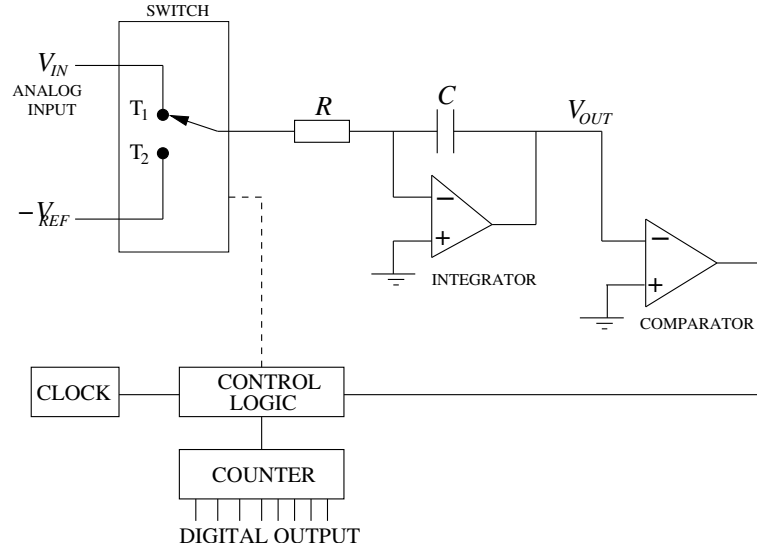


Figure 9.13: Dual-Slope ADC

For a fixed time T_1 ,

$$V_{OUT} = -\frac{V_{IN}T_1}{RC} \quad (9.14)$$

Hence

$$\text{integrator output for a fixed time } T_1 \propto \text{input voltage } V_{IN}. \quad (9.15)$$

After T_1 seconds have elapsed, the switch is moved to T_2 by the control logic. The higher the input voltage V_{IN} , the greater the charge stored in the capacitor during T_1 . Because the reference voltage is of the opposite polarity to the analog input, the capacitor now discharges. The control logic also starts the counter the moment the switch is moved to T_2 . When the capacitor has fully discharged, the comparator output level switches and the counter is halted. The value in the counter is directly proportional to the input voltage V_{IN} .

Consider the integrator equation (9.12). During the discharge phase, the input voltage v_{in} equals a constant voltage $-V_{REF}$, while the initial capacitor voltage is $-\frac{V_{IN}T_1}{RC}$. Substituting these values in equation (9.12) gives

$$V_{OUT} = -\frac{1}{RC} \int_0^t (-V_{REF}) dt + \left(-\frac{V_{IN}T_1}{RC} \right). \quad (9.16)$$

Integrating,

$$V_{OUT} = \frac{V_{REF}t}{RC} - \frac{V_{IN}T_1}{RC} \quad (9.17)$$

This is an equation of a straight line with a gradient $\frac{V_{REF}}{RC}$.

(Note that the charging graph and discharging graphs have different slopes, hence the name dual-slope converter).

When the capacitor has fully discharged, $V_{OUT} = 0$. Substituting this in equation (9.17) gives

$$t = T_2 = \frac{V_{IN}T_1}{V_{REF}}. \quad (9.18)$$

Hence

$$\text{discharge time} \propto \text{input voltage } V_{IN} \quad (9.19)$$

i.e. larger input voltages V_{IN} will require more time to discharge.

The charging and discharging graphs are shown on Figure 9.14.

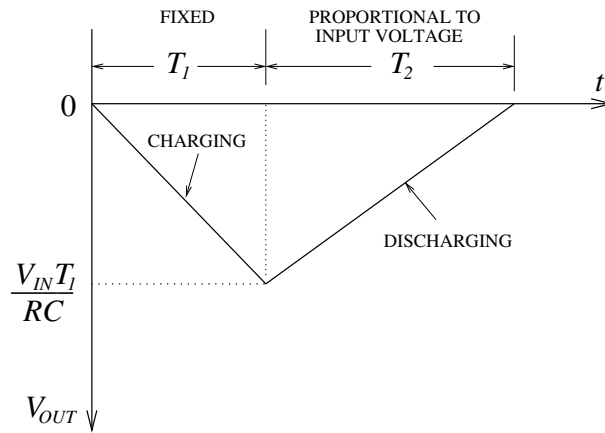


Figure 9.14: Dual-Slope ADC:charging and discharging graphs

For the dual-slope ADC, a comparison of 3 analog input voltages V_1 , V_2 and V_3 where $V_1 < V_2 < V_3$ is shown in Figure 9.15.

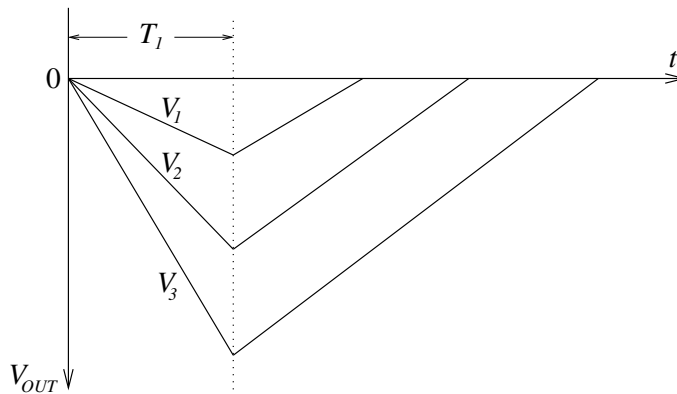


Figure 9.15: Dual-Slope ADC with analog inputs V_1 , V_2 and V_3 where $V_1 < V_2 < V_3$

Dual-slope converters are very accurate, but they are slow as they are essentially counting converters. They are commonly used in digital multimeters, and panel meters where conversion speed is not very important. Advantages of dual slope ADCs compared to other ADCs are simplicity, low cost and relative immunity to noise due to the long conversion time.

