CHRISTIAN EMINENT COLLEGE, INDORE

(Academy of Management, Professional Education and Research)

An Autonomous Institution Accredited with 'A' Grade by NAAC

E-Content

on

"NUMBER SYSTEM"

Prepared By: Prof. C.K.Tiwari

Department of Computer Science & Electronics

UNIT-1: NUMBER SYSTEMS

- *Decimal, Binary, Octal And Hexa-Decimal Number System And Their Interconversion.*
- *Binary And Hexadecimal Addition, SubtractionAnd Multiplication*
- *1's And 2's Compliment Methods Of Addition/Subtraction*

OBJECTIVES

After going through this unit, you will be able to

- understand the decimal, binary, octal and hexadecimal number systems
- convert from one number system into another
- apply arithmetic operations to binary numbers

.

INTRODUCTION

The binary number system and digital codes are fundamental to computers. In this chapter, the binary number system and its relationship to other systems such as decimal, hexadecimal, and octal are introduced. Arithmetic operations with binary numbers are discussed to provide a basis for understanding how computers and many other types of digital systems work.

NUMBER SYSTEMS

A number system relates quantities and symbols. The base or radix of a number system represents the number of digits or basic symbols in that particular number system. In decimal system the base is 10, because of use the numbers 0, 1, 2,3,4,5,6,7,8 and 9.

Binary Number System

A binary number system is a code that uses only two basic symbols. The digits can be any two distinct characters, but it should be 0 or 1. The binary equivalent for some decimal numbers are given below

decimal	0	1	2	3	4	5	6	7	8	9	10	11
binary	0	1	10	11	100	101	110	111	1000	1001	1010	1011

Each digit in a binary number has a value or weight. The LSB has a value of 1.The second from the right has a value of 2, the next 4 , etc.,

16	8	4	2	1
2^4	2^3	2^2	2^1	2^0

Binary to decimal conversion:

$(1001)_2 = X_{10}$

$1001 = 1x2^3 + 0x2^2 + 0x2^1 + 1x2^0$

$= 8+0+0+1$

$(1001)_2 = (9)_{10}$

Fractions:

For fractions the weights of the digit positions are written from right of the binary point and weights are given as follows.

2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}

E.g.:

$(0.0110)_2 = X_{10}$

$= 0x2^{-1} + 1x2^{-2} + 1x2^{-3} + 0x2^{-4}$

$= 0 \text{ x}0.5 + 1\text{x}0.25 + 1\text{x}0.125 + 0\text{x}0.0625$

$= (0.375)_{10}$

E.g.:

$(1011.101)_2 = X_{10}$

$= 1x2^3 + 0x2^2 + 1x2^1 + 1x2^0 + 1x2^{-1} + 0x2^{-2} + 1x2^{-3}$

$= 8 + 0 + 2 + 1 + 0.5 + 0 + 0.125$

$= (11.625)_{10}$

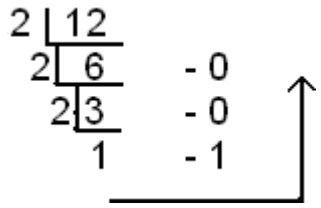
Decimal to Binary Conversion:

(Double Dabble method)

In this method the decimal number is divided by 2 progressively and the remainder is written after each division. Then the remainders are taken in the reverse order to form the binary number.

E.g.:

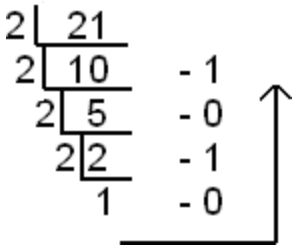
$(12)_{10} = X_2$



$(12)_{10} = (1100)_2$

E.g.:

$(21)_2 = X_2$



$(21)_2 = (10101)_2$

Fractions:

The fraction is multiplied by 2 and the carry in the integer position is written after each multiplication. Then they are written in the forward order to get the corresponding binary equivalent.

E.g.:

$(0.4375)_{10} = X_2$

2 x 0.4375 = 0.8750 => 0
2 x 0.8750 = 1.750 => 1
2 x 0. 750 = 1.5 => 1
2 x 0.5 = 1.0 => 1

$(0.4375)_{10} = (0.0111)_2$

Octal Number System

Octal number system has a base of 8 i.e., it has eight basic symbols. First eight decimal digits 0, 1,2,3,4,5,6,7 are used in this system.

Octal to Decimal Conversion:

In the octal number system each digit corresponds to the powers of 8. The weight of digital position in octal number is as follows

8^4	8^3	8^2	8^1	8^0	8^{-1}	8^{-2}	8^{-3}

To convert from octal to decimal multiply each octal digit by its weight and add the resulting products.

E.g.:

(48)8 = X10

48 = 4 x 81 + 7 x 80
= 32 + 7
= 39

(48)8 = (39)10

E.g.:

(22.34)8 = X10

22.34 = 2 x 81 + 2 x 80 + 3 x 8-1 + 4 x 8-2
=16 + 2 + 3 x 1/8 +4 x 1/64
= (18.4375)

(22.34)8 = (18.4375)10

Decimal to Octal Conversion:

Here the number is divided by 8 progressively and each time the remainder is written and finally the remainders are written in the reverse order to form the octal number. If the number has a fraction part, that part is multiplied by 8 and carry in the integer part is taken. Finally the carries are taken in the forward order.

E.g.:

$(19.11)_{10} = X_8$

$8\overline{)19}$
$2 - 3$

0.11 x 8 = 0.88 => 0
0.88 x 8 = 7.04 => 7
0.04 x 8 = 0.32 => 0
0.32 x 8 = 2.56 => 2
0.56 x 8 = 4.48 => 4

$(19.11)_{10} = (23.07024)_8$

Octal to Binary Conversion:

Since the base of octal number is 8, i.e., the third power of 2, each octal number is converted into its equivalent binary digit of length three.

E.g.:

$(57.127)_8 = X_2$

5	7	.	1	2	7
101	111	.	001	010	111

$(57.127)_8 = (101111001010111)_2$

Binary to Octal Conversion:

The given binary number is grouped into a group of 3 bits, starting at the octal point and each group is converted into its octal equivalent.

E.g.:

$(1101101.11101)_2 = X_8$

001	101	101.111	010
1	5	5 . 7	2

$(1101101.11101)_2 = (155.72)_8$

Hexadecimal Number System:

The Hexadecimal number system has a base of 16. It has 16 symbols from 0 through9 and A through F.

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

Binary to Hexadecimal:

The binary number is grouped into bits of 4 from the binary point then the corresponding hexadecimal equivalent is written.

E.g.:

$(100101110 . 11011)_2 = X_{16}$

0001 0010 1110 . 1101 1000
1 2 E . D 8

$(100101110 . 11011)_2 = (12E . D8)_{16}$

Hexadecimal to binary:

Since the base of hexadecimal number is 16, i.e., the fourth power of 2, each hexadecimal number is converted into its equivalent binary digit of length four.

E.g.:

$(5D. 2A)_{16} = X_2$

5 D . 2 A
0101 1101 . 0010 1010

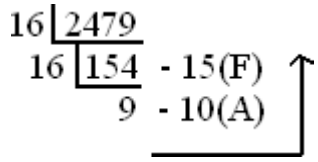
$(5D. 2A)_{16} = (01011101.00101010)_2$

Decimal to Hexadecimal:

The decimal number is divided by 16 and carries are taken after each division and then written in the reverse order. The fractional part is multiplied by 16 and carry is taken in the forward order.

E.g.:

$(2479.859))_{10} = X_{16}$



16 x 0.859 = 13.744 => 13 (D)
16 x 0.744 = 11.904 => 11 (B)
16 x 0.904 = 14.464 => 14 (E)
16 x 0.464 = 7.424 => 7
16 x 0.424 = 6.784 => 6

$(2479.859)_{10} = (9AF.DBE76)_{16}$

Hexadecimal to Decimal:

Each digit of the hexadecimal number is multiplied by its weight and then added.

E.g.:

$(81.21)_{16} = X_{10}$
$= 8 \times 16^1 + 1 \times 16^0 + 2 \times 16^{-1} + 1 \times 16^{-2}$
$= 8 \times 16 + 1 \times 1 + 2/16 + 1/16^2$
$= (129.1289)_{10}$

$(81.21)_{16} = (129.1289)_{10}$

Binary Arithmetic Opeartion

Binary Addition:

To perform the binary addition we have to follow the binary table given below.

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 0 => plus a carry-over of 1

Carry-overs are performed in the same manner as in decimal arithmetic. Since 1 is the largest digit in the binary system, any sum greater than 1 requires that a digit be considered over.

E.g.:

111	1010	11.01
+ 110	+ 1101	+ 101.11
1001	10111	1001.00

Binary Subtraction:

To perform the binary subtraction the following binary subtraction table should be followed.

0 – 0 = 0

1 – 0 = 1

1 – 1 = 0

0 – 1 = 1 with a borrow of 1 is equivalent to 10 – 1 = 1

E.g.:

111
010
101

E.g.:

110.01
100.10
001.11

1's complement:

To obtain 1's complement of a binary number each bit of the number is subtracted from 1.

E.g.:

Binary number	1's Complement
0101	1010
1001	0110
1101	0010
0001	1110

Thus 1's complement of a binary number is the number that results when we change each 0 to a 1 and each 1 to a 0.

1's complement subtraction:

Instead of subtracting the second number from the first, the 1's complement of the second number is added to the first number. The last carry which is said to be a END AROUND CARRY, is added to get the final result.

E.g.:

7 - 111 ----------> 111 +
3 011 1's complement 100
4 1011 +
└→1
100 ➐ result

If there is no carry in the 1's complement subtraction, it indicates that the result is a negative and number will be in its 1's complement form. So complement it to get the final result.

.:

8 - 1000 ----------> 1000 +
10 1010 1's complement 0101
4 1101 1's complement - 0010 ➐ result

The following points should be noted down when we do 1's complement subtraction.

1. Write the first number (minuend) as such.
2. Write the 1's complement of second number(subtrahend)
3. Add the two numbers.
4. The carry that arises from the addition is said to be "end around carry".
5. End-around carry should be added with the sum to get the result.
6. If there is no end around carry find out the 1's complement of the sum and put a negative sign before the result as the result is negative.

2's Complement:

2's complement results when we add '1' to 1's complement of the given number i.e.,
2's complement =1's complement + 1

Binary Number	1's complement	2's complement
1010	0101	0110
0101	1010	1011
1001	0110	0111
0001	1110	1111

2's Complement Subtraction:

Steps:

1. Write the first number as such
2. Write down the 2's complement of the second number.
3. Add the two numbers.
4. If there is a carry, discard it and the remaining part (sum) will be the result (positive).
5. If there is no carry, find out the 2's complement of the sum and put negative sign before the result as the result is negative.

E.g.:

1) 10 - 1010 ----------> 1010 +
8 1000 2's complement 1000
2 10010

0010 ➐ result

2) 5 - 0101 ----------> 0101 +
12 1100 2's complement 0100
4 1001 2's complement – 0111 ➐ result

Binary multiplication:

The table for binary multiplication is given below
0 x 0 = 0
0 x 1 = 0
1 x 0 = 0
1 x 1 = 1

E.g.:

1011 x 110

1011 x
110
0000
1011
1011
1000010

E.g.:

101.01 x 11.01

```
   101.01 x
  11.01101 01
   00000
  10101
 10101
10001.0001
```

Binary division:

The table for binary division is as follows.

0 ÷ 1 = 0

1 ÷ 1 = 1

As in the decimal system division by zero is meaning less.

.:

1) 1100 ÷ 11

```
     100
11 | 1100
   | 11
     0
```

2) 1001 ÷ 10

```
    100.1
10 | 1001
   | 10
     0010
       10
        0
```

BCD Addition

Binary Coded Decimal(BCD) is a way to express each of the decimal digits with a binary code. There are only ten code groups in the BCD system. The 8421 code is a type of BCD code. In BCD each decimal digit , 0 through 9 is represented by a binary code of four bits. The designation of 8421 indicates the binary weights of the four bits (2^3,2^2,2^1,2^0). The largest 4-bit code is 1001. The numbers 1010, 1011, 1100, 1101, 1110, and 1111 are called forbidden numbers. The following table represents the decimal and 8421 equivalent numbers.

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

8421 Addition:

In 8421 addition, if there is a carry or if it results in a forbidden group, then 0110(6) should be added in order to bring the result to the 8421 mode again.

E.g.:

8	+	1000 +
7		0111
15		1111
		+ 0110
		0001 0101

E.g.:

18	+	0001 1000 +
2		0000 0010
20		0001 1010
	+	0000 0110
		0010 0000

Alphanumeric code

Computers, printers and the other devices must process both alphabetic and numeric information. Serial coding systems have been developed to represent alphanumeric information as a series of 1's and 0's. The characters to be coded are alphabets(26), numerals (10) and special characters such as +,-, /,*, $ etc,

In order to code a character, string of binary digits is used. In order to ensure uniformity in coding, two standard codes have been used.

1. ASCII: American Standard Code for Information Interchange.
2. EBCDIC: Extended Binary Coded Decimal Interchange Code. It is an 8 bit code.

ASCII is 7-bit code of the form X_6, X_5, X_4, X_3, X_2, X_1, X_0 and is used to code two types of information. One type is the printable character such as alphabets, digits and special characters. The other type is known as control characters which represent the coded information to control the operation of the digital computer and are not printed.

CHECK YOUR PROGRESS 1

1. $2 \times 10^1 + 8 \times 10^0$ is equal to
(a) 10 (b) 280 (c) 2.8 (d) 28

2. The binary number 1101 is equal to the decimal number
(a) 13 (b) 49 (c) 11 (d) 3

3. The decimal 17 is equal to the binary number
(a) 10010 (b) 11000 (c) 10001 (d) 01001

4. The sum of 11010 + 01111 equals
(a) 101001 (b) 101010 (c) 110101 (d) 101000

Unit 2: LOGIC GATES

AND Gate

A gate is simply an electronic circuit which operates a one or more signals to produce an output signal. The output is high only for certain combination of input signals.

An AND gate (Figure 1.1) has a high output only when all inputs are high. The output is low when any one input is low.

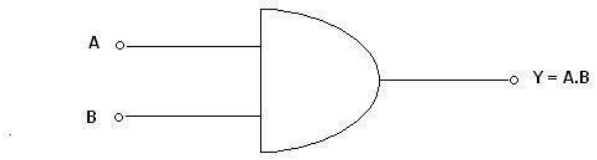


Figure 1.1 AND gate

Boolean expression for AND gate operation is

$$Y=A \cdot B$$

Truth table

A	B	Y = A . B
0	0	0
0	1	0
1	0	0
1	1	1

OR gate

An OR gate (Figure 1.2) produces a high output when any or the entire inputs are high. The output is low only when all the inputs are low.

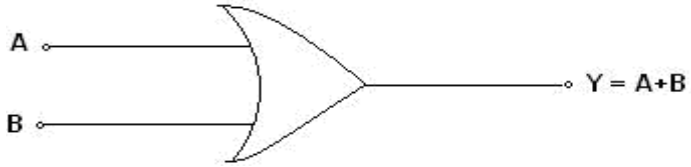


Figure 1.2 OR gate

The Boolean expression for an OR gate is

$$Y=A+B$$

Truth table:

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

NOT gate:

A NOT gate (Figure 1.3) is also called an inverter. The circuit has one input and one output. The output is the complement of the input. If the input signal is high, the output is low and vice versa.

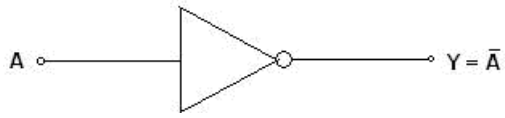


Figure 1.3 NOT gate

The Boolean expression for NOT gate is

$$Y = \bar{A}$$

Truth table:

A	Y = Ā
0	1
1	0

If two NOT gates are cascaded then the output will be same as the input and the circuit is called buffer circuit.

NAND gate

A NAND (Figure 1.4) gate has two or more input signals but only one output signal. All input signals must be high to get a low output. When one AND gate is combined with a NOT gate, a NAND gate is obtained.

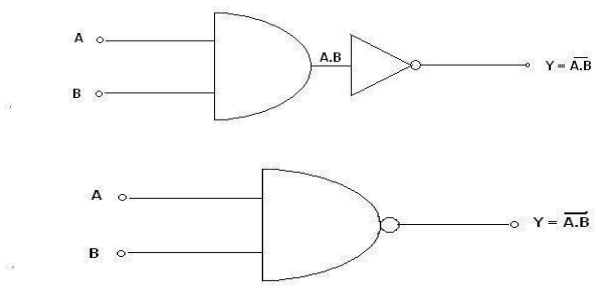


Figure 1.4 NAND gate

Truth table:

A	B	Y = A . B
0	0	1
0	1	1
1	0	1
1	1	0

NOR gate:

NOR gate (Fig. 1.5) has two or more input signals and one output signal. It consists of one OR gate followed by an inverter. A NOR gate produces a high output only when all the inputs are low.

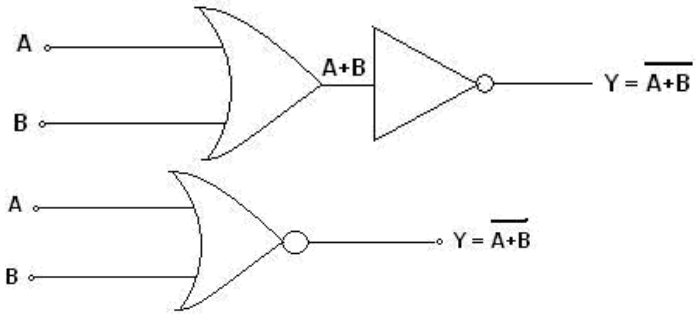


Figure 1.5 NOR gate

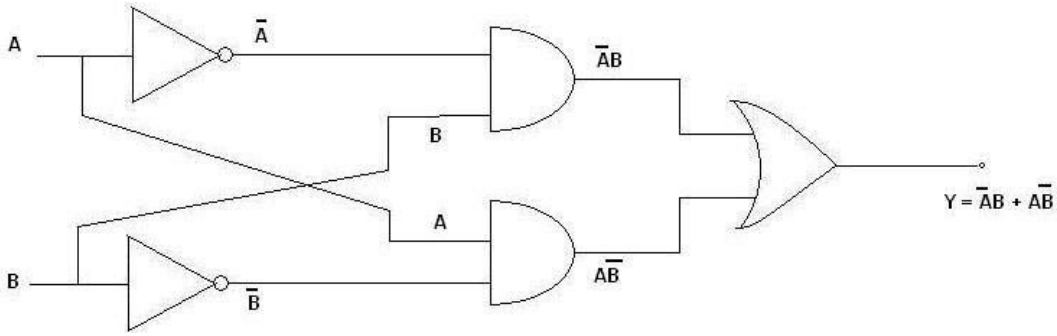
Truth table:

A	B	Y = $\overline{A + B}$
0	0	1
0	1	0
1	0	0
1	1	0

XOR gate

XOR (Figure 1.6) gate is an abbreviation of exclusive OR gate. It has two inputs and one output. For a two input XOR gate, the output is high when the inputs are different and the output is low when the inputs are same. In general, the output of an XOR gate is high when the number of its high inputs is odd. The Boolean expression of the XOR gate is

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



a) Logic diagram

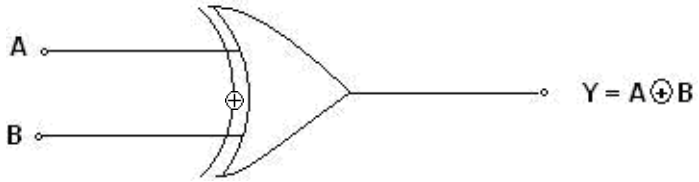


Figure 1.6 XOR gate

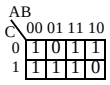
A	B	Y = A B
0	0	0
0	1	1
1	0	1
1	1	0

Truth table

Karnaugh map

- **Karnaugh map (K-map) allows viewing the function in a picture form**
- **Containing the same information as a truth table**
- **But terms are arranged such that two neighbors differ in only one variable**
- **It is easy to identify which terms can be combined**
- **Example:**

  A map with 3 variables

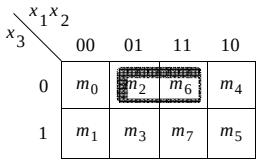


A	B	C	F
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Location of Min-terms in K-maps

x_1	x_2	x_3	
0	0	0	m_0
0	0	1	m_1
0	1	0	m_2
0	1	1	m_3
1	0	0	m_4
1	0	1	m_5
1	1	0	m_6
1	1	1	m_7

(a) Truth table



(b) Karnaugh map

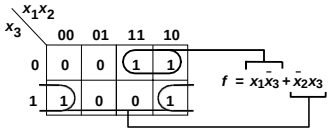
$\mathbf{m_2 + m_6 = x_1' x_2 x_3' + x_1 x_2 x_3'}$
$\mathbf{= x_2 x_3'}$

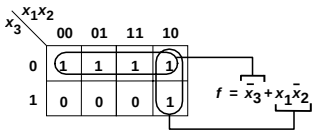
Simplification using K-map

- **Groups of '1's of size 1x1, 2x1, 1x2, 2x2, 4x1, 1x4, 4x2, 2x4, or 4x4 are called prime implicants (p.159 in textbook).**

- **A '1' in the K-map can be used by more than one group**
- **Some rule-of-thumb in selecting groups:**
  - **Try to use as few group as possible to cover all '1's.**
  - **For each group, try to make it as large as you can (i.e., if you can use a 2x2, don't use a 2x1 even if that is enough).**

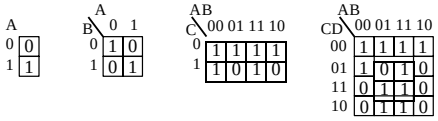
Examples of 3-Variable K-map





Karnaugh maps with up to 4 variables

- **Example: 1, 2, 3, and 4 variables maps are shown below**



- **What if a function has 5 variables?**

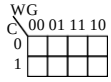
Farmer's example and truth tables

Farmer's example truth tables and K-maps

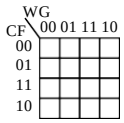
- With three variables, we do not want W and G or G and C to be equal (both 0 or both 1) at the same time
- With four variables, it is not a problem if F is also the same

W	G	C	A1
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

A1



A2



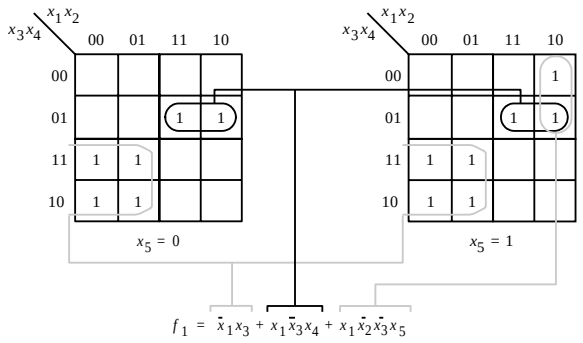
A1 =

A2 =

W	G	C	F	A2
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

7

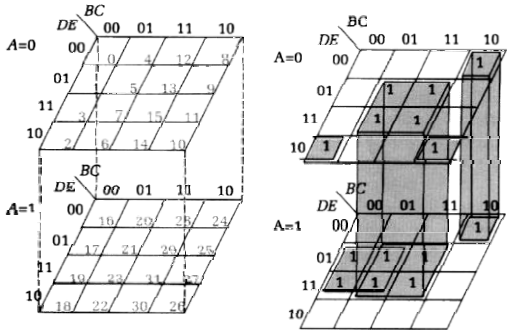
K-map for 5-variables functions



8

K-map for 5-variables functions

F(A,B,C,D,E) = Σ m(2,5,7,8,10,13,15,17,19,21,23,24,29,31)
F(A,B,C,D,E) = CE + AB'E + BC'D'E' + A'C'DE'

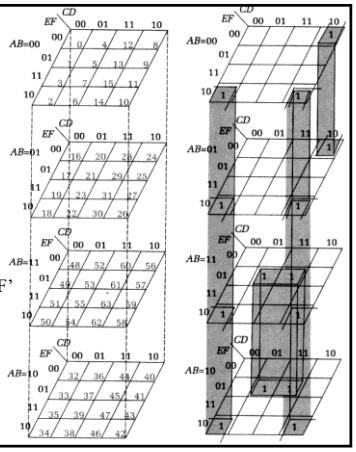


9

K-map for 6-variable functions

G(A,B,C,D,E,F)
= Σ m(2,8,10,18,24,26,34, 37,42,45,50,53,58,61)

G(A,B,C,D,E,F)
= D'EF' + ADE'F + A'CD'F'



Lec 9

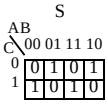
K-map Example for Adder functions

A	B	C	S	Cout
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

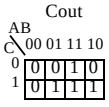
$S(A, B, C) = \sum m(1,2,4,7)$

$Cout(A, B, C) = \sum m(3,5,6,7)$

S



Cout

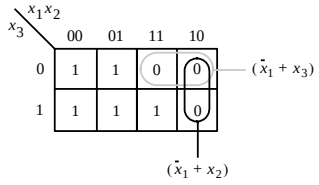


S = A'B'C + A'BC' + AB'C' + ABC

Cout = BC + AC + AB

11

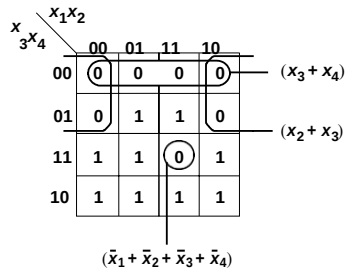
Minimization of POS Forms



POS minimization of $f = \Pi\, M(4, 5, 6)$

12

Minimization of 4-Var. Function in POS Form



POS minimization of $f = \Pi\, M(0, 1, 4, 8, 9, 12, 15)$

13

7-segment display

14

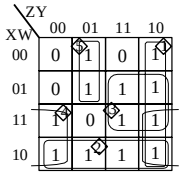
Simplification of 'g' in 7-segment display

g= Z'YX'W' +ZY'X'W'
+Z'YX'W +ZYX'W +ZY'X'W
+ Z'Y'XW +ZYXW +ZY'XW
+ Z'Y'XW' +Z'YXW' +ZYXW' +ZY'XW'

XW \ ZY	00	01	11	10
00	0	1	0	1
01	0	1	1	1
11	1	0	1	1
10	1	1	1	1

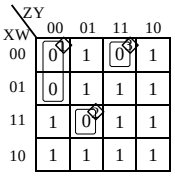
15

Minimization of Product-of-Sums Forms



g=ZY' ①
+XW' ②
+ZW ③
+Y'X ④
+Z'YX' ⑤

Cost = 22
(5 AND gates,
1 OR gates
16 inputs)



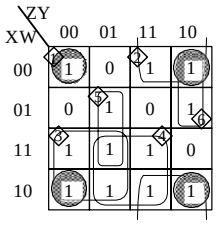
g=(Z+Y+X) ①
.(Z+Y'+X'+W') ②
.(Z'+Y'+X+W) ③

Cost = 18
(3 OR gates,
1 AND gates
14 inputs)

16

Simplification of 'a' in 7-segment display

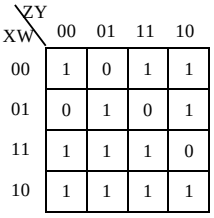
a=Z'Y'X'W' +ZYX'W' +ZY'X'W'
+Z'YX'W +ZY'X'W
+ Z'Y'XW +Z'YXW +ZYXW
+ Z'Y'XW' +Z'YXW' +ZYXW' +ZY'XW'



a=Y'W' ①
+ZW' ②
+Z'X ③
+YX ④
+Z'YW ⑤
+ZY'X' ⑥

17

Simplification of 'a' in POS Form



SOP: a=Y'W'
+ZW'
+Z'X
+YX
+Z'YW
+ZY'X'

Cost =
(AND gates,
OR gates
inputs)

POS:

Cost =
(AND gates,
OR gates
inputs)

18

K-map with Don't Care Conditions

- ***Don't care condition*** **is input combination that will never occur.**
- **So the corresponding output can either be 0 or 1.**
- **This can be used to help simplifying logic functions.**
- **Example: F(A,B,C,D)=Σ m(1,3,7,11,15)+ Σ D(0,2,5)**

AB \ CD	00	01	11	10
00	d	1	1	d
01	0	d	1	0
11	0	0	1	0
10	0	0	1	0

F = CD+A'B'

AB \ CD	00	01	11	10
00	d	1	1	d
01	0	d	1	0
11	0	0	1	0
10	0	0	1	0

F = CD+A'D

d: Don't Care Condition

19

Examples

- **Simplify the following function considering:**
  - **the sum-of-products form** **-- the product-of-sums form**

AB \ CD	00	01	11	10
00	1	0	0	1
01	1	1	1	d
11	0	d	1	1
10	0	0	0	1

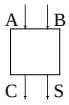
AB \ CD	00	01	11	10
00	1	0	0	1
01	1	1	1	d
11	0	d	1	1
10	0	0	0	1

20

1-bit building blocks to make n-bit circuit

- **Design a 1-bit circuit with proper "glue logic" to use it for n-bits**
  - **It is called a bit slice**
  - **The basic idea of bit slicing is to design a 1-bit circuit and then piece together n of these to get an n-bit component**
- **Example:**
- **A half-adder adds two 1-bit inputs**
- **Two half adders can be used to add 3 bits**
- **A 3-adder is a full adder**
- **A full adder can be a bit slice to construct an n-bit adder**

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

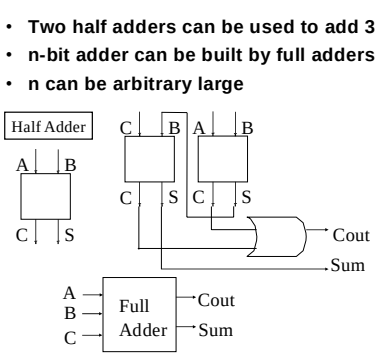


Half Adder

21

Full adder & multi-bit ripple-carry adder

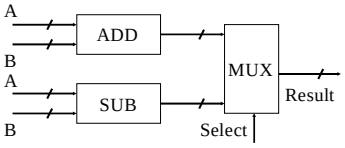
- **Two half adders can be used to add 3 bits**
- **n-bit adder can be built by full adders**
- **n can be arbitrary large**



22

Multiple Function Unit Design

- **Design a unit that can do more than one function**
- **In that case, we can design a function unit for each operation like ADD, SUB, AND, OR, ...**
- **And then select the desired output**
- **For example, if we want to be able to perform ADD and SUB on two given operands A and B, and select any one**
- **Then the following set up will work**

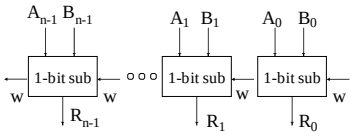


23

Design of a SUB Unit

- **An n-bit subtract unit can be designed in the same way as an n-bit adder**
- **One bit subtract unit: It has two inputs A and B (B is subtracted from A) and a borrow (w)**
- **Outputs: R (primary), W borrow (cascading)**
- **Borrow from one stage is fed to the next stage**
- **Truth table is shown**
- **An n-bit subtract circuit is shown**

A	B	w (in)	R	w(out)
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		



24

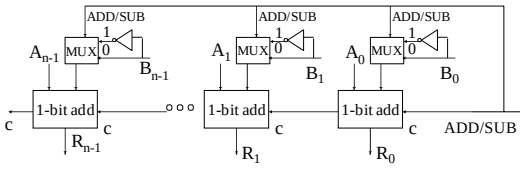
ADD/SUB unit design – First Idea

- Separate ADD and SUB units are expensive
- We can simplify the design
- A - B is the same as adding negation of B to A
- How to negate?
  - 2's complement (i.e., take 1's complement and add 1)
  - Adding 1 is also expensive
  - It needs an n-bit adder in general
  - However, we only need to add two bits in each stage
    - In the first stage, we need to add 1's complement of LSB and 1
    - In other stages, we need to add carry output of previous bit to 1's complement of current bit
- We select B or negation of B depending on the requirement
- We add A to the selected input to obtain the result

25

ADD/SUB unit design – Better Idea

- 2's complement generation of B is expensive
- We can take 1's complement of B and add it to A
- Then we need to add 1 to the result
- This can be done by setting input carry=1 to n-bit adder
- For add , we simply add B to A with input carry = 0
- Selection signal ADD/SUB = 0 for ADD and 1 for SUB
- The block diagram is shown below
- MUX and inverter can be replaced by XOR (B, ADD/SUB)



26

ADD/SUB unit design – Better Idea Example

Example: Find A-B
A=0101, B=0110

1's comp. of B (i.e., neg (B)) =

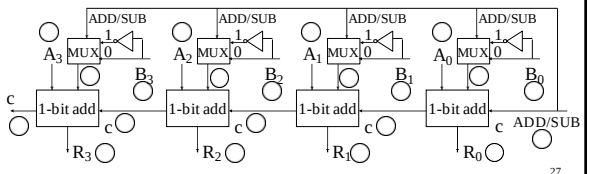
2's comp. of B = =

First Idea:
A+(comp(B)+1)

0101
+ 2's

Better Idea:
A+comp(B)+**1**

1
0101
+ 1's



27

ADD/SUB unit design – Better Idea Operation

Example: Find A-B
A=0101, B=0110

1's comp. of B (i.e., neg (B)) =1001

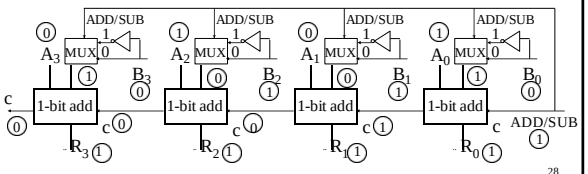
2's comp. of B =1001+1=1010

First Idea:
A+(neg(B)+1)

0000
0101
+ 1010 ← 2's
1111

Better Idea:
A+neg(B)+**1**

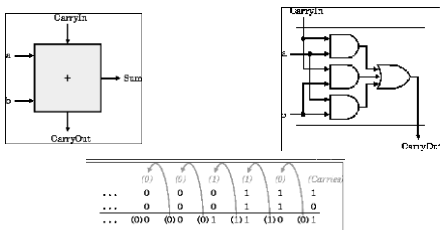
0001**1**
0101
+ 1001 ← 1's
1111



28

One-Bit Adder

- Takes three input bits and generates two output bits
- Multiple bits can be cascaded



29

Adder Boolean Algebra

A	B	CI	CO	S	
0	0	0	0	0	
0	0	1	0	1	
0	1	0	0	1	C = A.B + A.CI+ B.CI
0	1	1	1	0	
1	0	0	0	1	S = A.B.CI + A'.B'.CI+A'.B.CI'+A.B'.CI'
1	0	1	1	0	
1	1	0	1	0	
1	1	1	1	1	

30

Detecting Overflow

- No overflow when adding a positive and a negative number
- No overflow when signs are the same for subtraction
- Overflow occurs when the value affects the sign:
  - overflow when adding two positives yields a negative
  - or, adding two negatives gives a positive
  - or, subtract a negative from a positive and get a negative
  - or, subtract a positive from a negative and get a positive
- Consider the operations A + B, and A – B
  - Can overflow occur if B is 0 ?
  - Can overflow occur if A is 0 ?

Problem: ripple carry adder is slow

- Is a 32-bit ALU as fast as a 1-bit ALU?
- Is there more than one way to do addition?
  - two extremes: ripple carry and sum-of-products

Can you see the ripple? How could you get rid of it?

$c_1 = b_0c_0 + a_0c_0 + a_0b_0$
$c_2 = b_1c_1 + a_1c_1 + a_1b_1 \quad c_2 =$
$c_3 = b_2c_2 + a_2c_2 + a_2b_2 \quad c_3 =$
$c_4 = b_3c_3 + a_3c_3 + a_3b_3 \quad c_4 =$

Not feasible! Why?

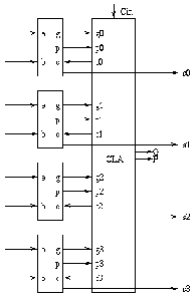
Carry-look-ahead adder

- An approach in-between our two extremes
- Motivation:
  - If we didn't know the value of carry-in, what could we do?
  - When would we always generate a carry? $g_i = a_i\, b_i$
  - When would we propagate the carry? $p_i = a_i + b_i$
- Did we get rid of the ripple?

$c_1 = g_0 + p_0c_0$
$c_2 = g_1 + p_1c_1 \qquad c_2 = g_1 + p_1g_0 + p_1p_0c_0$
$c_3 = g_2 + p_2c_2 \qquad c_3 = g_2 + p_2g_1 + p_2p_1g_0 + p_2p_1p_0c_0$
$c_4 = g_3 + p_3c_3 \qquad c_4 = g_3 + p_3g_2 + p_3p_2g_1 + p_3p_2p_1g_0 + p_3p_2p_1p_0c_0$

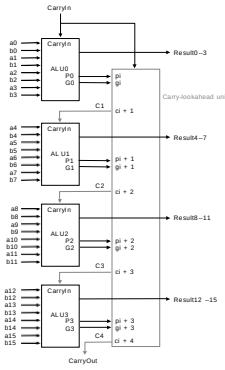
Feasible! Why?

A 4-bit carry look-ahead adder



- Generate g and p term for each bit
- Use g's, p's and carry in to generate all C's
- Also use them to generate block G and P
- CLA principle can be used recursively

Use principle to build bigger adders



- A 16 bit adder uses four 4-bit adders
- It takes block g and p terms and cin to generate block carry bits out
- Block carries are used to generate bit carries
  - could use ripple carry of 4-bit CLA adders
  - Better: use the CLA principle again!

Delays in carry look-ahead adders

- 4-Bit case
  - Generation of g and p: 1 gate delay
  - Generation of carries (and G and P): 2 more gate delay
  - Generation of sum: 1 more gate delay
- 16-Bit case
  - Generation of g and p: 1 gate delay
  - Generation of block G and P: 2 more gate delay
  - Generation of block carries: 2 more gate delay
  - Generation of bit carries: 2 more gate delay
  - Generation of sum: 1 more gate delay
- 64-Bit case
  - 12 gate delays

BOOLEAN ALGEBRA

Basic Laws of Boolean Algebra

Commutative law:

$A + B = B + A$
$B + A = A + B$

Associative law:

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

Distributive law

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

Other laws of Boolean algebra:

1. $A + 0 = A$
2. $A + 1 = 1$
3. $A + A = A$
4. $A + \bar{A} = 1$
5. $A .0 = 0$
6. $A .1 = A$
7. $A .A = A$
8. $A . \bar{A} = 0$
   =
9. $A = A$
10. $A + A.B = A$
11. $A.(A + B) = A$
12. $(A + B).(A+C) = A + B.C$
13. $A + \bar{A}.B = A + B$
14. $A.(\bar{A} + B) = A.B$
15. $(A + B).(\bar{A} + C) = A.C + \bar{A}.B$
16. $(A + C).(\bar{A} + B) = A.B + \bar{A}.C$

De Morgan's Theorems:

I Theorem statement:

The complement of a sum is equal to the product of the complements.

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

II Theorem Statement:

The complement of a product is equal to the sum of the complements.