9.3.4 Flash ADCs

These are the fastest ADCs, where the conversion time equals the sum of the comparator plus encoder propagation delays. A typical flash converter has a conversion time of about 50ns, compared with a typical 20 μ sec for successive approximation ADCs. A 3-bit flash ADC is illustrated in Figure 9.16. (For the comparators shown in the figure, if $V_+ > V_-$, output = HIGH, if $V_+ < V_-$, output = LOW.)

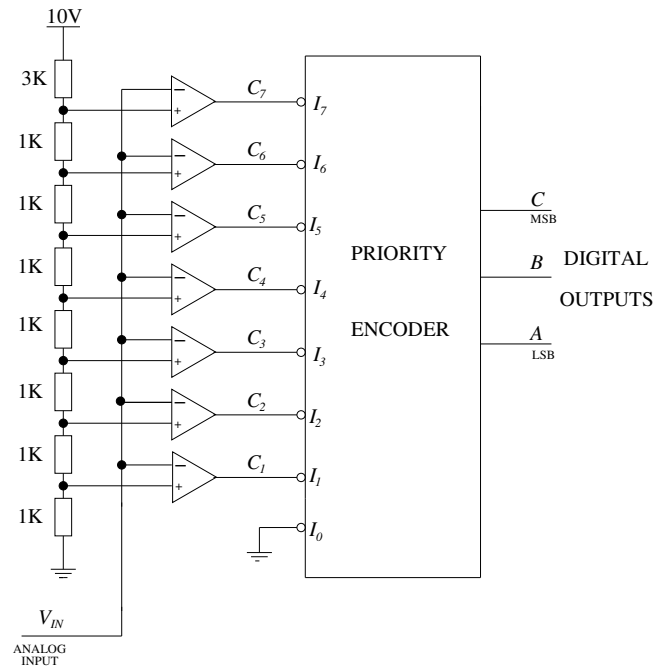


Figure 9.16: Flash ADC

The operation of this converter can be summarized in the table shown below:

V_{IN}	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C	B	A
$< 1V$	1	1	1	1	1	1	1	0	0	0
$\geq 1V, < 2V$	0	1	1	1	1	1	1	0	0	1
$\geq 2V, < 3V$	0	0	1	1	1	1	1	0	1	0
$\geq 3V, < 4V$	0	0	0	1	1	1	1	0	1	1
$\geq 4V, < 5V$	0	0	0	0	1	1	1	1	0	0
$\geq 5V, < 6V$	0	0	0	0	0	1	1	1	0	1
$\geq 6V, < 7V$	0	0	0	0	0	0	1	1	1	0
$\geq 7V$	0	0	0	0	0	0	0	1	1	1

The main disadvantage of this converter is the large number of comparators required. An N -bit converter requires $(2^N - 1)$ comparators and this increases the circuit complexity and cost when N is large. The use of flash ADCs is restricted to applications where speed is the prime requirement.

9.3.5 ADC symbol

The symbol for an ADC is shown in Figure 9.17.

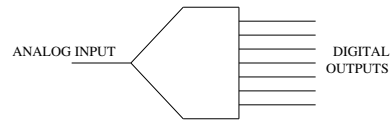


Figure 9.17: ADC symbol

9.3.6 ADC specifications

- Analog input range – maximum allowable input voltage range e.g. $0 - 10V$, $\pm 5V$ etc.
- Input impedance $1K\Omega$ to $1M\Omega$.
- Conversion time.
- Format of the output digital code - 2s complement, unsigned binary, BCD etc.

Examples of ADC ICs are: 12-bit successive approximation AD7870, 8-bit 'flash' ADC-304 (TTL compatible outputs), 8-bit 'flash' MC10319P (has tri-state outputs), 8-bit ZN439, 8-bit successive approximation ZN427E (microprocessor compatible) etc. More information about these ICs can be obtained from data books.

Chapter 10

INTRODUCTION TO MICROCOMPUTERS AND MICROCOMPUTER ORGANIZATION

10.1 Introduction

A microcomputer is defined as a stand-alone system based on a microprocessor. A stand-alone system is one that is able to operate without additional equipment.

A microprocessor is the central processing unit (CPU) of a microcomputer constructed within a single chip, and it is entirely useless on its own. To make it into a viable microcomputer, it needs to be connected to:

- i) Memory to store data and programs
- ii) Input/Output devices to communicate with the outside world
- iii) Timing and control circuits
- iv) A power supply

There are two types of microcomputers: the general purpose digital microcomputer and the embedded microcomputer. The general purpose digital microcomputer is what most people understand by the word *computer*. It is what is commonly referred to as a *personal computer (PC)* and it has all the necessary memory and input/output devices required by the user to execute a wide range of application programs.

An embedded computer is one dedicated to a specific application. A good example of an embedded microcomputer lies at the heart of an Automated Teller Machine (ATM). When a customer inserts his/her card in a slot, the microcomputer reads the relevant details from the magnetic strip on the card. The customer keys in an identity code (PIN), and the microcomputer validates the code and then invites the customer

to perform a transaction. Once the transaction is completed, the microcomputer updates the data on the magnetic strip and operates a mechanical subsystem that returns the card to the user.