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

Proof of first theorem:

To prove $\overline{A + B} = \bar{A} . \bar{B}$

Case 1: A=0, B=0

L.H.S => $\overline{A + B} = \overline{0 + 0} = \bar{0} = 1$

R.H.S => $\bar{A} . \bar{B} = \bar{0} . \bar{0} = 1 . 1 = 1$

Case 2: A=0, B=1

L.H.S => $\overline{A + B} = \overline{0 + 1} = \bar{1} = 0$

R.H.S => $\bar{A} . \bar{B} = \bar{0} . \bar{1} = 1 . 0 = 0$

Case 3: A=1, B=0

L.H.S => $\overline{A + B} = \overline{1 + 0} = \bar{1} = 0$

R.H.S => $\bar{A} . \bar{B} = \bar{1} . \bar{0} = 0 . 1 = 0$

Case 4: A=1, B=1

L.H.S => $\overline{A + B} = \overline{1 + 1} = \bar{1} = 0$

R.H.S => $\bar{A} . \bar{B} = \bar{1} . \bar{1} = 0 . 0 = 0$

Truth table

A	B	$\overline{\bar{A} + B}$	$\bar{A} . \bar{B}$
0	0	1	1
0	1	0	0
1	0	0	0
1	1	0	0

Proof of second theorem:

To prove $\overline{A . B} = \bar{A} + \bar{B}$

Case 1: A=0, B=0

L.H.S => $\overline{A . B} = \overline{0 . 0} = \overline{0} = 1$

R.H.S => $A + B = \bar{0} + \bar{0} = 1 + 1 = 1$

Case 2: A=0, B=1

L.H.S => $\overline{A . B} = \overline{0 . 1} = \overline{0} = 1$

R.H.S => $\bar{A} + \bar{B} = \bar{0} + \bar{1} = 1 + 0 = 1$

Case 3: A=1, B=0

L.H.S => $\overline{A . B} = \overline{1 . 0} = \overline{0} = 1$

R.H.S => $\bar{A} + \bar{B} = \bar{1} + \bar{0} = 0 + 1 = 1$

Case 4: A=1, B=1

L.H.S => $\overline{A . B} = \overline{1 . 1} = \overline{1} = 0$

R.H.S => $\bar{A} + \bar{B} = \bar{1} + \bar{1} = 0 + 0 = 0$

Truth table

A	B	$\overline{\bar{A} \cdot B}$	$\bar{A} + \bar{B}$
0	0	1	1
0	1	1	1
1	0	1	1
1	1	0	0

CHECK YOUR PROGRESS 2

1. An inverter performs an operation known as
(a) Complementation (b) assertion
(c) Inversion (d) both answers (a) and (c)

2.The output of gate is LOW when at least one of its inputs is HIGH. It is true for
(a) AND (b) NAND (c) OR (d) NOR

3.The output of gate is HIGH when at least one of its inputs is LOW. It is true for
(a) AND (b) OR (c) NAND (d) NOR

4.The output of a gate is HIGH if and only if all its inputs are HIGH. It is true for
(a)XOR (b) AND (c) OR (d) NAND

5. The output of a gate is LOW if and only if all its inputs are HIGH. It is true for
(a)AND (b) XNOR (c) NOR (d) NAND

6.Which of the following gates cannot be used as an inverter?
(a)NAND (b) AND (c) NOR (d) None of the above

7.The complement of a variable is always
(a) 0 (b) 1 (c) equal to the variable (d) the inverse of the variable

8.Which one of the following is not a valid rule of Boolean algebra?
(a) $A + 1 = 1$ (b) $A = \bar{\bar{A}}$ (c) $A.A = A$ (d) $A + 0 = A$

9. Which of the following rules states that if one input of an AND gate is always 1 , the output is equal to the other input ?
(a) $A + 1 = 1$ (b) $A + A = A$ (c) $A.A = A$ (d) $A \cdot 1 = A$

SUMMARY

- A binary number is a weighted number in which the weight of each whole number digit is a positive power of 2 and the weight of each fractional digit is a negative power of 2.
- The 1's complement of a binary number is derived by changing 1s to 0s and 0s to 1s
- The 2's complement of a binary number can be derived by adding 1 to the 1's complement.
- The octal number system consists of eight digits, 0 through 7.
- The hexadecimal number system consists of 16 digits and characters, 0 through 9 followed by A through F.
- The ASCII is a 7-bit alphanumeric code that is widely used in computer systems for input/output of information.
- The output of an inverter is the complement of its input
- The output of an AND gate is high only if all the inputs are high
- The output of an OR gate is high if any of the inputs is high
- The output of an NOR gate is low if any of the inputs is high
- The output of an NAND gate is low only if all the inputs are high
- The output of an exclusive-OR gate is high when the inputs are not the same

Multiplexer and Demultiplexer

A multiplexer is a circuit that accept many input but give only one output. A demultiplexer function exactly in the reverse of a multiplexer, that is a demultiplexer accepts only one input and gives many outputs. Generally multiplexer and demultiplexer are used together, because of the communication systems are bi directional.

Mutliplexer:

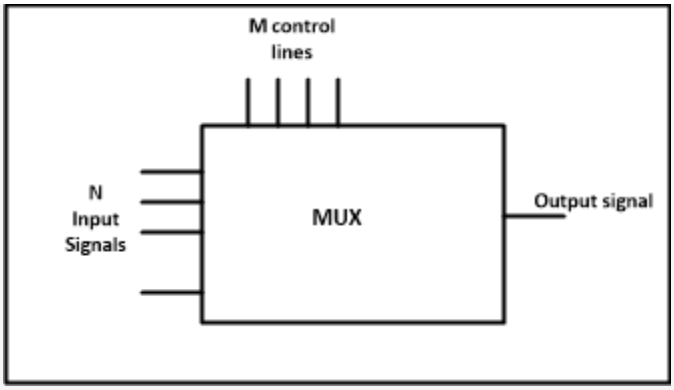
Multiplexer means many into one. A multiplexer is a circuit used to select and route any one of the several input signals to a signal output. An simple example of an non electronic circuit of a multiplexer is a single pole multiposition switch.

Multiposition switches are widely used in many electronics circuits. However circuits that operate at high speed require the multiplexer to be automatically selected. A mechanical switch cannot perform this task satisfactorily. Therefore, multiplexer used to perform high speed switching are constructed of electronic components.

Multiplexer handle two type of data that is analog and digital. For analog application, multiplexer are built of relays and transistor switches. For digital application, they are built from standard logic gates.

The multiplexer used for digital applications, also called digital multiplexer, is a circuit with many input but only one output. By applying control signals, we can steer any input to the output. Few types of multiplexer are 2-to-1, 4-to-1, 8-to-1, 16-to-1 multiplexer.

Following figure shows the general idea of a multiplexer with n input signal, m control signals and one output signal.

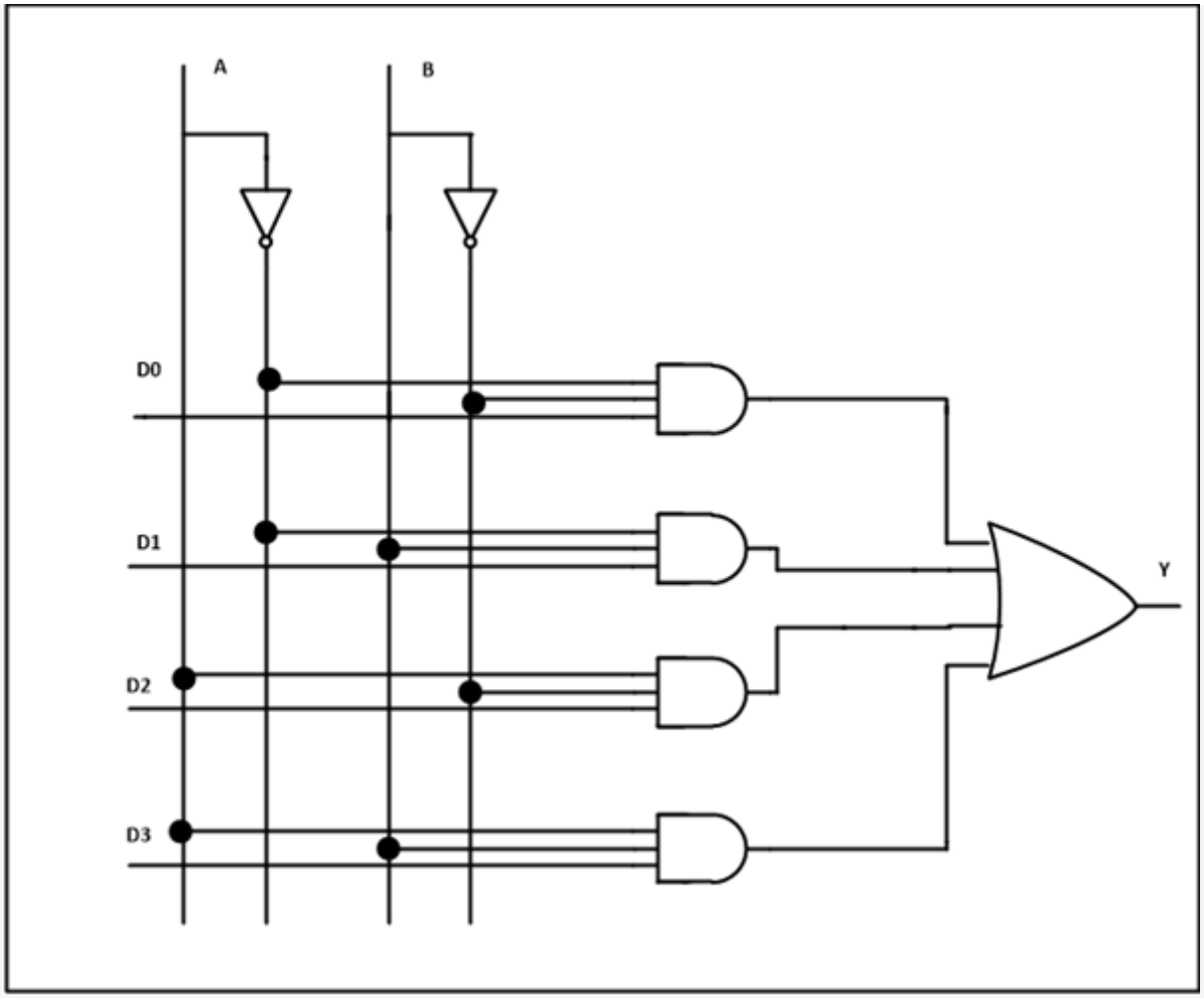


Multiplexer Pin Diagram

Understanding 4-to-1 Multiplexer:

The 4-to-1 multiplexer has 4 input bit, 2 control bits, and 1 output bit. The four input bits are D0,D1,D2 and D3. only one of this is transmitted to the output y. The output depends on the value of AB which is the control input. The control input determines which of the input data bit is transmitted to the output.

For instance, as shown in fig. when AB = 00, the upper AND gate is enabled while all other AND gates are disabled. Therefore, data bit D0 is transmitted to the output, giving Y = Do.



4 to 1 Multiplexer Circuit Diagram – ElectronicsHub.Org

If the control input is changed to AB =11, all gates are disabled except the bottom AND gate. In this case, D3 is transmitted to the output and Y = D3.

- An example of 4-to-1 multiplexer is IC 74153 in which the output is same as the input.
- Another example of 4-to-1 multiplexer is 45352 in which the output is the compliment of the input.
- Example of 16-to-1 line multiplexer is IC74150.

Applications of Multiplexer:

Multiplexer are used in various fields where multiple data need to be transmitted using a single line. Following are some of the applications of multiplexers -

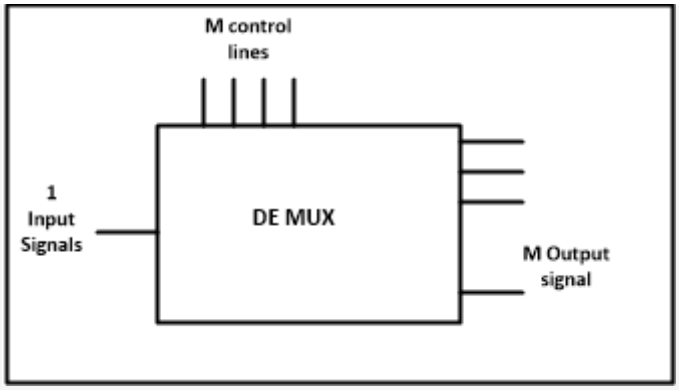
1. **Communication system** – Communication system is a set of system that enable communication like transmission system, relay and tributary station, and communication network. The efficiency of communication system can be increased considerably using multiplexer. Multiplexer allow the process of transmitting different type of data such as audio, video at the same time using a single transmission line.

2. **Telephone network –** In telephone network, multiple audio signals are integrated on a single line for transmission with the help of multiplexers. In this way, multiple audio signals can be isolated and eventually, the desire audio signals reach the intended recipients.
3. **Computer memory** - Multiplexers are used to implement huge amount of memory into the computer, at the same time reduces the number of copper lines required to connect the memory to other parts of the computer circuit.
4. **Transmission from the computer system of a satellite** – Multiplexer can be used for the transmission of data signals from the computer system of a satellite or spacecraft to the ground system using the GPS (Global Positioning System) satellites.

Demultiplexer:

Demultiplexer means one to many. A demultiplexer is a circuit with one input and many output. By applying control signal, we can steer any input to the output. Few types of demultiplexer are 1-to 2, 1-to-4, 1-to-8 and 1-to 16 demultiplexer.

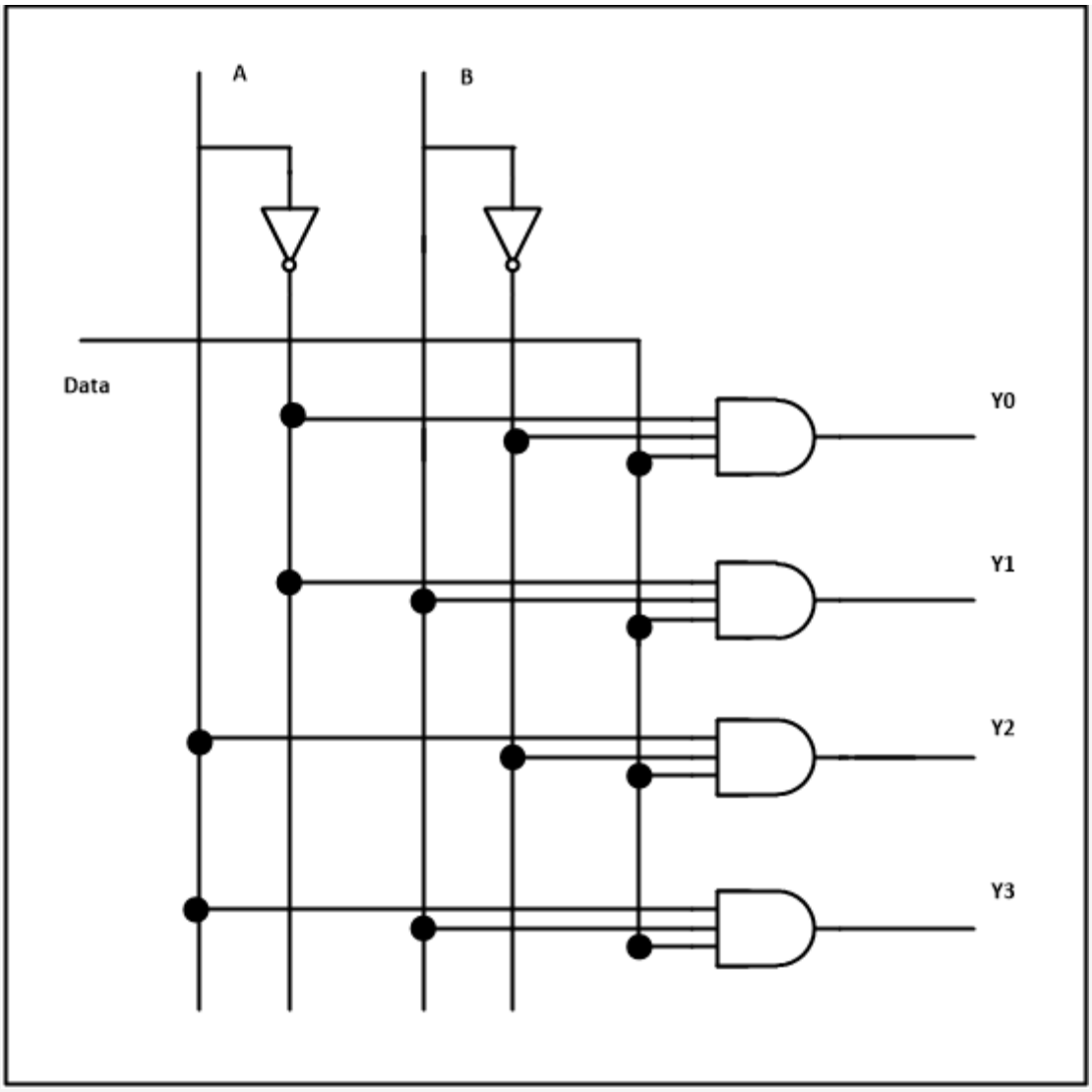
Following figure illustrate the general idea of a demultiplexer with 1 input signal, m control signals, and n output signals.



Demultiplexer Pin Diagram

Understanding 1- to-4 Demultiplexer:

The 1-to-4 demultiplexer has 1 input bit, 2 control bit, and 4 output bits. An example of 1-to-4 demultiplexer is IC 74155. The 1-to-4 demultiplexer is shown in figure below-



1 to 4 Dempultiplexer Circuit Diagram – ElectronicsHub.Org

The input bit is labelled as Data D. This data bit is transmitted to the data bit of the output lines. This depends on the value of AB, the control input.

When AB = 01, the upper second AND gate is enabled while other AND gates are disabled. Therefore, only data bit D is transmitted to the output, giving Y1 = Data.

If D is low, Y1 is low. IF D is high,Y1 is high. The value of Y1 depends upon the value of D. All other outputs are in low state.

If the control input is changed to AB = 10, all the gates are disabled except the third AND gate from the top. Then, D is transmitted only to the Y2 output, and Y2 = Data.

Example of 1-to-16 demultiplexer is IC 74154 it has 1 input bit, 4 control bits and 16 output bit.

Applications of Demultiplexer:

1. Demultiplexer is used to connect a single source to multiple destinations. The main application area of demultiplexer is communication system where multiplexer are used. Most of the communication system are bidirectional i.e. they function in both ways (transmitting and receiving signals). Hence, for most of the applications, the multiplexer and demultiplexer work in sync. Demultiplexer are also used for reconstruction of parallel data and ALU circuits.
2. **Communication System** - Communication system use multiplexer to carry multiple data like audio, video and other form of data using a single line for transmission. This process make the transmission easier. The demultiplexer receive the output signals of the multiplexer and converts them back to the original form of the data at the receiving end. The multiplexer and demultiplexer work together to carry out the process of transmission and reception of data in communication system.
3. **ALU (Arithmetic Logic Unit)** – In an ALU circuit, the output of ALU can be stored in multiple registers or storage units with the help of demultiplexer. The output of ALU is fed as the data input to the demultiplexer. Each output of demultiplexer is connected to multiple register which can be stored in the registers.
4. **Serial to parallel converter** - A serial to parallel converter is used for reconstructing parallel data from incoming serial data stream. In this technique, serial data from the incoming serial data stream is given as data input to the demultiplexer at the regular intervals. A counter is attach to the control input of the demultiplexer. This counter directs the data signal to the output of the demultiplexer where these data signals are stored. When all data signals have been stored, the output of the demultiplexer can be retrieved and read out in parallel.

Source: http://www.electronicshub.org/multiplexer-and-demultiplexer/

Flip flops

Flip flop are actually an application of logic gates. With the help of Boolean logic you can create memory with them. Flip flops can also be considered as the most basic idea of a Random Access Memory [RAM]. When a certain input value is given to them, they will be remembered and executed, if the logic gates are designed correctly. A higher application of flip flops is helpful in designing better electronic circuits.

The most commonly used application of flip flops is in the implementation of a feedback circuit. As a memory relies on the feedback concept, flip flops can be used to design it.

There are mainly four types of flip flops that are used in electronic circuits. They are

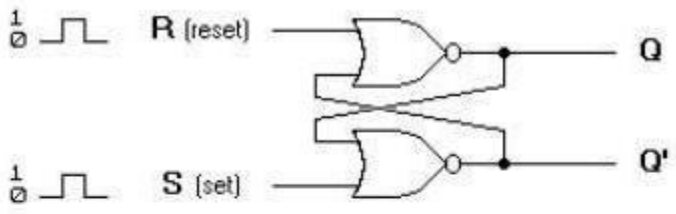
1. **The basic Flip Flop or S-R Flip Flop**
2. **Delay Flip Flop [D Flip Flop]**
3. **J-K Flip Flop**
4. **T Flip Flop**

1. S-R Flip Flop

The SET-RESET flip flop is designed with the help of two NOR gates and also two NAND gates. These flip flops are also called S-R Latch.

- ### S-R Flip Flop using NOR Gate

The design of such a flip flop includes two inputs, called the SET [S] and RESET [R]. There are also two outputs, Q and Q'. The diagram and truth table is shown below.



(a) Logic diagram

S R	Q Q'	
1 0	1 0	
0 0	1 0	(after S=1, R=0)
0 1	0 1	
0 0	0 1	(after S=0, R=1)
1 1	0 0	

(b) Truth table

Basic flip-flop circuit with NOR gates

S-R Flip Flop using NOR Gate

From the diagram it is evident that the flip flop has mainly four states. They are

S=1, R=0—Q=1, Q'=0

This state is also called the SET state.

S=0, R=1—Q=0, Q'=1

This state is known as the RESET state.

In both the states you can see that the outputs are just compliments of each other and that the value of Q follows the value of S.

S=0, R=0—Q & Q' = Remember

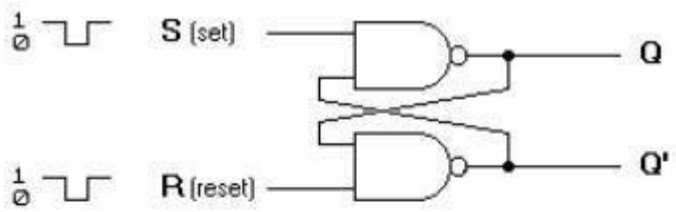
If both the values of S and R are switched to 0, then the circuit remembers the value of S and R in their previous state.

S=1, R=1—Q=0, Q’=0 [Invalid]

This is an invalid state because the values of both Q and Q’ are 0. They are supposed to be compliments of each other. Normally, this state must be avoided.

- ## S-R Flip Flop using NAND Gate

The circuit of the S-R flip flop using NAND Gate and its truth table is shown below.



(a) Logic diagram

S	R	Q	Q'	
1	0	0	1	
1	1	0	1	(after S=1, R=0)
0	1	1	0	
1	1	1	0	(after S=0, R=1)
0	0	1	1	

(b) Truth table

Basic flip-flop circuit with NAND gates

S-R Flip Flop using NAND Gate

Like the NOR Gate S-R flip flop, this one also has four states. They are

S=1, R=0—Q=0, Q’=1

This state is also called the SET state.

S=0, R=1—Q=1, Q’=0

This state is known as the RESET state.

In both the states you can see that the outputs are just compliments of each other and that the value of Q follows the compliment value of S.

S=0, R=0—Q=1, & Q' =1 [Invalid]

If both the values of S and R are switched to 0 it is an invalid state because the values of both Q and Q' are 1. They are supposed to be compliments of each other. Normally, this state must be avoided.

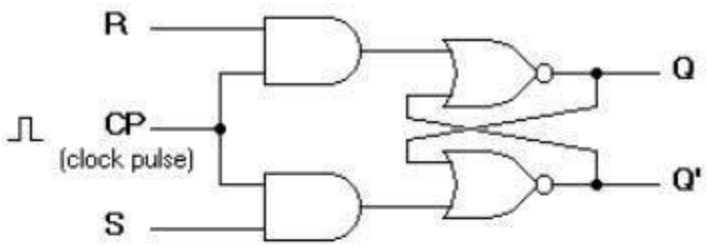
S=1, R=1—Q & Q'= Remember

If both the values of S and R are switched to 1, then the circuit remembers the value of S and R in their previous state.

- ## Clocked S-R Flip Flop

It is also called a Gated S-R flip flop.

The problems with S-R flip flops using NOR and NAND gate is the invalid state. This problem can be overcome by using a bistable SR flip-flop that can change outputs when certain invalid states are met, regardless of the condition of either the Set or the Reset inputs. For this, a clocked S-R flip flop is designed by adding two AND gates to a basic NOR Gate flip flop. The circuit diagram and truth table is shown below.



(a) Logic diagram

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

(b) Truth table

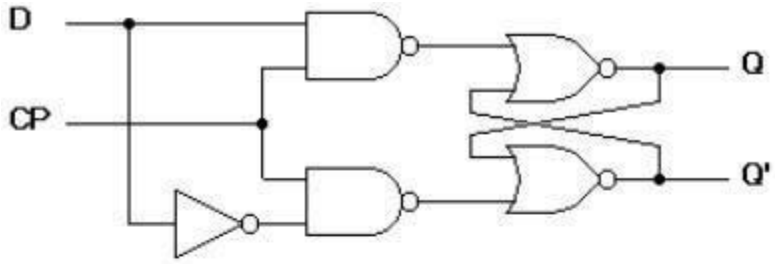
Clocked SR flip-flop

Clocked S-R Flip Flop

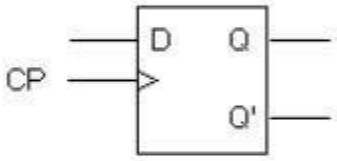
A clock pulse [CP] is given to the inputs of the AND Gate. When the value of the clock pulse is ‘0’, the outputs of both the AND Gates remain ‘0’. As soon as a pulse is given the value of CP turns ‘1’. This makes the values at S and R to pass through the NOR Gate flip flop. But when the values of both S and R values turn ‘1’, the HIGH value of CP causes both of them to turn to ‘0’ for a short moment. As soon as the pulse is removed, the flip flop state becomes intermediate. Thus either of the two states may be caused, and it depends on whether the set or reset input of the flip-flop remains a ‘1’ longer than the transition to ‘0’ at the end of the pulse. Thus the invalid states can be eliminated.

2. D Flip Flop

The circuit diagram and truth table is given below.



(a) Logic diagram with NAND gates



(b) Graphical symbol

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

(c) Transition table

Clocked D flip-flop

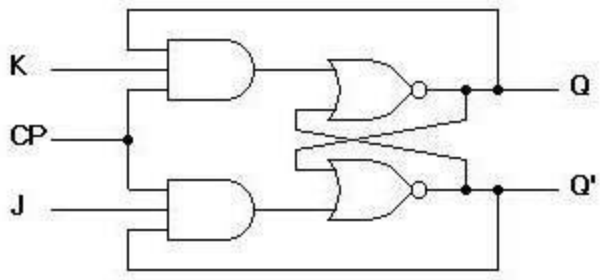
D Flip Flop

D flip flop is actually a slight modification of the above explained clocked SR flip-flop. From the figure you can see that the D input is connected to the S input and the complement of the D input is connected to the R input. The D input is passed on to the flip flop when the value of CP is ‘1’. When CP is HIGH, the flip flop moves to the SET state. If it is ‘0’, the flip flop switches to the CLEAR state.

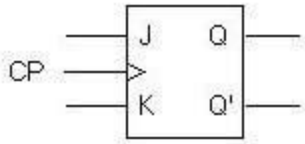
To know more about the triggering of flip flop click on the link below.

3. J-K Flip Flop

The circuit diagram and truth-table of a J-K flip flop is shown below.



(a) Logic diagram



(b) Graphical symbol

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

(c) Transition table

Clocked JK flip-flop

J-K Flip Flop

A J-K flip flop can also be defined as a modification of the S-R flip flop. The only difference is that the intermediate state is more refined and precise than that of a S-R flip flop.

The behavior of inputs J and K is same as the S and R inputs of the S-R flip flop. The letter J stands for SET and the letter K stands for CLEAR.

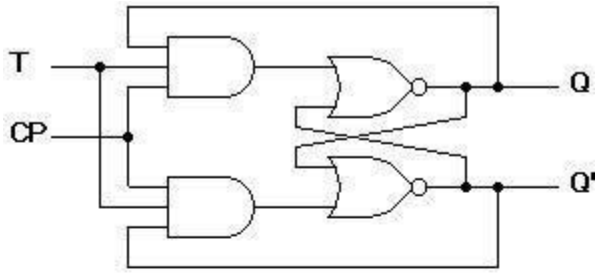
When both the inputs J and K have a HIGH state, the flip-flop switch to the complement state. So, for a value of Q = 1, it switches to Q=0 and for a value of Q = 0, it switches to Q=1.

The circuit includes two 3-input AND gates. The output Q of the flip flop is returned back as a feedback to the input of the AND along with other inputs like K and clock pulse [CP]. So, if the value of CP is '1', the flip flop gets a CLEAR signal and with the condition that the value of Q was earlier 1. Similarly output Q' of the flip flop is given as a feedback to the input of the AND along with other inputs like J and clock pulse [CP]. So the output becomes SET when the value of CP is 1 only if the value of Q' was earlier 1.

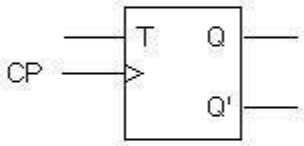
The output may be repeated in transitions once they have been complimented for J=K=1 because of the feedback connection in the JK flip-flop. This can be avoided by setting a time duration lesser than the propagation delay through the flip-flop. The restriction on the pulse width can be eliminated with a master-slave or edge-triggered construction.

4. T Flip Flop

This is a much simpler version of the J-K flip flop. Both the J and K inputs are connected together and thus are also called a single input J-K flip flop. When clock pulse is given to the flip flop, the output begins to toggle. Here also the restriction on the pulse width can be eliminated with a master-slave or edge-triggered construction. Take a look at the circuit and truth table below.



(a) Logic diagram



(b) Graphical symbol

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

(c) Transition table

Clocked T flip-flop

T Flip Flop

TAKE A LOOK : TRIGGERING OF FLIP FLOPS

TAKE A LOOK : MASTER-SLAVE FLIP FLOP CIRCUIT

Shift registers

1.0 Introduction

Shift registers are a type of sequential logic circuit, mainly for storage of digital data. They are a group of flip -flops connected in a chain so that the output from one flip-flop becomes the input of the next flip -flop. Most of the registers possess no characteristic internal sequence of states. All flip-flop is driven by a common clock, and all are set or reset simultaneously.

In these few lectures, the basic types of shift registers are studied, such as Serial In - Serial Out, Serial In - Parallel Out, Parallel In – Serial Out, Parallel In - Parallel Out, and bidirectional shift registers. A special form of counter - the shift register counter, is also introduced.

Register:

- ??A set of n flip-flops
- ??Each flip-flop stores one bit
- ??Two basic functions: data storage (Figure 1.2) and data movement (Figure 1.1).

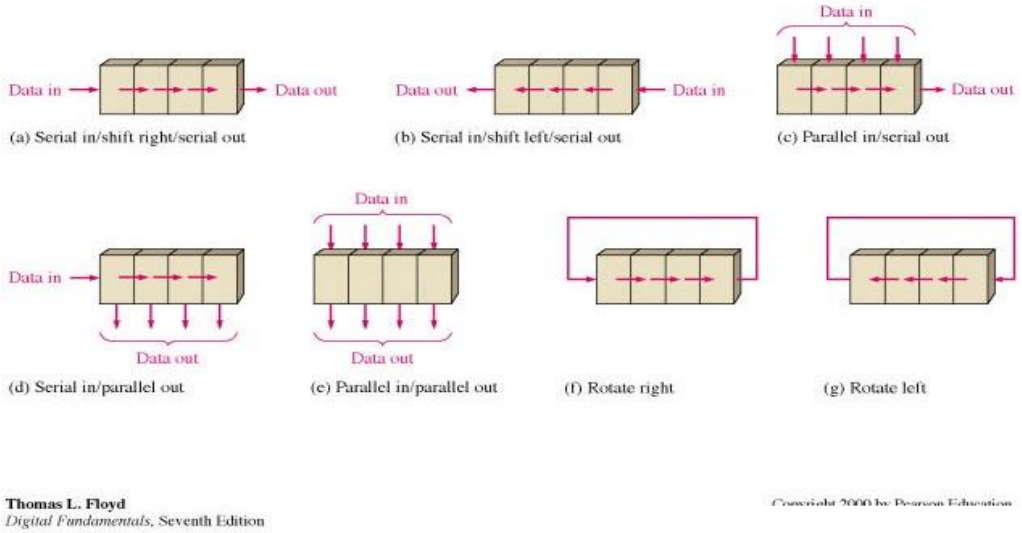
Shift Register:

- ??A register that allows each of the flip-flops to pass the stored information to its adjacent neighbour
- ??Figure 1.1 shows the basic data movement in shift registers.

Counter:

- ??A register that goes through a predetermined sequence of states

Figure 1.1: Basic data movement in shift registers [Floyd]



Storage Capacity:

The storage capacity of a register is the total number of bits (1 or 0) of digital data it can retain. Each stage (flip flop) in a shift register represents one bit of storage capacity. Therefore the number of stages in a register determines its storage capacity.

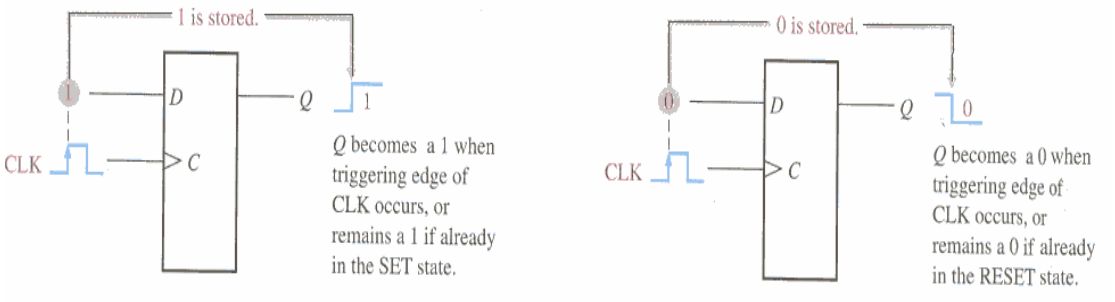


Figure 1.2: The flip-flop as a storage element.

Serial In - Serial Out Shift Registers

The serial in/serial out shift register accepts data serially – that is, one bit at a time on a single line. It produces the stored information on its output also in serial form.

Example: Basic four-bit shift register

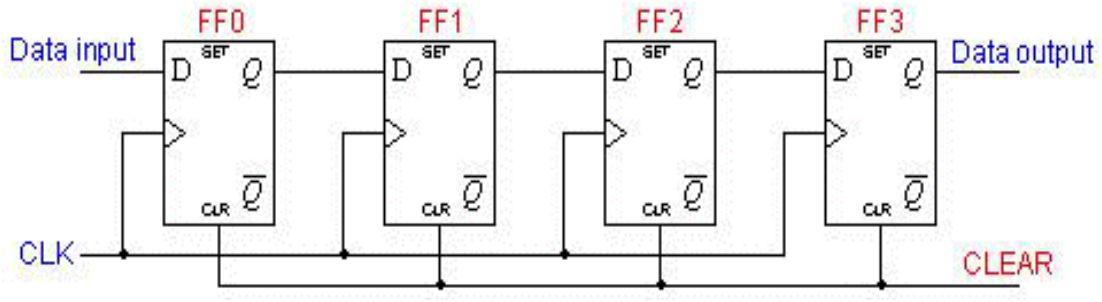


Figure 2.1

A basic four-bit shift register can be constructed using four D flip -flops, as shown in Figure 2.1.

The operation of the circuit is as follows.

??The register is first cleared, forcing all four outputs to zero.

??The input data is then applied sequentially to the D input of the first flip-flop on the left (FF0).
??During each clock pulse, one bit is transmitted from left to right.
??Assume a data word to be 1001.
??The least significant bit of the data has to be shifted through the register from FF0 to FF3.

In order to get the data out of the register, they must be shifted out serially. This can be done destructively or non-destructively. For destructive readout, the original data is lost and at the end of the read cycle, all flip-flops are reset to zero.

FF0	FF1	FF2	FF3	
0	0	0	0	1001

The data is loaded to the register when the control line is HIGH (ie WRITE). The data can be shifted out of the register when the control line is LOW (ie READ).

Clear	FF0	FF1	FF2	FF3
1001	0	0	0	0

WRITE:

FF0	FF1	FF2	FF3	
1	0	0	1	0000

READ:

FF0	FF1	FF2	FF3	
1	0	0	1	1001

Figure 2.2 illustrates entry of the four bits 1010 into the register. Figure 2.3 shows the four bits (1010) being serially shifted out of the register and replaced by all zeros.

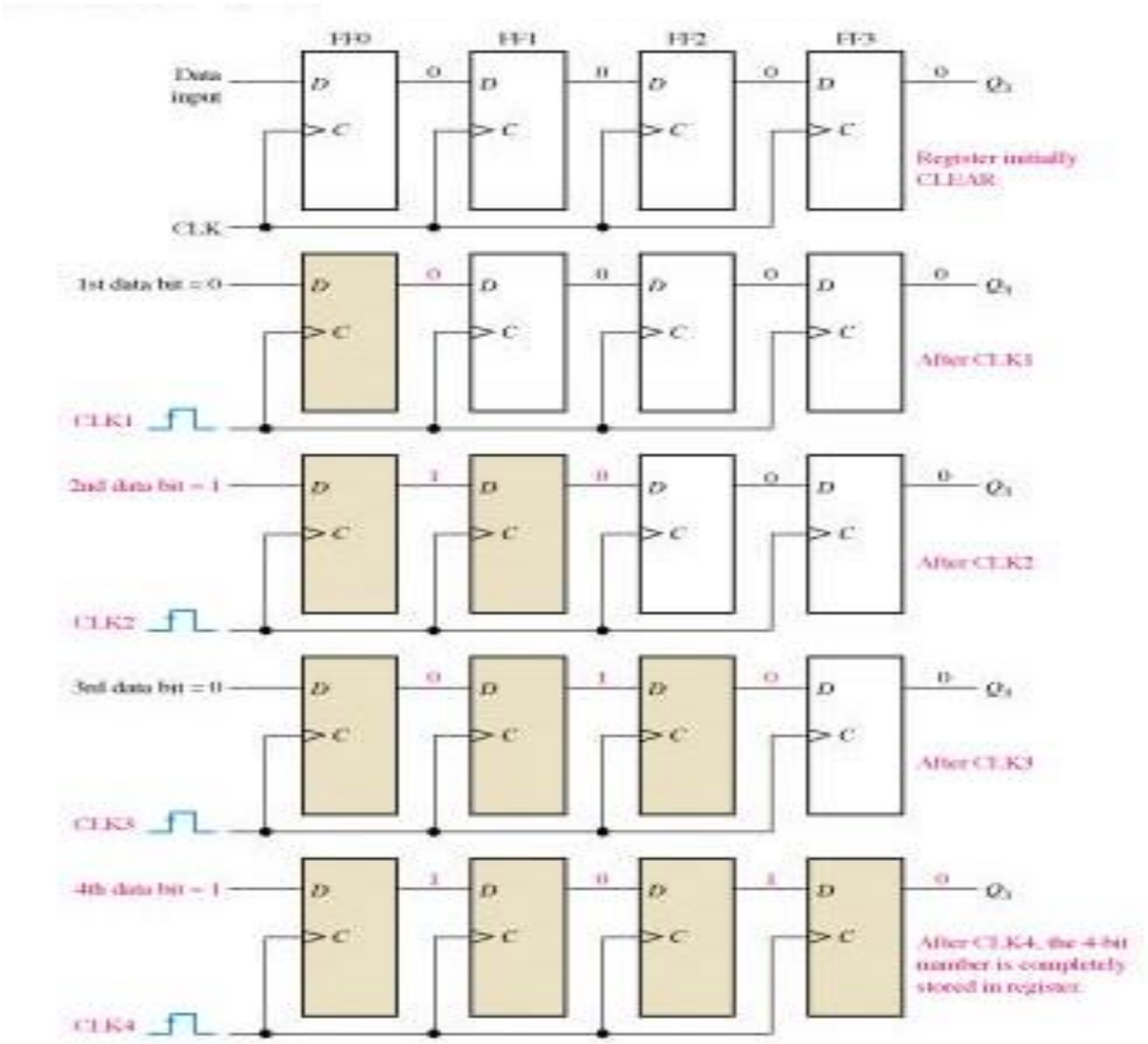


Figure 2.2: Four bits (1010) being entered serially into the register.