A system incorporating an embedded microcomputer is referred to as an embedded system. Examples of embedded systems include:

- ◇ Photocopiers and printers
- ◇ Engine controllers, anti-lock braking systems, traction control systems and navigational systems for modern automobiles.
- ◇ Home automation systems like thermostats, air conditioners and security monitoring systems.
- ◇ Household appliances including microwave ovens, washing machines, television sets, DVD players. etc.
- ◇ Flight control systems and missile guidance systems.
- ◇ Medical equipment
- ◇ Measuring equipment such as digital storage oscilloscopes and logic analyzers.
- ◇ Mobile phones with additional capabilities, e.g. mobile phone with PDA.
- ◇ Video game consoles.
- ◇ Programmable Logic Controllers (PLCs) for industrial automation.
- ◇ Traffic light controllers.

The fundamental difference between an embedded microcomputer and a general purpose microcomputer is the optimization of the former to suit a single function. Typically, an embedded microcomputer contains only the functions strictly necessary for it to execute the allocated task. A general purpose microcomputer has the memory and input/output devices to perform a broad range of functions. In particular, a general purpose microcomputer almost always has facilities for expansion, thereby enabling the user to add more memory or input/output devices at a later date.

10.2 Basic Microcomputer (μC) Components

A *basic* microcomputer is shown on Figure 10.1

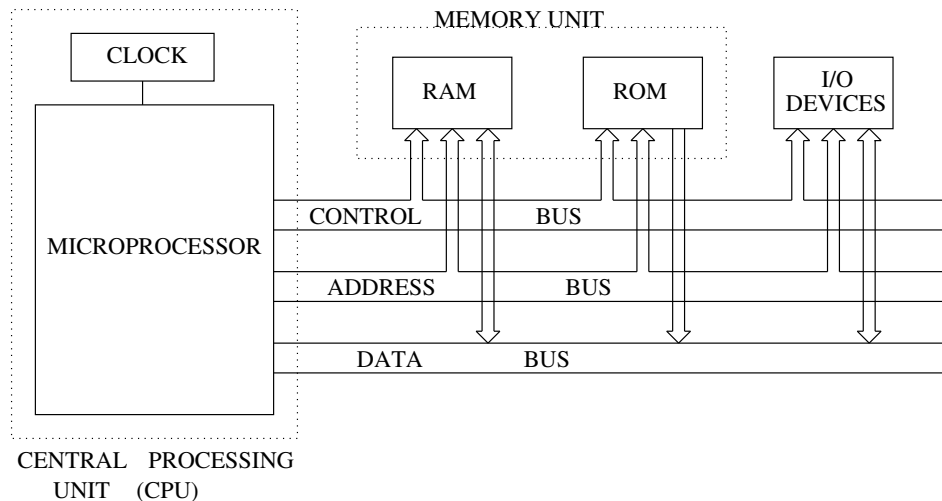


Figure 10.1: Basic microcomputer components

Memory Unit Stores the instructions that the computer is going to perform (programs) and the data to be operated on by the programs. The memory unit is also used to store intermediate and final results of operations performed during program execution. The operation of the memory is controlled by information carried in the control bus.

Input/Output Unit Contains the circuits that form an interface between the microcomputer and the rest of the world. For some devices, data transfer with the microcomputer is in serial form (one bit at a time) e.g. the CRT, while for others, data transfer is in parallel form e.g. the microcomputer transfers information to a printer one *byte* at a time. Examples of input devices include ADCs, Keyboards, mouse, joystick e.t.c. Output devices include DACs, printers, LED readouts, CRTs etc.

The bus system Consists of the data bus, the control bus and the address bus. These buses connect the microprocessor to each of the memory and I/O elements so that information can be exchanged between the microprocessor and these devices.

The Central Processing Unit (CPU)/Microprocessor It performs a large number of functions, including:

- directing the operation of all other units in the microcomputer by providing timing and control signals for all elements.
- fetching instructions and data from memory.
- transferring data to and from input/output devices.
- performing arithmetic and logic operations on data

- responding to input/output generated control signals such as RESET and INTERRUPT.

10.3 The Bus System (The Highway Structure)

This comprises the Address Bus, the Data Bus and the Control Bus.

The Address bus This is a unidirectional bus, because information flows over it only in one direction - from the CPU to the memory or I/O elements. It is used to specify the address of a memory location or an I/O device involved in a data transfer.

The Data Bus This is a bidirectional bus, since data can flow to or from the CPU. It is used to carry data associated with a memory or I/O transfer. During a READ operation, the CPU's data pins act as inputs and receive data that has been placed on the data bus by the memory or I/O element selected by the address on the address bus. During a WRITE operation, the CPU's data pins act as outputs and place data on the data bus, which is then sent to the selected memory or I/O device.

The Control Bus This is a set of signals that are used to synchronize the activities of the separate microcomputer elements. Some control signals are from the CPU to other elements to tell them what kind of operation is in progress, e.g. READ/WRITE signal. The I/O elements can also send control signals to the CPU e.g. RESET signal which causes the CPU to reset to a particular starting state, INTERRUPT REQUEST signal used by I/O devices to get attention of the CPU when it is performing other tasks.

10.4 Microprocessors

10.4.1 General introduction

A microprocessor is the central processing unit (CPU) of a microcomputer constructed within a single chip. For many years, manufacturers identified the microprocessors with number codes such as 4004, 8086, 68000, Z8000, etc., but of late, the microprocessors have brand names e.g. PowerPC, Pentium, Itanium, Xeon, etc.

In 1971 the Intel Corporation produced the first microprocessor, the 4-bit 4004, and since then, there has been an explosion of research in this area, spawning thousands of microprocessors.

Microprocessors are usually classified by the number of bits the microprocessor can process at a time. The Intel 4004 could only manipulate 4 bits at a time hence it was referred to as a 4-bit microprocessor. Modern microprocessors are 32-bit or 64-bit. Generally, a microprocessor with a large number of bits can process instructions more quickly than one with a small word size when both microprocessors are running at the same clock speed. Microprocessors are also designed such that the ones with a larger word size can process more instructions than their predecessors (the computer made with the Intel 4004 was a calculator-like device, while modern 32 and 64-bit microprocessors are used to make full multimedia computers (those that can process data, video and sounds)).