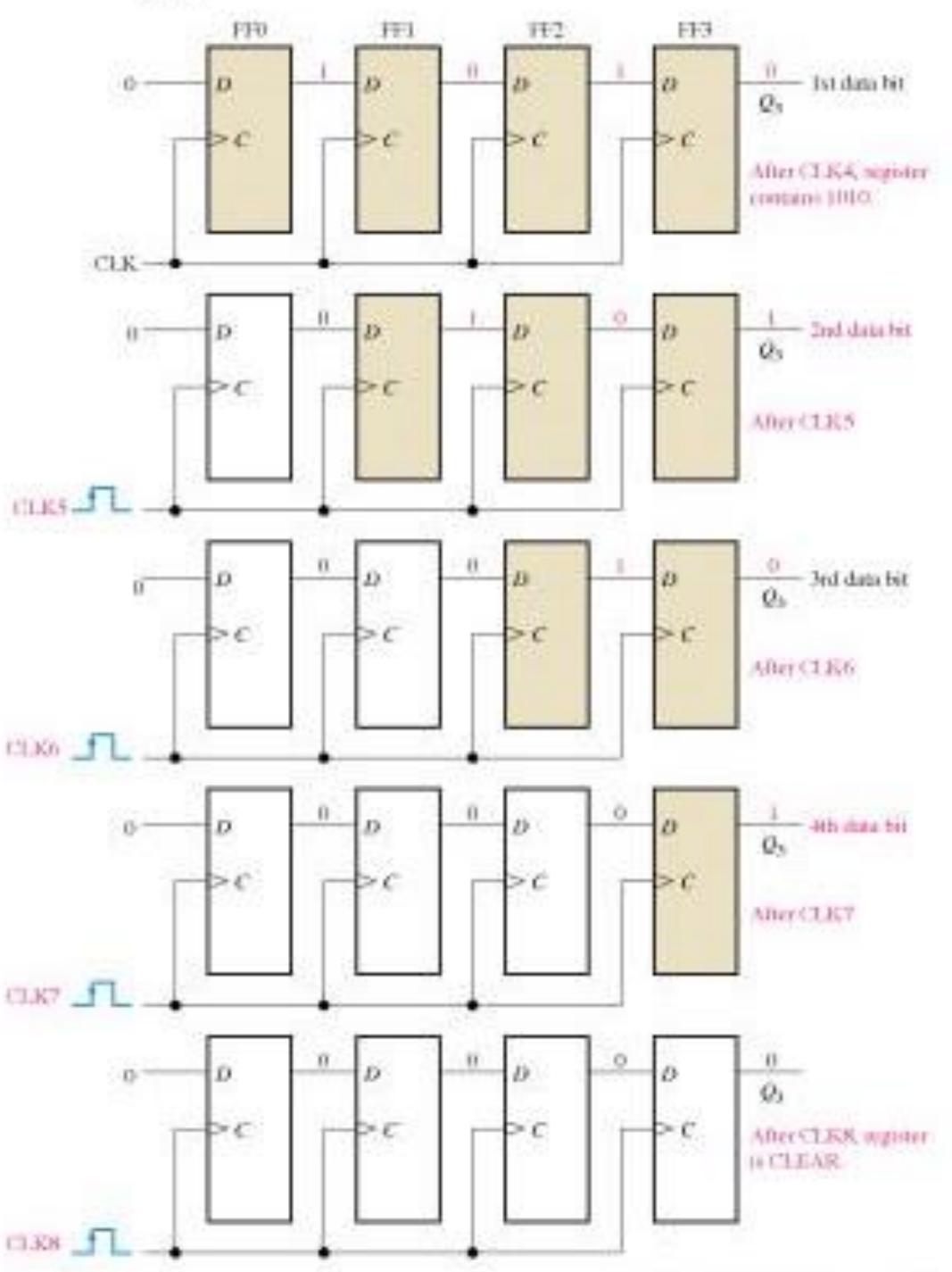


Figure 2.3: Four bits (1010) being serially shifted out of the register and replaced by all zeros

2.2 5-bit serial in/serial out shift registers

Figure 2.4 illustrates entry of the five bits 11010 into the register.

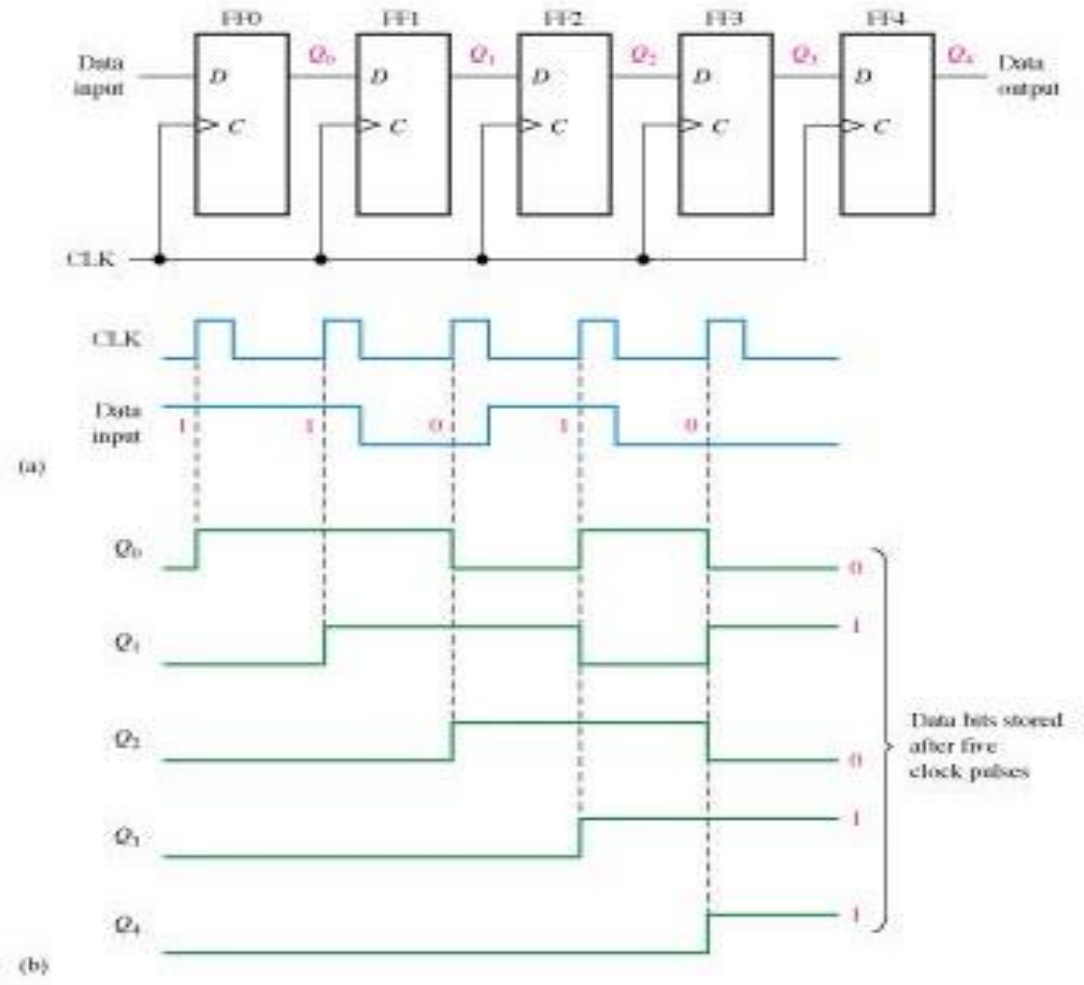
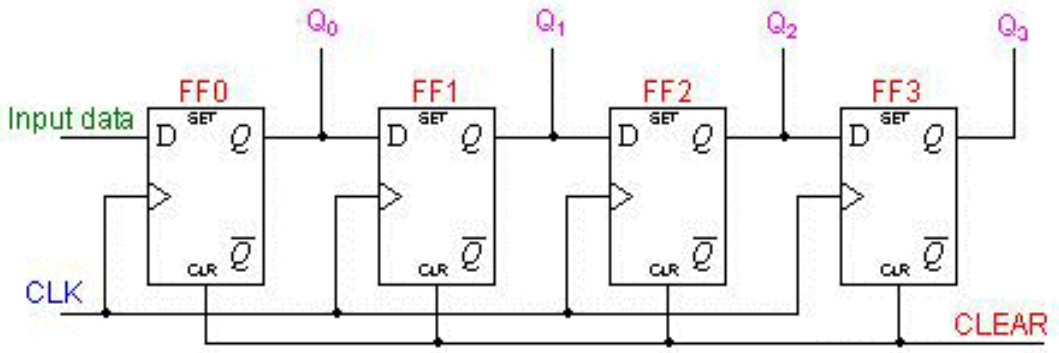


Figure 2.4

Serial In - Parallel Out Shift Registers

For this kind of register, data bits are entered serially in the same manner as dis cussed in the last section. The difference is the way in which the data bits are taken out of the register. Once the data are stored, each bit appears on its respective output line, and all bits are available simultaneously. A construction of a four-bit serial in - parallel out register is shown below.

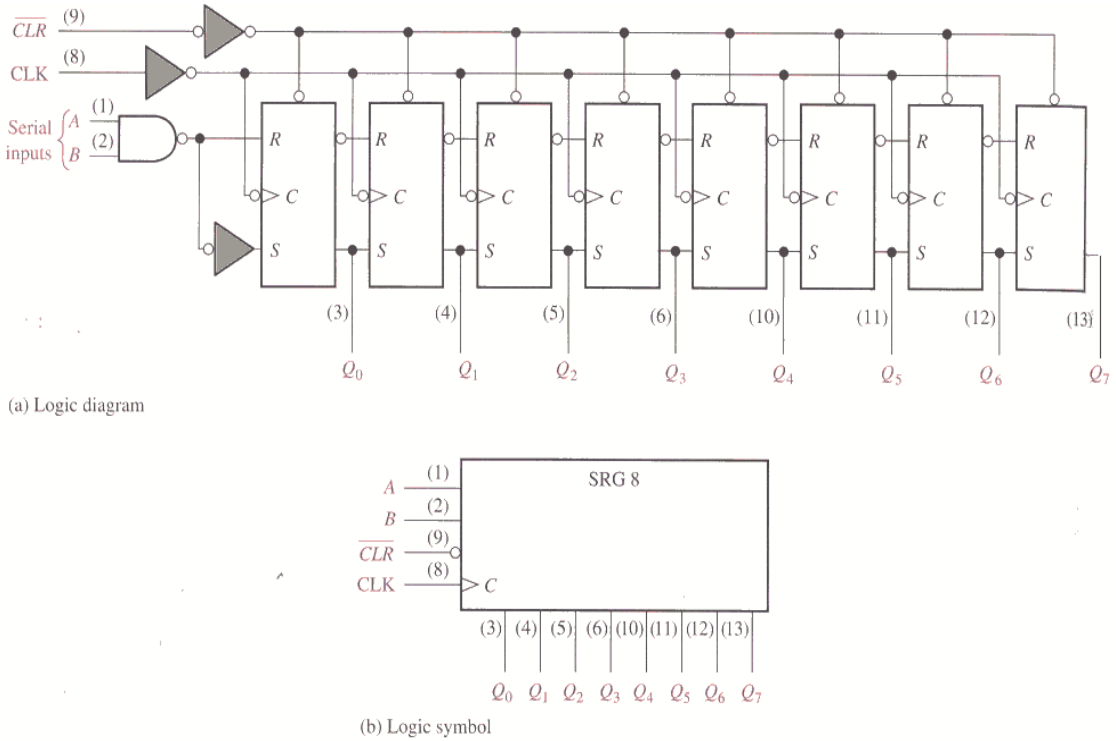


In the table below, we can see how the four-bit binary number 1001 is shifted to the Q outputs of the register.

Clear	FF0	FF1	FF2	FF3
1001	0	0	0	0
	1	0	0	0
	0	1	0	0
	0	0	1	0
	1	0	0	1

An 8-bit serial in/parallel out shift register (74HC164)

The 74HC164 is an example of an IC shift register having serial in/parallel out operation. The logic diagram and logic block are shown in Figure 3.1 (a),(b).

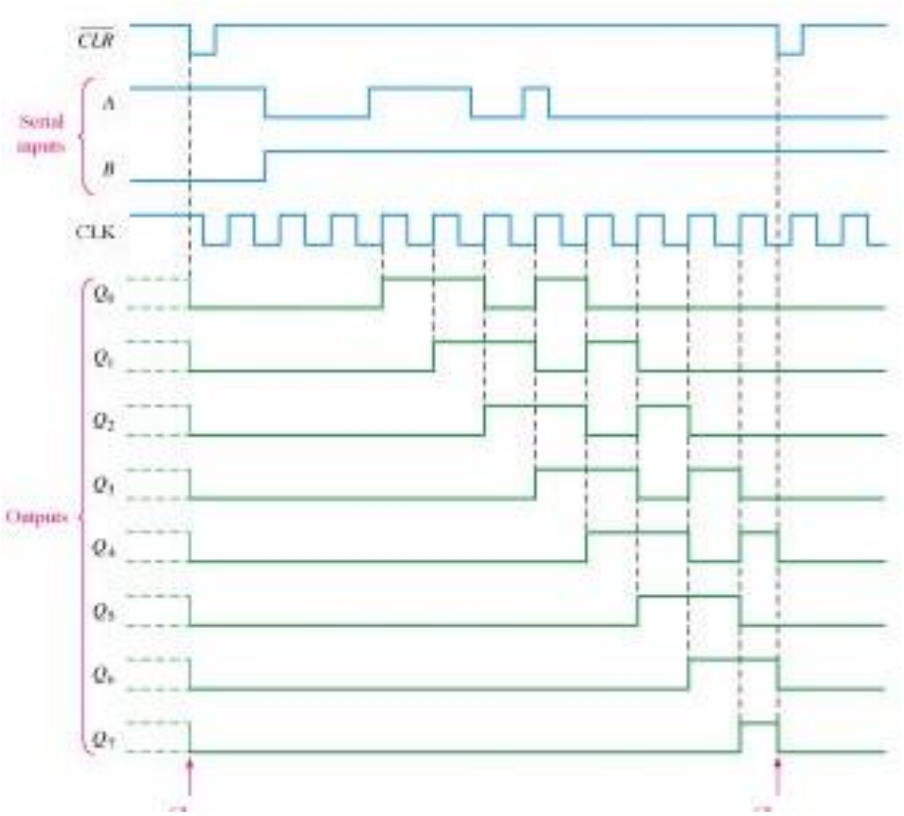


(a) Logic diagram

(b) Logic symbol

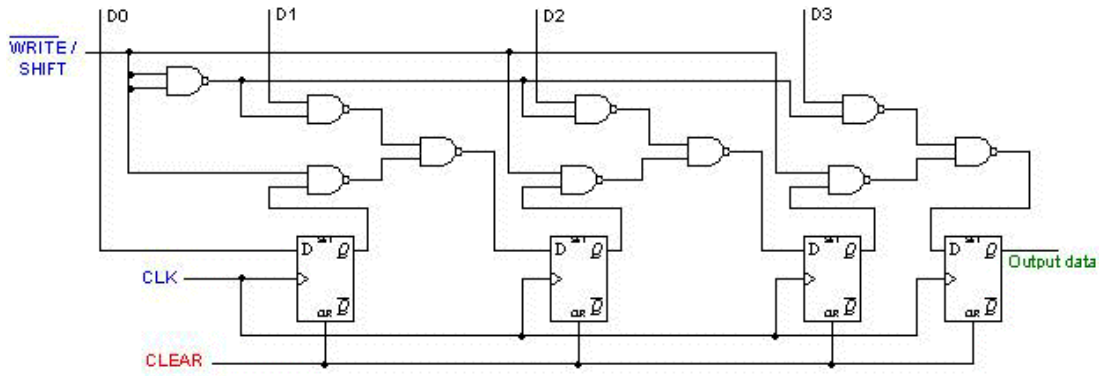
Figure 3.1: The logic diagram and logic block of 74HC164

Figure 3.2: The timing diagram of 74HC164



Parallel In - Serial Out Shift Registers

A four-bit parallel in - serial out shift register is shown below. The circuit uses D flip -flops and NAND gates for entering data (ie writing) to the register.



D0, D1, D2 and D3 are the parallel inputs, where D0 is the most significant bit and D3 is the least significant bit. To write data in, the mode control line is taken to LOW and the data is clocked in. The data can be shifted when the mode control line is HIGH as SHIFT is active high. The register performs right shift operation on the application of a clock pulse, as shown in the table below.

	Q_0	Q_1	Q_2	Q_3	
Clear	0	0	0	0	
Write	1	0	0	1	
Shift	1	0	0	1	
	1	1	0	0	1
	1	1	1	0	01
	1	1	1	1	001
	1	1	1	1	1001

8-bit Parallel Load Shift Register (74HC165)

The 74HC165 is an example of an IC shift register that has a parallel in/serial out operation. It can also be operated as serial in/serial out. Figure 4.1 shows the logic diagram and logic symbol of 74HC165.

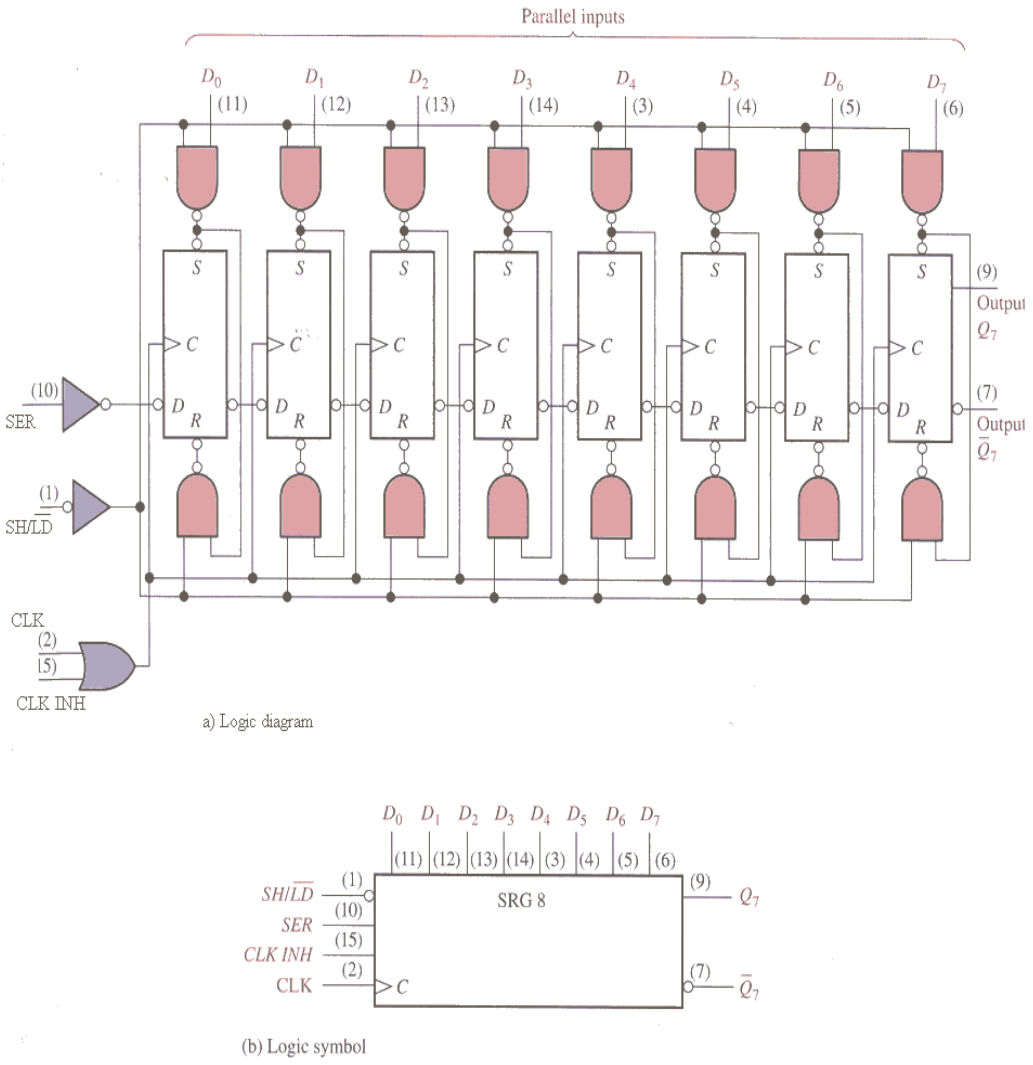


Figure 4.1: the logic diagram and logic symbol of 74HC165.

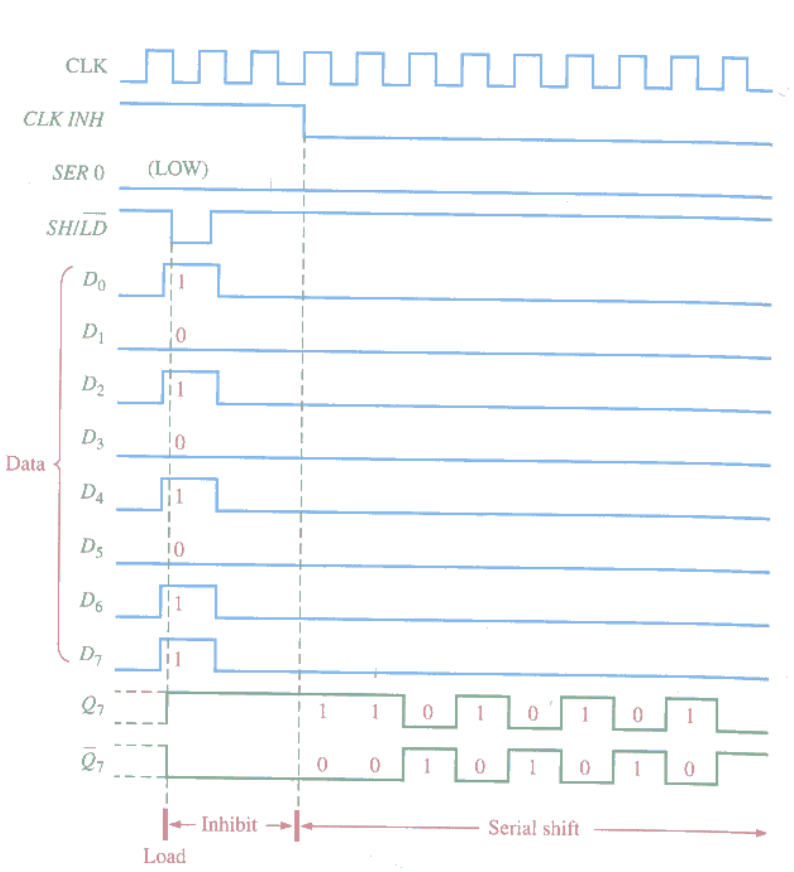


Figure 4.2: The timing diagram of 74HC165.

Parallel In - Parallel Out Shift Registers

For parallel in - parallel out shift registers, all data bits appear on the parallel outputs immediately following the simultaneous entry of the data bits. The following circuit is a four-bit parallel in - parallel out shift register constructed by D flip-flops.

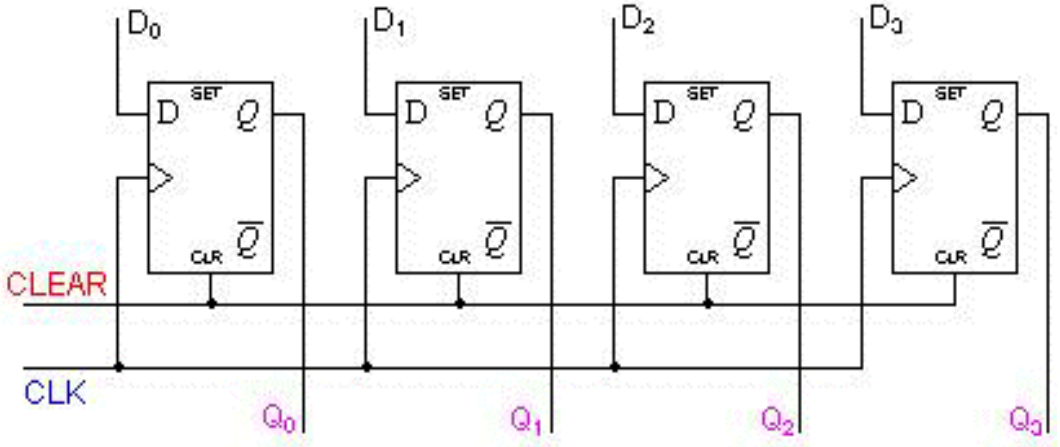


Figure 5.1

The D's are the parallel inputs and the Q's are the parallel outputs. Once the register is clocked, all the data at the D inputs appear at the corresponding Q outputs simultaneously.

4-bit Parallel-Access Shift Register (74HC195)

The 74HC195 can be used for parallel in/parallel out operation, serial in/serial out and serial in/parallel out operations. Q_3 is the output when it is used for parallel in/serial out operation.

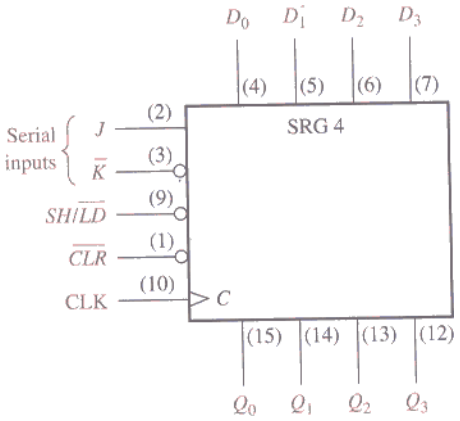


Figure 5.2: The 74LS195A 4-bit parallel access shift register

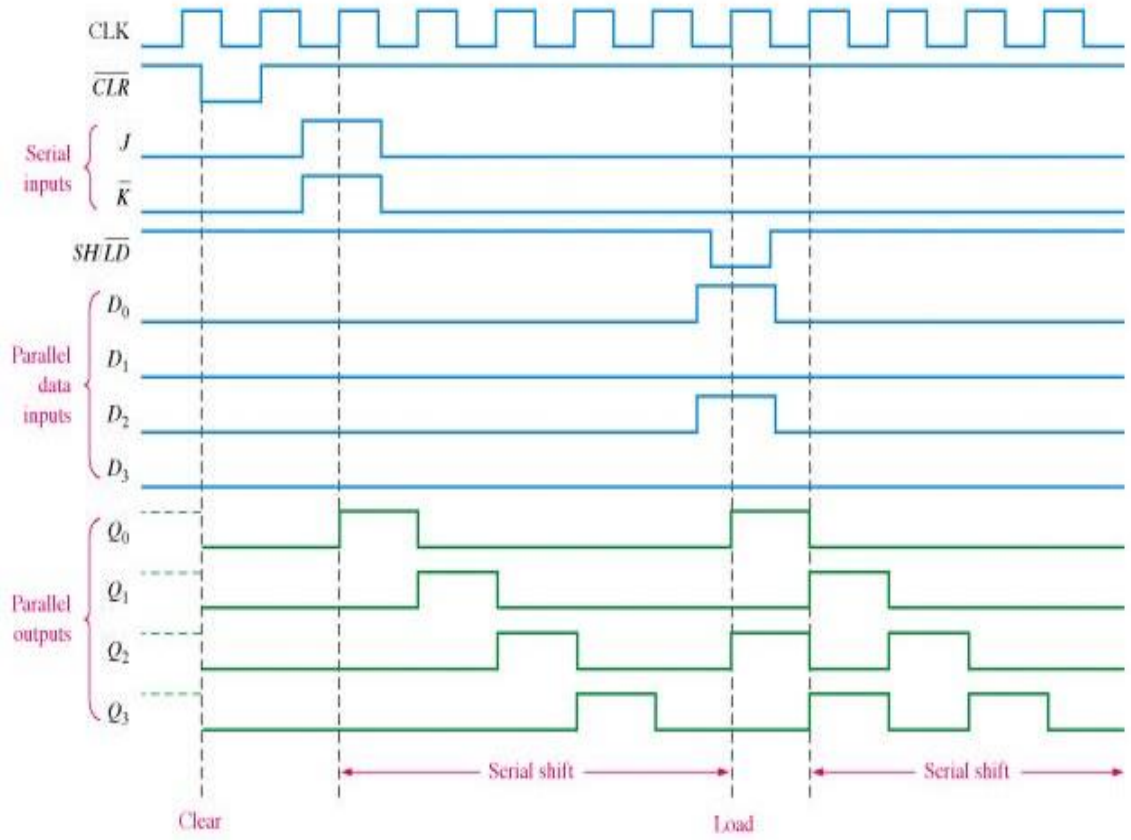


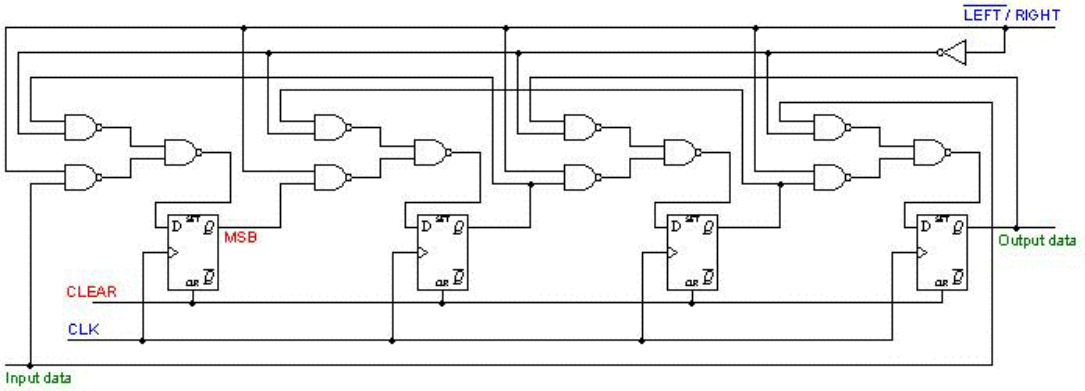
Figure 5.3: The timing diagram for 74LS195A shift register

Thomas L. Floyd
Digital Fundamentals, Seventh Edition
Pearson Education
Upper Saddle River, New Jersey 07458
All rights reserved.

Bidirectional Shift Registers

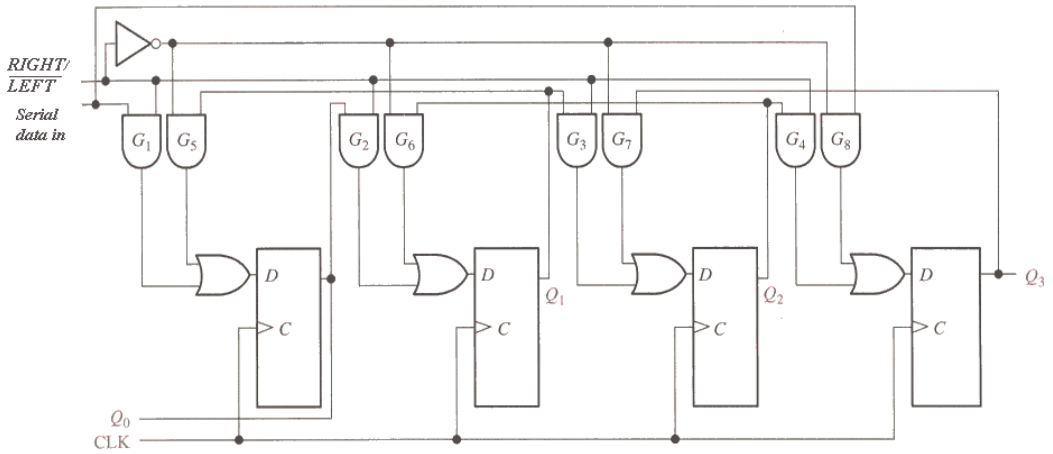
The registers discussed so far involved only right shift operations. Each right shift operation has the effect of successively dividing the binary number by two. If the operation is reversed (left shift), this has the effect of multiplying the number by two. With suitable gating arrangement a serial shift register can perform both operations.

A bidirectional, or reversible, shift register is one in which the data can be shift either left or right. A four-bit bidirectional shift register using D flip-flops is shown below.



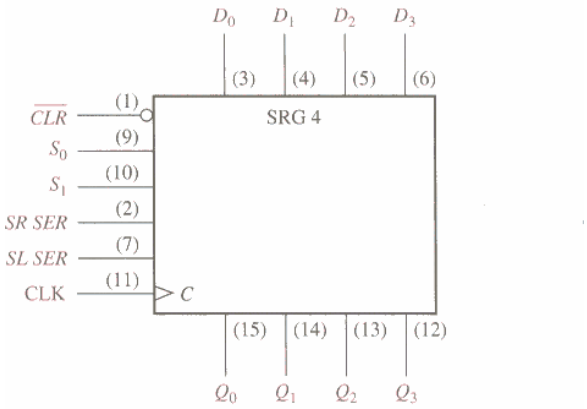
Here a set of NAND gates are configured as OR gates to select data inputs from the right or left adjacent bistables, as selected by the LEFT/RIGHT control line.

Alternative Circuit: [Floyd]



4-Bit Bidirectional Universal Shift Registers (74HC194)

The 74HC194 is a universal bi-directional shift register. It has both serial and parallel input and output capability.



[Floyd]

Figure 6.1:The 74HC194 4-bit bi-directional universal shift register

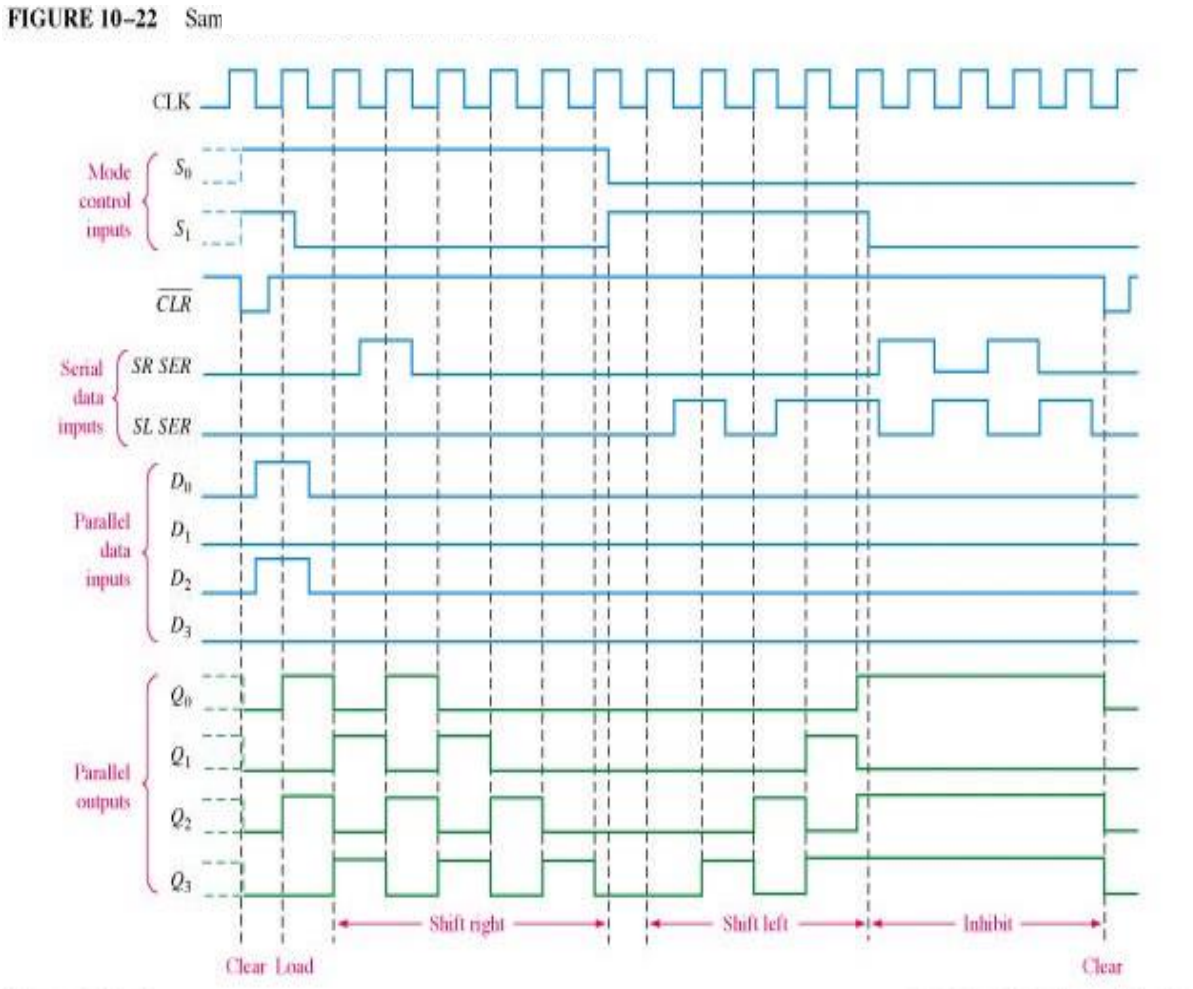


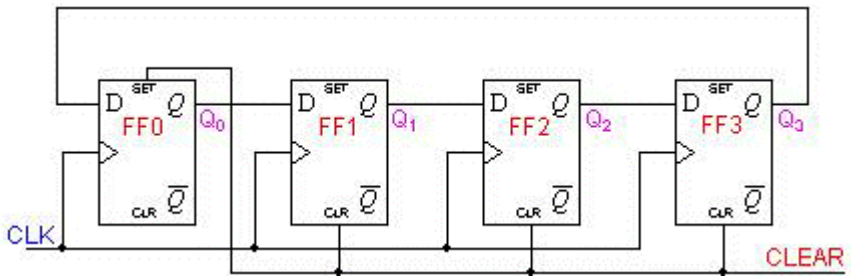
Figure 6.2:The timing diagram of 74HC194

Shift Register / Counters

Two of the most common types of shift register counters are introduced here: the Ring counter and the Johnson counter. They are basically shift registers with the serial outputs connected back to the serial inputs in order to produce particular sequences. These registers are classified as counters because they exhibit a specified sequence of states.

Ring Counters

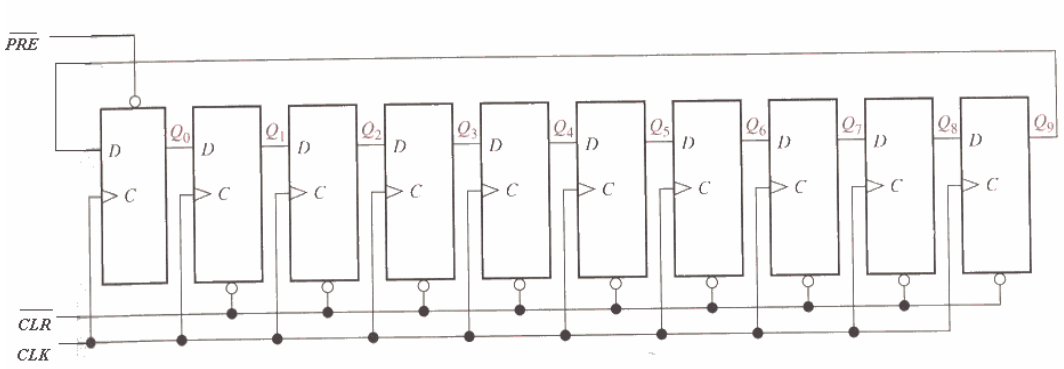
A ring counter is basically a circulating shift register in which the output of the most significant stage is fed back to the input of the least significant stage. The following is a 4-bit ring counter constructed from D flip-flops. The output of each stage is shifted into the next stage on the positive edge of a clock pulse. If the CLEAR signal is high, all the flip - flops except the first one FF0 are reset to 0. FF0 is preset to 1 instead.



Since the count sequence has 4 distinct states, the counter can be considered as a mod-4 counter. Only 4 of the maximum 16 states are used, making ring counters very inefficient in terms of state usage. But the major advantage of a ring counter over a binary counter is that it is self-decoding. No extra decoding circuit is needed to determine what state the counter is in.

Clock Pulse	Q3	Q2	Q1	Q0
0	0	0	0	1
1	0	0	1	0
2	0	1	0	0
3	1	0	0	0

Example: A 10-bit Ring Counter [Flyod]



Clock Pulse	Q_0	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8	Q_9
0	1	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0
2	0	0	1	0	0	0	0	0	0	0
3	0	0	0	1	0	0	0	0	0	0
4	0	0	0	0	1	0	0	0	0	0
5	0	0	0	0	0	1	0	0	0	0
6	0	0	0	0	0	0	1	0	0	0
7	0	0	0	0	0	0	0	1	0	0
8	0	0	0	0	0	0	0	0	1	0
9	0	0	0	0	0	0	0	0	0	1

Figure 7.1: 10-bit ring counter & its sequence

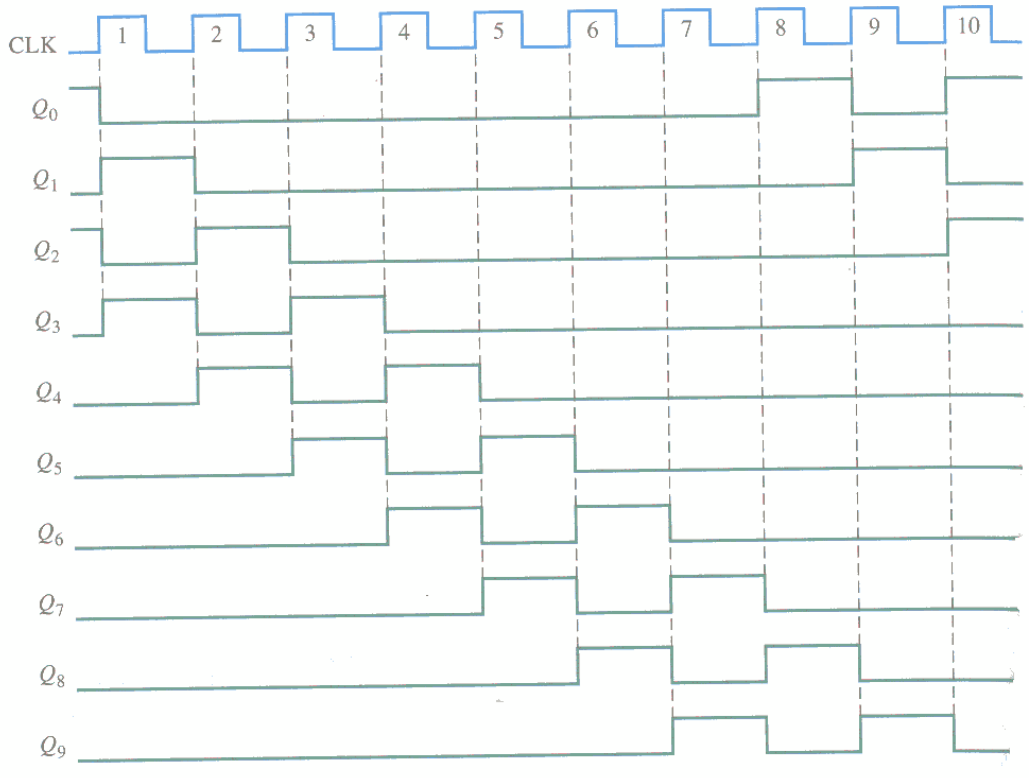
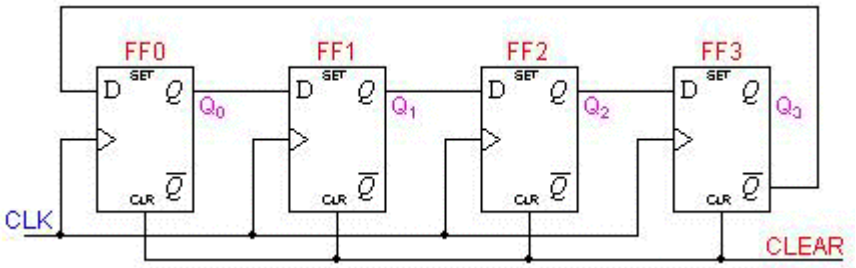


Figure 7.2: 10-bit ring counter waveform (initial state 1010000000)

Johnson Counters

Johnson counters are a variation of standard ring counters, with the inverted output of the last stage fed back to the input of the first stage. They are also known as twisted ring counters. An n-stage Johnson counter yields a count sequence of length 2n, so it may be considered to be a mod-2n counter. The circuit below shows a 4-bit Johnson counter. The state sequence for the counter is given in the table .