Microprocessors are made for different applications. The most powerful microprocessors are used in servers and workstations. Then there are microprocessors for PCs (microcomputers). At the lower end are the microprocessors used in embedded systems – these are low-cost microprocessors, and they make over well over 50% of all the microprocessors that are sold. Microprocessors have provided a cost-effective solution for control and many modern appliances have a number of microprocessors in their control circuitry. A modern luxury car, such as a Mercedes Benz S-class or a BMW 7-series has over 100 microprocessors for functions such as engine management, dashboard control, braking system, traction control, climate control, cruise control, navigational system, etc. A Boeing 777 jet has over 1200 microprocessors.

In this course, we concentrate on the microprocessors used in microcomputers (PCs). The PC market is dominated by Intel Corporation (with over 50% market share). The table below gives a summary of some microprocessors by Intel.

Microprocessor	Year	No. of transistors	Word Size	Clock speed
4004	1971	2300	4 bits	108kHz
8008	1972	3500	8 bits	500kHz
8080	1974	6000	8 bits	2MHz
8085	1976	6500	8 bits	5MHz
8086	1978	29000	16 bits	up to 10MHz
80286	1982	134000	16 bits	up to 12.5MHz
80386DX	1985	275000	32 bits	up to 33MHz
80486DX	1989	1.2 million	32 bits	up to 50MHz
80486DX2	1992	1.2 million	32 bits	up to 66MHz
80586 (renamed Pentium I)	1993	3.1 million	32 bits	up to 200MHz
Pentium II	1997	over 5 million	32 bits	up to 800MHz
Pentium III	1999	over 10 million	32 bits	up to 1200MHz
Pentium IV	2000	over 40 million	32 bits	

Intel microprocessors are used in IBM computers (and IBM compatibles, or clones).

Other manufacturers of microprocessors for microcomputers include Motorola, Advanced Micro Devices (AMD), Apple, IBM and National Semiconductors. The Motorola range of microprocessors features the 8-bit 6800, and the 32-bit 68000, 68020, 68030, 68040 and the 68060. Motorola microprocessors are used in Apple computers. Motorola have since abandoned the 68XXX series microprocessors and have teamed up with IBM and Apple to produce the PowerPC range of microprocessors.

The most recent microprocessors are 64-bit and these include Intel's Xeon and Itanium 2, AMD's Opteron and Athlon 64, and IBM's Power PC G5. These processors are mostly used in servers, but they are slowly trickling down to the PC market.

NOTES:

- *Some microprocessors have a data bus of a smaller size than the data word that can be processed by the microprocessor e.g. the Intel 8088 is a 16-bit microprocessor but it has an 8-bit data bus, compared to the 16-bit Intel 8086 that has a 16-bit data bus. This means that the 16-bit word to be processed by the 8088 has to be fetched from memory in two (READ) cycles, compared to the one cycle required for the 8086. The smaller data bus makes the 8088 less expensive than the 8086 but the 8088 is significantly slower than the 8086.*
- *A math coprocessor (also known as a numeric coprocessor) is a special circuitry that performs complex arithmetic operations much faster than the microprocessor's Arithmetic and Logic Unit (ALU). Computation-intensive activities are finished faster if a microcomputer has a math coprocessor. The older microcomputer systems had a separate coprocessor ICs but the newer microprocessors (Pentium I and up) have the math coprocessor built into the microprocessor.*
- *High speed microprocessors such as the Pentium process data so quickly that they often wait for data to be delivered from RAM, and this slows down the processing. Cache memory is a special high-speed Read/Write Memory that gives the microprocessor more rapid access to data. As you begin a task, the computer anticipates what data you are likely to need for this task and loads or "caches" this data into the memory. Then, when an instruction calls for data, the microprocessor first checks to see if the required data is in the cache, instead of fetching it from the slower RAM. Cache memory can be built into the microprocessor and/or in a separate chip.*
- *The Pentium microprocessors have millions of transistors packed within a small area e.g. the Pentium IV processor has over 40 million transistors. The millions of transistors switching in such close proximity collectively generate a lot of heat energy which could potentially damage the IC. For this reason, Pentium microprocessors have fans mounted on the IC for cooling purposes.*

10.4.2 General microprocessor block diagram

In block diagram form, a basic microprocessor can be represented as shown in Figure 10.2.

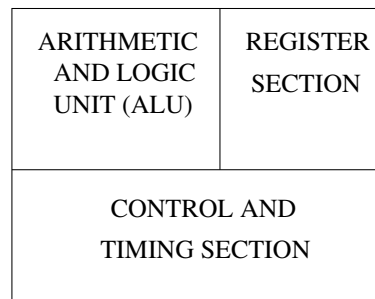


Figure 10.2: Basic microprocessor: Block diagram

The register section contains the registers that the microprocessor uses to store temporary information. The timing and control unit is the one that directs the operation of all the components of a microcomputer while the Arithmetic and Logic Unit (ALU) is the one that performs arithmetic and logic operations on data.

10.5 The Programmer's model of a microcomputer

10.5.1 Introduction

The programmer sees the microcomputer as having three characteristics:

An address space This is a set of storage locations, each of which has an address and contains binary coded data. The storage locations include the computers main memory and I/O devices. The size of the address space is defined by the number of bits the computer uses to form an address e.g. 16-bit address bus can access 2^{16} locations.

Registers Registers are used to hold temporary data and control information. Some registers have specialized functions while others are general purpose registers.