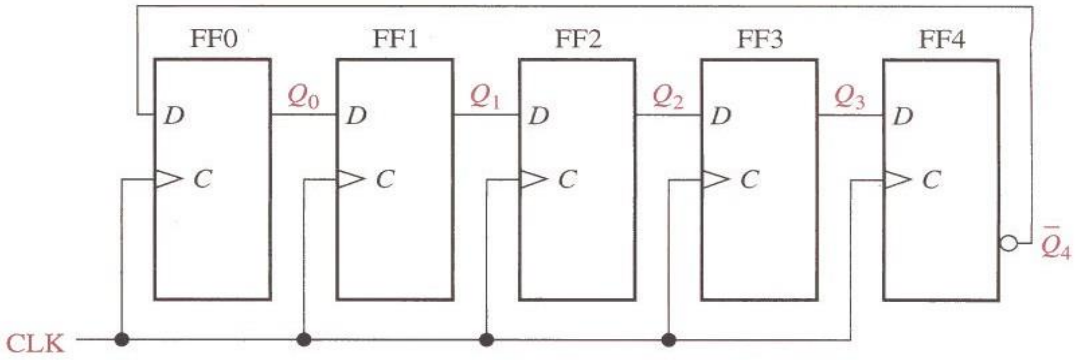


Clock Pulse	Q3	Q2	Q1	Q0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	1
3	0	1	1	1
4	1	1	1	1
5	1	1	1	0
6	1	1	0	0
7	1	0	0	0

?? Again, the apparent disadvantage of this counter is that the maximum available states are not fully utilized. Only eight of the sixteen states are being used.

?? Beware that for both the Ring and the Johnson counter must initially be forced into a valid state in the count sequence because they operate on a subset of the available number of states. Otherwise, the ideal sequence will not be followed.

Example: 5 bit Johnson Counter [Flyod]



Clock Pulse	Q_0	Q_1	Q_2	Q_3	Q_4
0	0	0	0	0	0
1	1	0	0	0	0
2	1	1	0	0	0
3	1	1	1	0	0
4	1	1	1	1	0
5	1	1	1	1	1
6	0	1	1	1	1
7	0	0	1	1	1
8	0	0	0	1	1
9	0	0	0	0	1

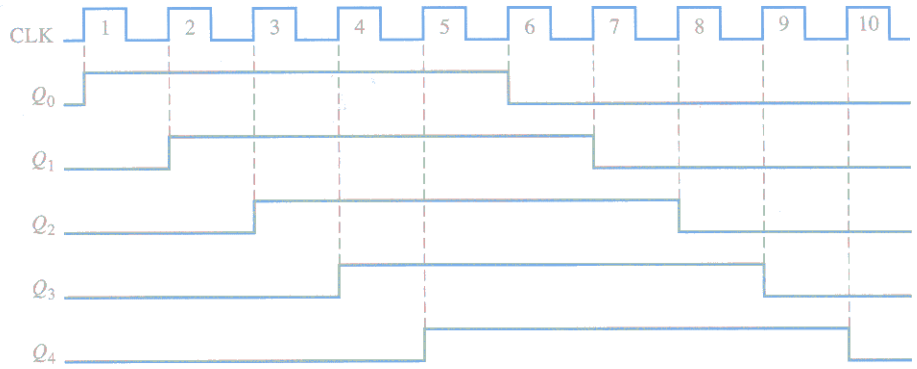


Figure 7.3: 5-bit Johnson Counter, its sequence and waveform

Applications

Shift registers can be found in many applications. Here is a list of a few.

??To produce time delay

The serial in -serial out shift register can be used as a time delay device. The amount of delay can be controlled by:

1. the number of stages in the register
2. the clock frequency

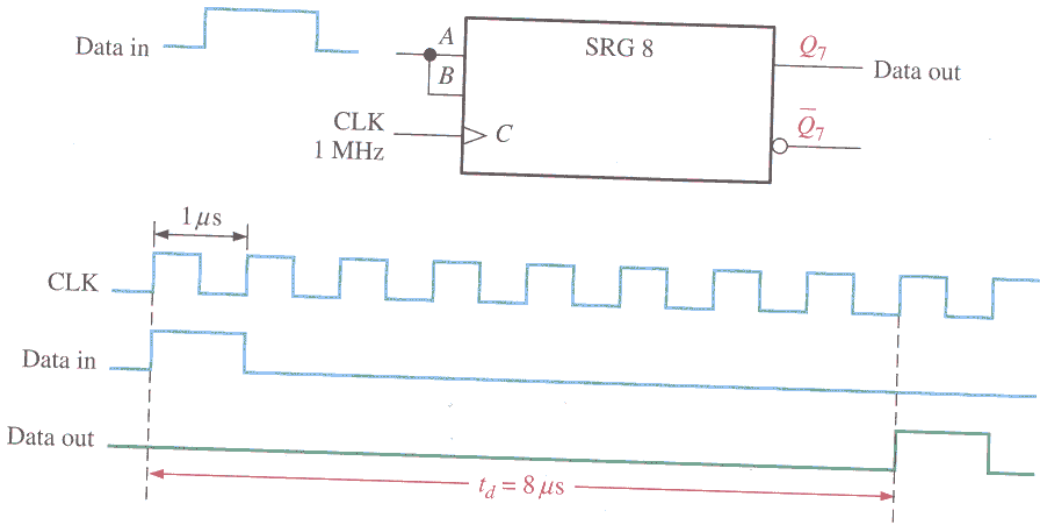


Figure 8.1: The shift register as a time-delay device.

To simplify combinational logic

The ring counter technique can be effectively utilized to implement synchronous sequential circuits. A major problem in the realization of sequential circuits is the assignment of binary codes to the internal states of the circuit in order to reduce the complexity of circuits required. By assigning one flip -flop to one internal state, it is possible to simplify the combinational logic required to realize the complete sequential circuit. When the circuit is in a particular state, the flip -flop corresponding to that state is set to HIGH and all other flip-flops remain LOW.

To convert serial data to parallel data

A computer or microprocessor-based system commonly requires incoming data to be in parallel format. But frequently, these systems must communicate with external devices that send or receive serial data. So, serial-to-parallel conversion is required. As shown in the previous sections, a serial in - parallel out register can achieve this.

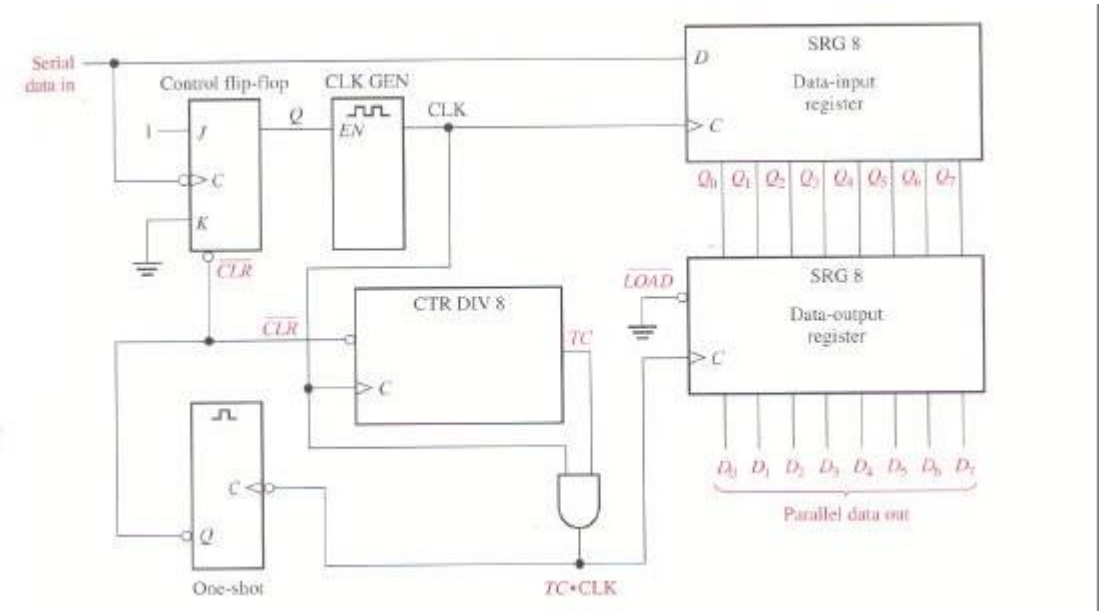


Figure 8.2: Simplified logic diagram of a serial-to-parallel converter

A/D and D/A Converter

Interfacing with the Analog World

Learning Objectives

On completion of this lesson you will be able to :

- learn about various terms of A/D and D/A converters.

Interfacing with the Analog World

A digital quantity will have a value that is specified as one of two possibilities such as 0 or 1.

A digital quantity will have a value that is specified as one of two possibilities such as 0 or 1, LOW or HIGH, true or false, and so on. In practice, the voltage representation a digital quantity such as a may actually have a value that is anywhere within specified ranges. For example, for TTL logic :

0V to 0.8V = logic 0
2V to 5V = logic 1

Any voltage falling in the range 0 to 0.8 V is given the digital value 0, and any voltage in the range 2 to 5 V is assigned the digital value 1. The digital circuits respond accordingly to all voltage values within a given range.

Most physical variables are analog in nature and can take on any value within a continuous range of values.

Most physical variables are analog in nature and can take on any value within a continuous range of values. Examples include temperature, pressure, light intensity, audio signals, position, rotational speed, and flow rate. Digital systems perform all of their internal operations using digital circuitry and digital operations. Any information that has to be inputted to a digital system must first be put into digital form. Similarly, the outputs from a digital system are always in digital form.

Transducer

A transducer is a device that converts the physical variable to an electrical variable.

The physical variable is normally a nonelectrical quantity. A transducer is a device that converts the physical variable to an electrical variable. Some common transducers include thermistors, photocells, photodiodes, flow meters, pressure transducers, and tachometers. The electrical output of the transducer is an analog current or voltage that is proportional to the physical variable it is monitoring. For example, the physical variable could be the temperature of water. Let's say that the water temperature varies from 80 to 150 F and that a thermistor and its associated circuitry convert this water temperature to a voltage ranging from 800 to

0

1500mV. Note that the transducer's output is directly proportional to temperature; such that each 1 F produces a 10mV output. Analog-to-digital converter (ADC) and digital-to-converter (DAC) are used to interface a computer to the analog world so that the computer can monitor and control a physical variable Fig. 7.1.

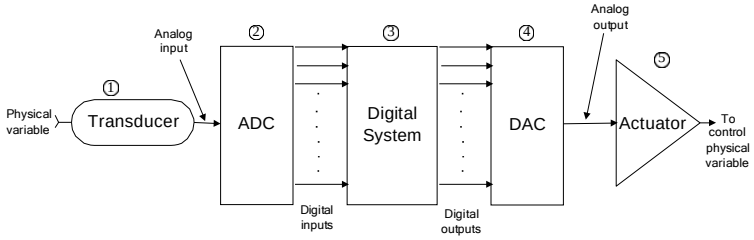


Fig. 7.1 : Interfacing with the analog world using Analog-to-Digital Converter (ADC) and Digital-to-Analog Converter (DAC).

Analog-to-Digital Converter (ADC)

The transducer's electrical analog output serves as the analog input to the ADC. The ADC converts this analog input to a digital output.

The transducer's electrical analog output serves as the analog input to the ADC. The ADC converts this analog input to a digital output. This digital output consists of a number of bits that represent the value of the analog input. For example, the ADC might convert the transducer's 800- to 1500-mV analog values to binary values ranging from 01010000 (80) to 10010110 (150). Note that the binary output from the ADC is proportional to the analog input voltages so that each unit of the digital output represents 10mV.

The digital representation of the analog vales is transmitted from the ADC to the digital computer, which stores the digital value and processes it according to a program of instructions that it is executing.

Digital-to-Analog Converter (DAC)

This digital output from the computer is connected to a DAC, which converts it to a proportional analog voltage or current.

This digital output from the computer is connected to a DAC, which converts it to a proportional analog voltage or current. For example, the computer might produce a digital output ranging from 0000000 to 11111111, which the DAC converts to a voltage ranging from 0 to 10V.

Actuator

The analog signal from the DAC is often connected to some device or circuit that serves as an actuator to control the physical variable.

The analog signal from the DAC is often connected to some device or circuit that serves as an actuator to control the physical variable. For our water temperature example, the actuator might be an electrically controlled valve that regulates the flow of hot water into the tank in accordance with the analog voltage from the DAC. The flow rate would

vary in proportion to this analog voltage, with 0 V producing no flow and 10 V producing the maximum flow rate.

Thus we see that ADCs and DACs function as interfaces between a completely digital system, like a computer, and the analog world.

Digital-to-Analog Conversion

D/A conversion is the process of taking a value represented in digital code.

Basically, D/A conversion is the process of taking a value represented in digital code (such as straight binary or BCD) and converting it to a voltage or current which is proportional to the digital value. Fig. 7.2 shows the symbol for a typical 4-bit D/A converter. Now, we will examine the various input/output relationships.

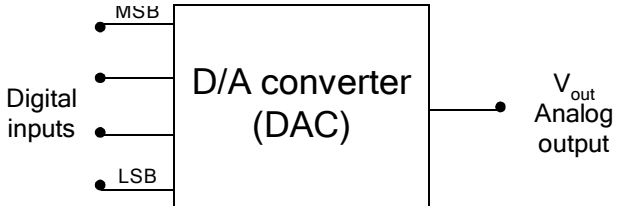


Fig. 7.2 : Four bit DAC with voltage output.

The digital inputs D,C,B, and A are usually derived from the output register of a digital system. The $2^4 = 16$ different binary numbers represented by these 4 bits for each input number, the D/A converter output voltage is a unique value. In fact, for this case, the analog output voltage Vout is equal in volts to the binary number.

In general,
Analog output = K × digital input

where K is the proportionality factor and it is constant value for a given DAC. The analog output can of course be a voltage or current. When it is a voltage, K will be in voltage units, and when the output is current, K will be in current units. For the DAC of K=1 V, so that

V_{OUT} = (1 V) × digital input

We can use this to calculate V_{OUT} for any value of digital input. For example, with a digital input of $1100_2 = 12_{10}$, we obtain

V_{OUT} = 1V × 12 = 12V

Problem 1

Problem 1 and Solution

A 5-bit DAC has a current output. For a digital input of 101000, an output current of 10mA is produced. What will IOUT be for a digital input of 11101?

Solution

The digital input 10100_2 is equal to decimal 20. Since I_{OUT} = 10 mA for this case, the proportionality factor as 0.5 mA. Thus, we can find for a digital input such as $11101_2 = 29_{10}$ as follows :

$I_{OUT} = (0.5mA) \times 29$
$= 14.5$ mA

Remember, the proportionality factor, K, will vary from one DAC to another.

Problem 2

What is the largest value of output voltage from an 8-bit DAC that produces 1.0V for a digital input of 00110010?

Solution

$00110010_2 = 50_{10}$
$1.0\ V = K \times 50$

Therefore,
$K = 20$ mV

The largest output will occur for an input of $11111111_2 = 255_{10}$.

$V_{OUT}(max) = 20mV \times 255$
$= 5.10$ V

Analog Output

Analog Output

The output of a DAC is technically not an analog quantity because it can take on only specific values like the 16 possible voltage levels for Vout. Thus, in that sense, it is actually digital. However, the number of different possible output levels can be increased and the difference between successive values can be decreased by increasing the number of input bits. This will allow us to produce an output that is more and more like an analog quantity that varies continuously over a range of values.

Input Weights

For the DAC of it should be noted that each digital input contributes a different amount to the analog output. This is easily seen if we examine the cases where only one input is HIGH Table 7.1. The contributions of each digital input are weighted according to their position in the binary number.

D	C	B	A		V_{OUT} (V)
0	0	0	1	→	1
0	0	1	0	→	2
0	1	0	0	→	4
1	0	0	0	→	8

Table 7.1

Thus, A, which is the LSB, has a weight of 1V, B has a weight of 2V, C has a weight of 4 V, and D, the MSB, has the largest weight 8V. The weights are successively doubled for each bit, beginning with the LSB. Thus, we can consider V_{OUT} to be the weighted sum of the digital inputs. For instance, to find V_{OUT} for the digital input 0111 we can add the weights of the C, B, and A bits to obtain 4 V + 2V + 1V=7V.

Problem 3

Input Weights
Problem 3 and Solution

A 5-bit D/A converter produces V_{OUT} = 0.2 V for a digital input of 0001. Find the value of Vout for an input of 11111.

Solution

Obviously, 0.2 V is the weight of the LSB. Thus, the weights of the other bits must be 0.4 V, 0.8 V, 1.6 V, and 3.2 V respectively. For a digital input of 11111, then, the value of V_{OUT} will be 3.2 V + 1.6 V+ 0.8V + 0.4V + 0.2 V = 6.2 V.

Resolution

Resolution of a D/A converter is defined as the smallest change that can occur in he analog output as a result of a change in the digital input.

Resolution of a D/A converter is defined as the smallest change that can occur in he analog output as a result of a change in the digital input. We can see that the resolution is 1V, since V_{OUT} can change by no less than 1 V when the digital input value is changed. The resolution is always equal to the weight of the LSB and is also referred to as the step size. As the counter is being continually cycled through its 16 states by the clock signal, the DAC output is a staircase waveform that goes up 1 V per step. When the counter is at 1111, the DAC output is at its maximum value of 15 V; this is its full-scale output. When the counter recycles to

0000, the DAC output returns to 0V. The resolution or step size of the jumps in the staircase waveform; in this case, each step is 1 V.

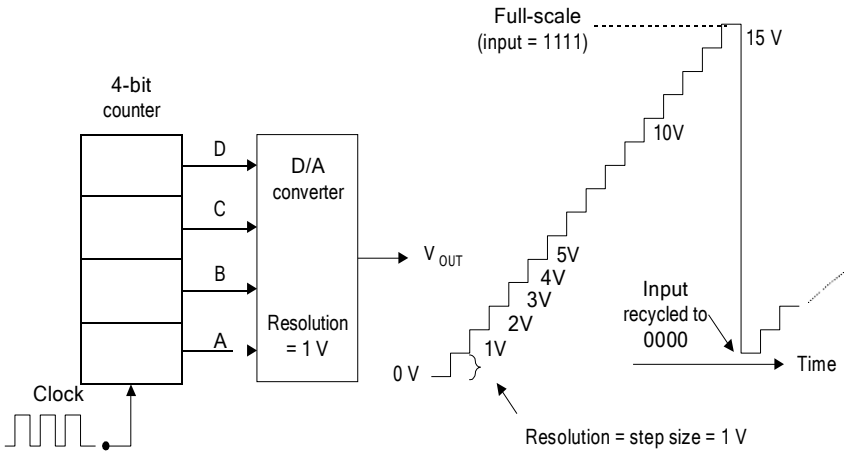


Fig. 7.3 : Output wave forms of a four bit DAC.