Instruction Set Is the set of commands which the computer can obey, out of which machine code programs are constructed. The instruction set includes operations that can be performed on operand data and the addressing modes available for accessing that operand data.

10.5.2 The Programmer's Model of the Intel 8085

Address Space = $2^{16} = 64K = 65536$ 8-bit words.

Register Set: This is shown in Figure 10.3.

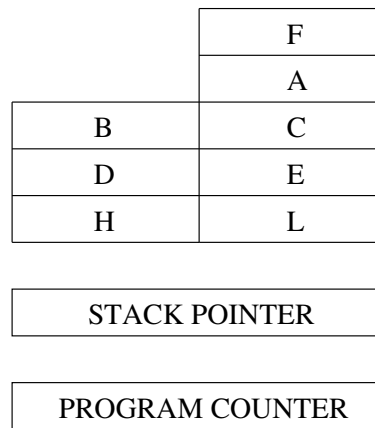


Figure 10.3: 8085 register set

Register A This is known as the accumulator. It is the register which contains the results of all mathematical and logic operations. Data transfers between the 8085 and input/output devices take place through this register.

Program Counter A 16-bit register that holds the address of the next instruction that the microprocessor will be fetching from memory.

Stack Pointer Holds the 16-bit address of a special section in memory known as the stack. It is also used to save return address when a subroutine call is given.

B,C,D,E,H and L These are general purpose 8-bit registers.

Some instructions allow these general purpose registers to be grouped as register pairs so that they can hold 16-bit data words. The pairings allowed are BC, DE and HL.

Instruction Register Used to store the instruction that has been fetched from memory while the decoding circuitry determines which operation is to be performed. *The instruction register is automatically used by the CPU during each instruction cycle and the programmer has no access to this register.*

Register F The flag register. This register is a collection of flip-flops that are used to indicate the result of an operation of the accumulator. In general, each instruction will affect all, some or none of the flags.

The 8085 flags register is shown below:

Data Transfer Group Instructions in this group move data from register to register, memory to a register, or a register to memory. Data transferred can be 8 - 16 bits in length. An examples of such instructions is:

```
MOV d,s    %copy contents of s (source) to d (destination)
```

Arithmetic Group This group includes all 8 and 16-bit mathematical commands such as add, subtract, increment and decrement e.g.

```
ADD r      %adds contents of register r to A, result stored in A
INCR r     %increment contents of register r by 1 i.e. r = r + 1
```

The Logical and Rotate group Includes the logical operators AND, OR, EXCLUSIVE-OR and rotate instructions e.g.

```
AND r      %A AND contents of register r, bit by bit, result in A
RLC        %Rotate contents of accumulator left by one bit position
```

The Branch Group Also referred to as the JUMP commands. These commands cause the control of the program to be transferred to a new address, breaking the normal sequential flow of a program (similar to GOTO command in BASIC). The jump command can be conditional or unconditional e.g.

```
JMP addr   %jump to address addr - unconditional jump
JZ addr    %jump to addr if Zero (Z) flag is set (conditional)
JNZ addr   %jump to addr if Zero (Z) flag is not set (conditional)
```

Stack, I/O and machine control group Includes instructions for maintaining the stack, reading from input ports, setting and clearing flags.

Generally speaking, a microprocessor works by FETCHING instructions from memory, DECODING the instructions and then EXECUTING these instructions. This is known as the FETCH, DECODE and EXECUTE cycle of the microprocessor, and it works as follows:

- i) Using the contents of the program counter as a memory address, FETCH the next instruction from memory into the instruction register.
- ii) Change the value in the instruction register and make it point to the next location in memory.
- iii) DECODE the instruction in the instruction register to determine which operation is to be performed.

- iv) EXECUTE the instruction, typically by fetching the data to be operated on (operand data) and performing the specified operation on that data.

The sequencing is all done automatically by the microprocessor's control unit - all the programmer has to do is to put the list of instructions in memory (program) and set the program counter to point to the first one. Having performed one FETCH/DECODE/EXECUTE cycle, the microprocessor simply repeats this process for ever - *this is all a microprocessor ever does*.

10.7 Memory Interfacing

This involves the design of a circuit to map the address space of a memory component onto a microprocessor's address space. The circuit can be designed for *full address decoding* or *partial address decoding*. This circuit, which is essentially a decoder, can be implemented using a decoder, a ROM or "random logic" (using logic gates). (The use of "random logic" in memory interfacing results in the use of a large number of logic gates and it is rarely used).

10.7.1 Memory map (Address map)

Before we talk about address decoding, let us look at a memory map of a microprocessor. A memory map (also known as an address map) maps the storage locations in particular memory devices to addresses in the address space of the microprocessor. Input/Output devices may also be included in the map.

Each entry maps a subrange of addresses to the storage locations in a particular device.

Some of the address space may be unused.

An example of a memory map for a microprocessor using 16-bit address inputs is shown on Table 10.1.

Example

A microprocessor generates a 16-bit address A_{15} (MSB), $A_{14}, A_{13} \cdots A_1 A_0$ (LSB), and a $READ/\overline{WRITE}$ signal R/\overline{W} . The processor has an 8-bit data bus. $8K \times 8$ of ROM is to be placed in consecutive memory locations of the microprocessor, starting from 0000h. $18K \times 8$ RAM is to be placed in consecutive memory locations starting at A000h, while $4K \times 8$ ROM is to be placed in consecutive memory locations starting from F000h. The rest of the address space is unused. Draw a memory map for the address space

Solution

$8K \times 8$ ROM takes $8 \times 1024 = 8192$ memory locations.

Table 10.1:

ADDRESS RANGE	DEVICE
0000h to 0FFFh	4K ROM
1000h to 3FFFh	Unused
4000h to 5FFFh	8K RAM
6000h to 7FFFh	8K EPROM
8000h to 9FFFh	Unused
A000h to C7FFh	10K RAM
C800h to DFFFh	Unused
F000h to FFFFh	I/O devices

$8192_{10} = 2000h$ hence the ROM takes address locations 0000h to 1FFFh on the memory map of the microprocessor.

$18K = 18 \times 1024 = 18432$ memory locations = 4800h. Hence the RAM takes address locations A000h to E7FFh

$4K = 4 \times 1024 = 4096$ memory locations = 1000h. hence the ROM takes address locations F000h to FFFFh

The memory map is shown on Table 10.2.

Table 10.2:

ADDRESS RANGE	DEVICE
0000h to 1FFFh	4K ROM
2000h to 9FFFh	Unused
A000h to E7FFh	18K RAM
E800h to EFFFh	Unused
F000h to FFFFh	10K ROM

10.7.2 Partial Address Decoding

Partial address decoding is so called because all the address lines available for decoding do not take part in the decoding process. Partial address decoding is the simplest, and consequently the most inexpensive form of address decoding to implement. However, this type of decoding has a major disadvantage in that it prevents full use of the microprocessor's available memory space and produces difficulties when expanding the memory system at a later date. This procedure is illustrated by the example below:

Example

A microprocessor generates a 16-bit address A_{15} (MSB), $A_{14}, A_{13} \cdots A_1 A_0$ (LSB), and

a $READ/\overline{WRITE}$ signal R/\overline{W} . The processor has an 8-bit data bus. $1K \times 8$ ROM ICs are to be placed in consecutive memory locations from F000h to FFFFh. Each ROM IC has one active-LOW chip-select input \overline{CS} and an active-HIGH output enable OE . The rest of the address space is unused. Design a partial address decoder that uniquely decodes the ROM addresses.

Solution

ROM takes the address space F000h - FFFFh, which is 1000h memory locations.

$1000h = 4096_{10}$, hence the ROM takes 4096 address locations.

We are given that this ROM is to be made up using $1K \times 8$ ROM ICs. One $1K \times 8$ ROM IC has 1024 locations, hence we need 4 ROM ICs to fill up the address space F000h to FFFFh.

$1024_{10} = 400h$ hence we can draw a table as shown on Table 10.3, which shows how address locations F000h to FFFFh are shared by the 4 ROM ICs. The partial address decoder will then be the circuit that will uniquely select each of the 4 ROM ICs.

Table 10.3:

ADDRESS RANGE	DEVICE
F000h F3FFh	ROM IC1
F400h F7FFh	ROM IC2
F800h FBFFh	ROM IC3
FC00h FFFFh	ROM IC3

Looking at Table 10.3, we see that we can use the second-most-significant-digit to uniquely select the ROM ICs. Expanding this digit to binary, we get the information on Table 10.4.

From this table, we see that we can use A_{11} and A_{10} to select between the 4 ROM ICs.

Using a 74LS138 decoder, the decoding circuit can be implemented as shown in Figure 10.4.

The microprocessor in the above example generates a 16-bit address $A_{15}A_{14} \cdots A_1A_0$, but in the above example, we have only used twelve bits $A_{11}A_{10} \cdots A_0$ in the address decoding. The circuit of Figure 10.4 is therefore an example of partial address decoding as we have not used A_{15} , A_{14} , A_{13} and A_{12} . The addresses to select the ROMs range from $A_{11}A_{10}A_9A_8A_7A_6A_5A_4A_3A_2A_1A_0 = 000000000000$ to 111111111111 (000h to FFFh). However, since the microprocessor generates a 16-bit address and

Table 10.4:

A_{11}	A_{10}	A_9	A_8	DEVICE
0	0	0	0	ROM IC1
0	0	1	1	
0	1	0	0	ROM IC2
0	1	1	1	
1	0	0	0	ROM IC3
1	0	1	1	
1	1	0	0	ROM IC4
1	1	1	1	

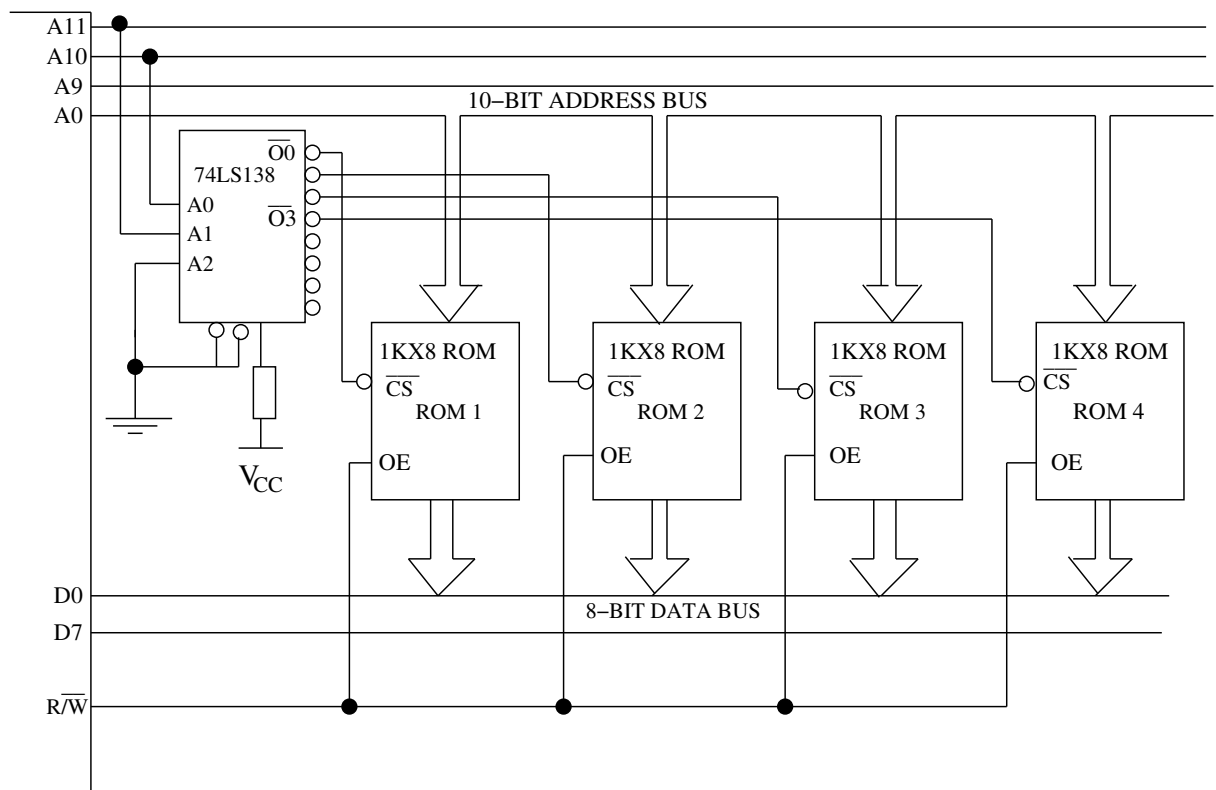


Figure 10.4: Partial Address Decoding using a decoder

we have not used address bits A_{15} , A_{14} , A_{13} and A_{12} , then we can see that all addresses issued by the microprocessor will select an address on the ROM, in other words, the ROM is mapped onto the entire address space of the microprocessor (e.g. addresses ending with 000h, like 0000h, 1000h, F000h e.t.c. are mapped onto the same address location). *This circuit will work well only if we do not intend to use the rest of the microprocessor's address space.*

10.7.3 Full Address Decoding

For full address decoding, we use all the address bits of the microprocessor and design the address decoding circuitry in such a way that each addressable location within a memory device will respond to a single address on the address bus. This way, we are able to use the whole of the address space of the microprocessor - there is no wasted space. This technique will be illustrated using the above example.

Looking at Table 10.3, we can see that the most significant digit for all the ROMs is Fh, hence, for all these ROM ICs, $A_{15}A_{14}A_{13}A_{12} = 1111$. To incorporate this condition address decoding, we use these inputs to enable the decoder only when $A_{15}A_{14}A_{13}A_{12} = 1111$ to get a full address decoding circuit, illustrated on Figure 10.5.

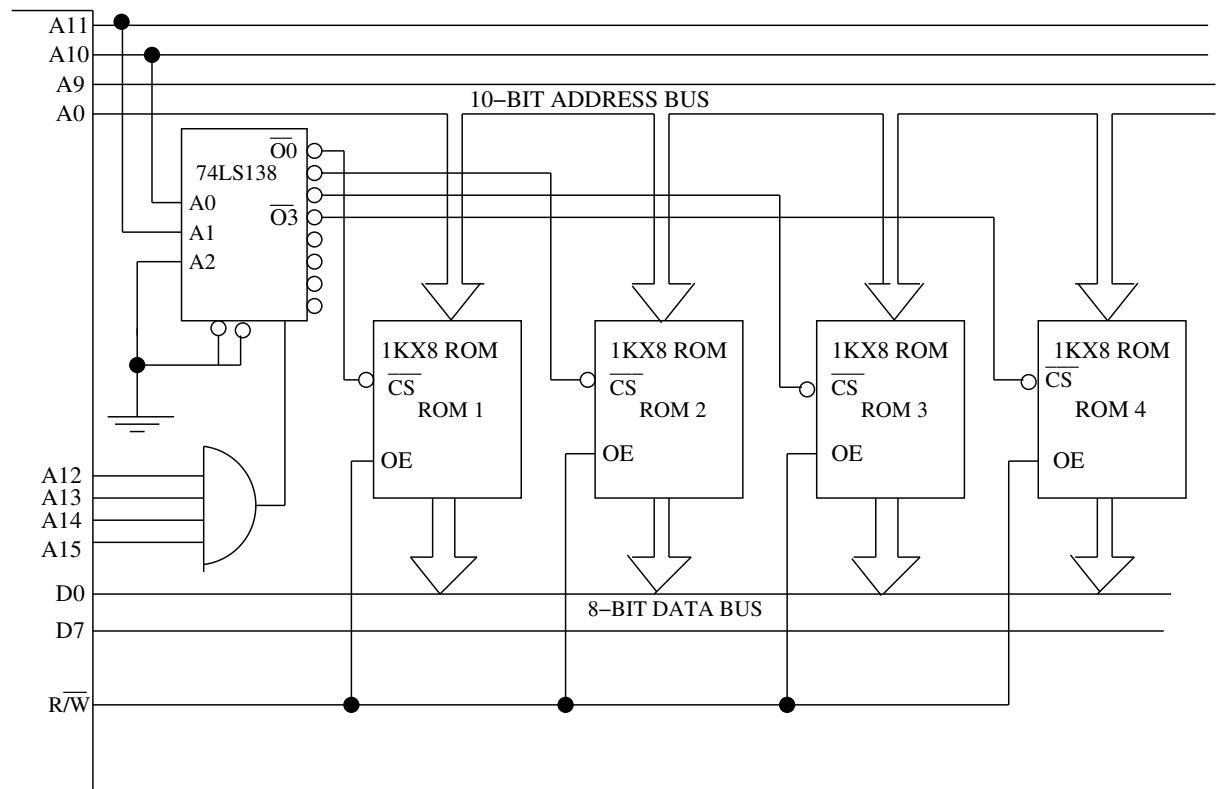


Figure 10.5: Full Address Decoding using a decoder

Since we have used all the address bits in this case, the ROMs can only be addressed using addresses F000h to FFFFh. We can therefore connect other devices on the rest of the microprocessor's address space.

It is also possible to carry out address decoding using a ROM. From Table 10.4, we saw that A_{11} and A_{10} can be used to select between the 4 ROM ICs. We use these two bits as the address of the ROM and the data we are to store at each address location is such that only one ROM IC is selected, as shown on Table 10.5

Table 10.5:

ADDRESS		DATA STORED				DEVICE
A_1	A_0	D_3	D_2	D_1	D_0	
0	0	0	1	1	1	ROM IC1
0	1	1	0	1	1	ROM IC2
1	0	1	1	0	1	ROM IC3
1	1	1	1	1	0	ROM IC4

We then use A_{15} , A_{14} , A_{13} and A_{12} to enable the ROM: connect them such that the ROM is enabled only when $A_{15}A_{14}A_{13}A_{12} = 1111$. The decoding circuit is then implemented as shown on Figure 10.6.

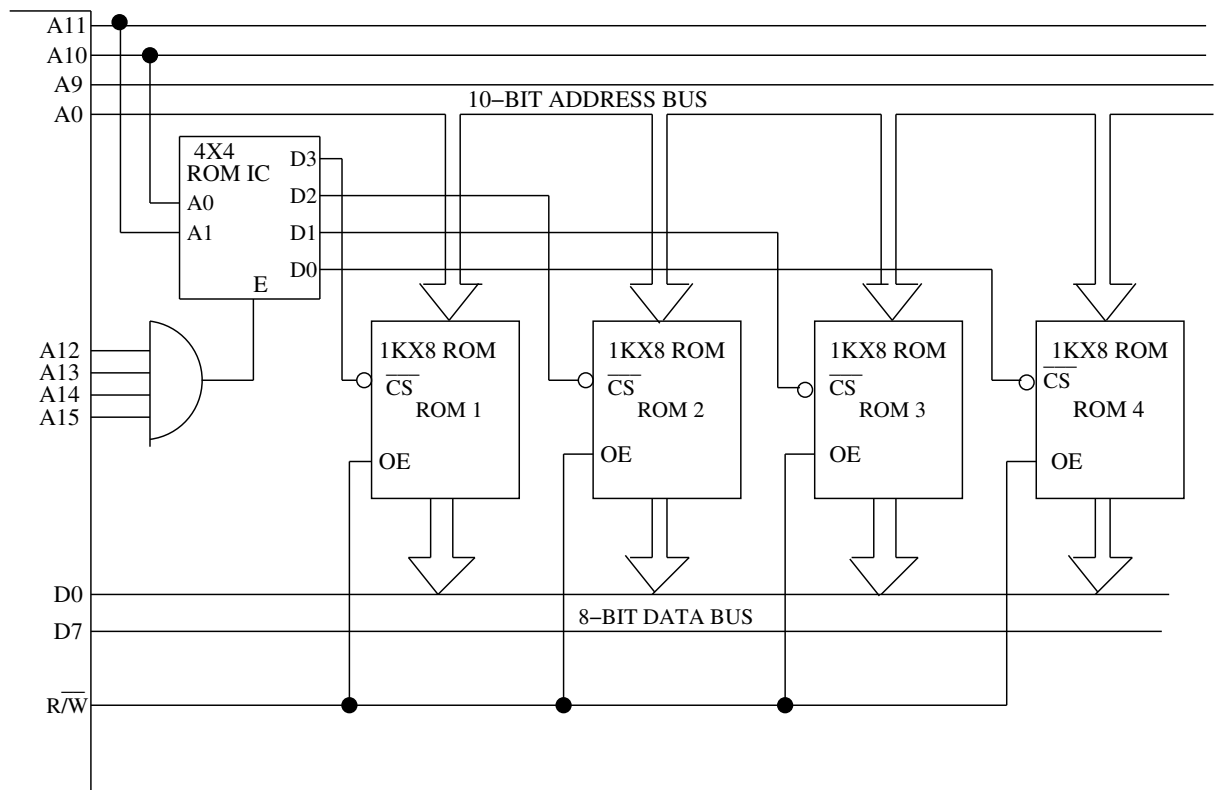


Figure 10.6: Address Decoding using a ROM

Bibliography

- [1] TOCCI, Ronald J., WIDMER, Neal S., MOSS, Gregory L., *Digital Systems, Principles and Applications*, 9th Edition, Prentice Hall International Inc. 2004.
- [2] JAIN R.P., *Modern Digital Electronics*, Tata McGraw-Hill Publishing Company 1984.
- [3] MANO M.M., *Digital Design*, 3rd Edition, Prentice Hall International Inc. 2002.
- [4] FLETCHER, *An Engineering Approach to Digital Design*
- [5] SANDIGE, *Modern Digital Design*
- [6] TAUB & SCHILLING, *Digital Integrated Electronics* McGraw-Hill Publishers.
- [7] BANNISTER & WHITEHEAD, *Fundamentals of Modern Digital Systems*
- [8] WILKINSON, *Digital System Design*
- [9] CLEMENTS, Alan, *Microprocessor Systems Design, 68000 Hardware, Software and Interfacing* PWS-KENT Publishing Company, 1987.

SK '05

*THIS DOCUMENT WAS PREPARED USING THE \LaTeX
WORD – PROCESSOR RUNNING ON A LINUX PLATFORM.
PICTURES DRAWN USING THE xfig PROGRAM.*