Note that the staircase has 16 levels corresponding to the 16 input states, but there are only 15 steps or jumps between the 0-V level and full-scale, In general, for an N-bit DAC the number of different levels will be 2^N, and the number of steps will be $2^N - 1$.

You may have already figured out the resolution (step size) is the same as the proportionality factor in the DAC input/output relationship :

$$\text{analog output} = K \times \text{digital input}$$

A new interpretation of this expression would be that the digital input is equal to the number of the step, K is the amount of voltage (or current) per step, and the analog output is the product of the two.

Problem 4

For the DAC of Example 3 determine V_{OUT} for a digital input of 10001.

Solution

The step size is 0.2 V, which is the proportionality factor K. The digital input is $10001 = 17_{10}$. Thus we have :

$V_{OUT} = (0.2\ V) \times 17$
$= 3.4V$

Percentage Resolution

Although resolution can be expressed as the amount of voltage or current per step, it is also useful to express it as a percentage of the full-scale output. To illustrate, in Fig. 7.3 the DAC has a maximum full-scale output of 15 V (when the digital input is 1111). The step size is 1V, which gives a percentage resolution.

$$\%\ resolution = \frac{step\ size}{full\ scale\ (F.S.)} \times 100\%$$
$$= \frac{1V}{15V} \times 100\% = 6.67\%$$

Percentage Resolution Problem 5 and Solution

Problem 5

A 10-bit DAC has a step size of 10 mV. Determine the full-scale output voltage and the percentage resolution.

Solution

With 10 bits, there will be 2^{10} - 1 = 1023 steps of 10mV each. The full-scale output will therefore be 10mV × 1023 = 10.23 V and

$$\%\ resolution = \frac{10\ mV}{10.23V} \times 100\% \approx 0.1\%$$

Problem 4 helps to illustrate the fact that the percentage resolution becomes smaller as the number of input bits is increased. In fact, the percentage resolution can also be calculated from.

$$\%\ resolution = \frac{1}{total\ number\ of\ \ steps} \times 100\%$$

For an N-bit binary input code the total number of steps is 2^N-1. Thus, for the previous example,

$$\%\ resolution = \frac{1}{2^{10} - 1} \times 100\%$$
$$= \frac{1}{1023} \times 100\%$$
$$\approx 0.1\%$$

This means that it is only the number of bits which determines the percentage resolution. Increase of the number of bits increases the number of steps to reach full scale.

Exercise

Multiple choice questions

a) Each digital input of DAC are weighted according to their position in the

i) binary number
ii) decimal number
iii) hexa- decimal number
iv) octal number.

b) The ADC converts analog input to a digital

i) input
ii) output
iii) number
iv) all of the above.

Questions for short answers

a) What is the function of a transducer and actuator?
b) What do you mean by input weights?
c) Define resolution. What is the full scale output?
d) What is the function of an ADC?
e) What function does a DAC perform?

Analytical question

a) “Five elements are involved when a computer is monitoring and controlling a physical variable that is assumed to be analog.” Illustrate the above situation.

D/A Converter

Learning Objectives

On completion of this lesson you will be able to :

- design different types of D/A converter circuit and describe their operation.
- Understand the advantages, disadvantages, and limitation of several typesof digital-to-analog converters (DAC).

D/A Converter

The purpose of a digital-to-analog converter is to convert a binary word to a proportional current or voltage.

The purpose of a digital-to-analog converter is to convert a binary word to a proportional current or voltage.

The binary weighted resistors produce binary-weighted current which are summed up by the op-amp to produce proportional output voltage. The binary word applied to the switches produces a proportional output voltage.

Several different binary codes such as straight binary, BCD and offset binary are commonly used as inputs to D/A converters.

D/A-Converter Circuitry

D/A-Converter Circuitry

There are several methods and circuits for producing the D/A operation. We shall examine several of the basic schemes, to gain an insight into the ideas used.

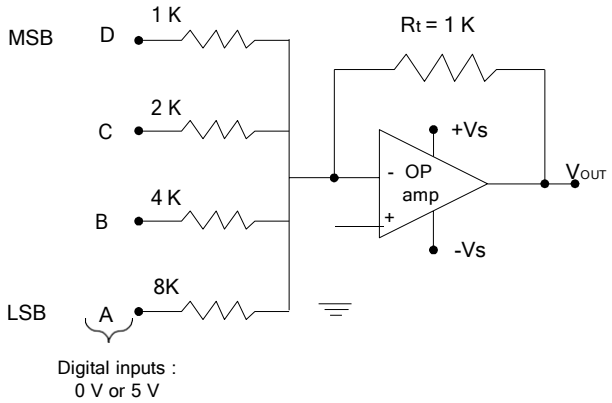


Fig. 7.4 : DAC circuitry using op-amp with binary weighted resistors.

Fig. 7.4 shows the basic circuit of 4-bit DAC. The inputs A,B,C, and D are binary inputs which are assumed to have values of either 0 V or 5 V. The operational amplifier is employed as a summing amplifier, which produces

the weighted sum of these input voltages. the summing amplifier multiplies each input voltage by the ratio of the feedback resistor R_F to the corresponding input resistor R_{IN}. In this circuit R_F R_{IN} 1kΩ and the input resistors range from 1 to 8 kΩ . The D input has R_{IN} = 1KΩ, so the summing amplifier passes the voltage at D with no attenuation. The C input has R_{IN} = 2 kΩ, so that it will be attenuated $_1$ by. Similarly, the B input will be attenuated by ¼ and the A input by $/_8$. The amplifier output can thus be expressed as

$$V_{OUT} = - (V_D + {}^1/_2\ V_C + ¼\ V_B + {}^1/_8\ V_A)$$

The negative sign is present because the summing amplifier is a polarity-inverting amplifier, but it will but concern us here.

Clearly, the summing amplifier output is an analog voltage which represents a weighted sum of the digital inputs. The output is evaluated for any input condition by setting the appropriate inputs to either 0 V or 5 V. For example, if the digital input is 1010, then $V_D = V_B = 5V$ and $V_C = V_A = 0V$. Thus, using equation

$$V_{OUT} = - (5V + OV + ¼ \times 5V + OV)$$
$$= - 6.25V$$

The resolution of this D/A converter is equal to the weighting of the lSB, which is $^1/_8 \times 5V = 0.625$ V. The analog output increases by 0.625 V as the binary input number advances one step.

Problem 6

Problem 6 Solution

Assume V_{REF} = 10 V and R = R = 10 kΩ. Determine the resolution and full-scale output for this DAC. Assume that R_L is much smaller than R.

Solution

$I_0 = V_{REE}/R = 1$ mA. This is the weight of the MSB. The other three currents will be 0.5, 0.25, and 0.125 mA. The LSB is 0.125 mA, which is also the resolution.

The full-scale output will occur when the binary inputs are all HIGH so that each current switch is closed and

$$I_{OUT} = 1 + 0.5 + 0.25 + 0.125 = 1.875 \text{ mA}$$

Note that the output current is proportional to V_{REF}. If V_{REF} is increased or decreased, the resolution and full-scale output will change proportionally.

R/2R Ladder

R/2R Ladder

The DAC circuits we have looked at, has some practical limitations. The biggest problem is the large difference in resistor values between the LSB and MSB, especially in high-resolution DACs. One of the most widely used DAC circuits that uses resistance's fairly close in value is the R/2R ladder network. Here the resistance values span a range of only 2 to 1.

Note, how the resistors are arranged, and only two different values are used, R and 2R. The current I_{OUT} depends on the positions of the four switches, and the binary inputs $B_3B_2B_1B_0$ control the states of the switches. This current is allowed to flow through an op-amp current-to-voltage converter to develop V_{OUT}. It can be shown that the value of V_{OUT} is given by the expression.

$$V_{OUT} = \frac{-V_{REF}}{8} \times B.$$

where B is the value of the binary input, which can range from 000 (0) to 1111 (15).

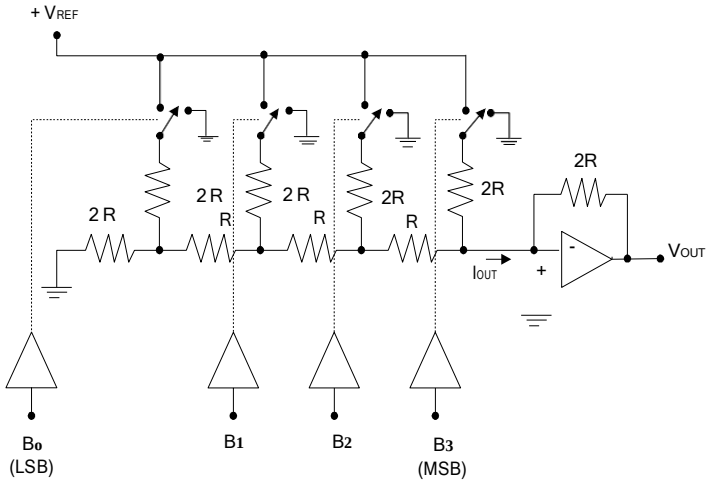


Fig. 7.5 : R/2R ladder DAC.

DAC Specifications

Resolution

Resolution
Accuracy

As mentioned earlier, the percentage resolution of a DAC is dependent on the number of bits. A 10-bit DAC has a finer (smaller) resolution than an 8-bit DAC.

Accuracy

There are several ways of specifying accuracy. The two most common are called full-scale error and linearity error. which are normally expressed as a percentage of the converter's full-scale output (%F.S.)

Full-scale error is the maximum deviation of the DAC's output from its expected (ideal) value, expressed as a percentage of full scale. For example, assume that the DCA has an accuracy of ± 0.01% F.S. Since this converter has a full-scale output of 9.375 V, this percentage converts to

$$\pm 0.01\% \times 9.375 \text{ V} = \pm 0.9375 \text{ mV}$$

This means that the output of this DAC can, at any time, be off by as much as 0.9375mV from its expected value.

Linearity error is the maximum deviation in step size from the ideal step size. For example, the DAC has an expected step size of 0.625 V. If this converter has a linearity error of ± 0.01F.S,, this would mean that the actual step size could be off by as much as 0.9375 mV.

Problem 7

Example 7 and Solution

A certain 8-bit DAC has a full-scale output of 2mA and a full-scale error of ± 0.5% F.S. What is the range of possible outputs for an input of 10000000?

Solution

The step size is 2mA/255 = 7.84 µA. Since 10000000 = 128_{10}, the ideal output should be 128 × 7.84 µA. The error can be as much as

$$\pm 0.5\% \times 2\text{mA} = \pm 10\mu\text{A}$$

Thus, the actual output can deviate by this amount from the ideal 1004µA , so the actual output can be anywhere from 994 to 1014 µA.

Offset Error

Offset Error

Ideally, the output of a DAC will be zero volts when the binary input is all 0's. In practice, however, there will be a very small output voltage for this situation; this is called offset error. This offset error, if not corrected, will be added to the expected DAC output for all input cases. Offset error can be negative as well as positive.

Many DACs will have an external offset adjustment that allows you to zero the offset. This is usually accomplished by applying all 0s to the DAC input and monitoring the output while an offset adjustment potentiometer is adjusted until the output is as close to 0 V as required.

Settling Time

Settling Time

The operating speed of a DAC is usually specified by giving its settling time, which is the time required for the DAC output to go from zero to full scale as the binary input is changed from all 0's to all 1's. Typical values for settling time range from 50 ns to 10 µs.

Monotonicity

A DAC is monotonic if its output increases as the binary input is incremented from one value to the next. Another way to describe this is that the staircase output will have no downward steps as the binary input is incremented from zero to full scale.

Monotonicity
DAC Applications
Control

DAC Applications

DACs are used whenever the output of a digital circuit has to provide an analog voltage or current to drive an analog device. Some of the most common applications are described in the following paragraphs.

Control

The digital output from a computer can be converted to an analog control signal to adjust the speed of a motor or the temperature of a furnace, to control almost any physical variable.

Automatic Testing

Automatic Testing
Signal Reconstruction

Computers can be programmed to generate the analog signals (through a DAC) needed to test analog circuitry. The test circuit's analog output response will normally be converted to a digital value by an ADC and fed into the computer to be stored, displayed, and sometimes analyzed.

Signal Reconstruction

In many applications, an analog signal is digitized, meaning that successive points on the signal are converted to their digital equivalent and stored in memory. This conversion is performed by an analog-to-digital converter (ADC). A DAC can then be used to convert the stored digitized data back to analog-one point at a time-thereby reconstructing the original signal. This combination of digitizing and reconstruction is

used in digital storage oscilloscopes, audio compact disk systems, and digital audio and video recording.

Exercise

Multiple choice questions

a) Resolution of DAC is equal to the weight of

i) LSB
ii) MSB
iii) full scale output
iv) 1 volt.

b) When the binary input is all 0.s, ideally the output of a DAC will be?

i) Zero volt
ii) Full scale output voltage
iii) 1 volt
iv) One step voltage.

Questions for short answers

a) Define full scale error and offset error.
b) What is the advantage of R/2R ladder DAC over the DAC that uses binary weighted resistors?
c) An 8-bit DAC has an output of 3.92 mA for an input of 01100010. What are the DAC's resolution and full-scale output?
d) What is the percentage resolution of an 8-bit DAC?
e) How many different output voltages can a 12-bit DAC produce?
f) Define full-scale error.
g) What is settling time?
h) Describe offset error and its effect on a DAC output.

Analytical questions

a) Describe the operation of a DAC.
b) What is the advantage of R/2R ladder DACs over those that use binary weighted resistors?
c) Discuss some of the DAC applications.

A/D Converter

Learning Objectives

On completion of this lesson you will be able to :

- understand the operation of various types of analog-to-digital converter circuitry such as the single ramp A/D, the digital ramp A/D circuit.
- compare the advantages and disadvantages of different analog-to-digital converter circuits

Analog-to-Digital Conversion

An analog-to-digital converter takes an analog input voltage and after a certain amount of time produces a digital output code which represents the analog input.

An analog-to-digital converter takes an analog input voltage and after a certain amount of time produces a digital output code which represents the analog input. The A/D conversion process is generally more complex and time-consuming than the D/A process. The techniques that are used provide and insight into what factors determine an ADCs performance.

Several important types of ADC utilize a DAC as part of their circuitry. Fig. 7.6 is a general block diagram for this class of ADC. The timing for the operation is provided by the input clock signal. The control unit contains the logic circuitry for generating the proper sequence of operations. The START COMMAND, initiates the conversion process, The op-amp compactor has two analog inputs and a digital output that switches states, depending on which analog input is greater.

The basic operation of ADC.

The basic operation of ADCs of this type consists of the following steps:

1. The START COMMAND pulse initiates the operation.
2. At a rate determined by the clock, the control unit continually modifies the binary number that is stored in the register.
3. The binary number in the register is converted to an analog voltage, V_{AX}, by the DAC.

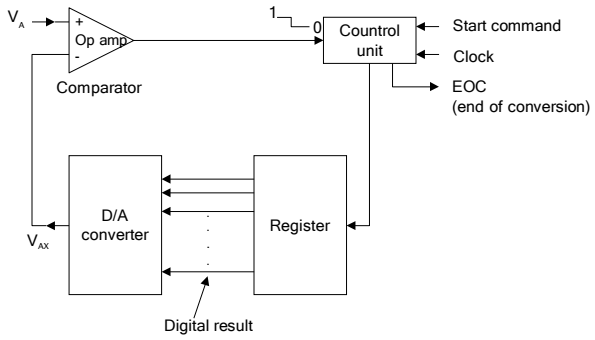


Fig. 7.6 : Basic diagram of ADC.

4. The comparator compares V_{AX} with the analog input V_A. As long as $V_{AX} < V_{A1}$ the comparator output stays HIGH. When V_{AX} exceeds V_A by at least an amount = V_T (threshold voltage), the comparator output goes LOW and stops the process of modifying the register number. At this point, V_{AX} is a close approximation to V_A. The digital number in the register, which is the digital equivalent of V_{AX}, is also the approximate digital equivalent of V_{A1} within the resolution and accuracy of the system.
5. The control logic activates the end-of-conversion signal, EOC, when the conversion is complete.

Digital-Ramp ADC

Operation procedure of a digital-ramp ADC.

One of the simplest versions of the general ADC of Fig. 7.7 uses a binary counter as the register and allows the clock to increment the counter one step at a time until $V_{AX} \geq V_A$. It is called a digital-ramp ADC because the wave form at V_{AX} is a step-by-step ramp (actually a staircase) like the one shown in Fig. 7.7. It is also referred to as a counter-type ADC. Fig. 7.7 is the diagram for a digital-ramp ADC. It contains a counter, a DAC, an analog comparator, and a control AND gate. The comparator output serves as the active-LOW end-of-conversion signal, $\overline{EOC}$. If we assume that V_A, the analog voltage to be converted, is positive, the operation proceeds as follows :

1. A START pulse is applied to reset the counter to zero. The HIGH at START also inhibits clock pulse form passing through the AND gate into the counter.
2. With all 0's at its input, the DAC's output will be V_{AX} = 0V.
3. Since $V_A > V_{AX}$, the comparator output, $\overline{EOC}$, will be HIGH.
4. When START returns LOW, the AND gate is enabled and clock pulses get through to the counter.
5. As the counter advances, the DAC output, V_{AX}, increases one step at a time as shown in Fig. 7.7.
6. This continues unit V_{AX} reaches a step that exceeds V_A by an amount equal to or greater than V_T (typically 10 to 100 μV). At this point, $\overline{EOC}$ will go LOW and inhibit the flow of pulses into the counter and the counter will stop counting.
7. The conversion process is now complete as signaled by the HIGH-to-LOW transition at $\overline{EOC}$, and the contents of the counter are the digital representation of V_A.
8. The counter will hold the digital value until the next START pulse initiates a new conversion.

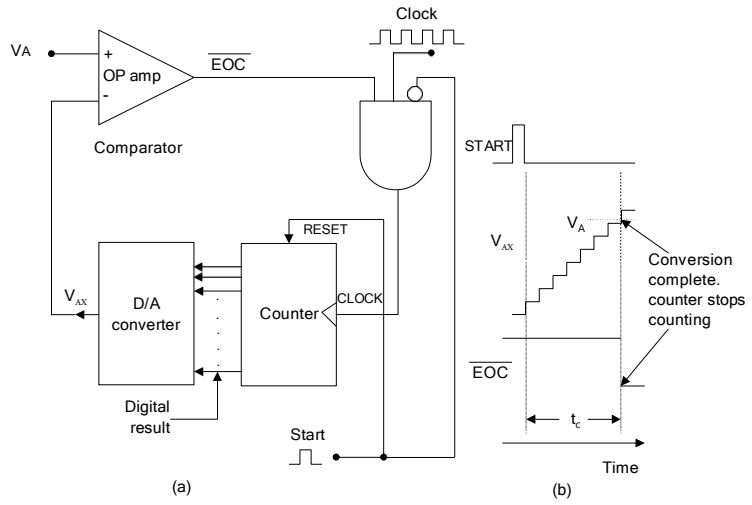


Fig. 7.7 : Digital-ramp ADC.

Problem 8

Assume the following values for the ADC clock frequency = 1 MHz; V_T = 0.1 mV; DAC has F.S. output = 10.23 V and a 10-bit input. Determine the following values.

Problem 8 and Solution

a. The digital equivalent obtained for V_A = 3.728 V.
b. The conversion time.
c. The resolution of this converter.

Solution

a. The DAC has a 10-bit input and a 10.23-V F.S. output. Thus, the number of total possible steps is $2^{10} - 1 = 1023$, and so the step size is

$$\frac{10.23V}{1023} = 10mV$$

This means that V_{AX} increases in steps of 10 mV as the counter counts up from zero. Since V_A = 3.728 V and V_T = 0.1 mV, V_{AX} has to reach 3.7281 V or more before the comparator switches LOW. This will require.

$$\frac{3.7281V}{10mV} = 372.81 = 373\ steps$$

At the end of the conversion, then, the counter will bold the binary equivalent of 373, which is 0101110101. This is the desired digital equivalent of $V_A = 3.728$ V, as produced by this ADC.

b. Three hundred seventy-three steps were required to complete the conversion. Thus, 373 clock pulses occurred at the rate of one per microsecond. This gives a total conversion time of 373 μs.
c. The resolution of this converter is equal to step size of the DAC, which is 10mV. In percent it is $1/1023 \times 100\% \approx 0.1\%$.

Problem 9

Problem 9 and Solution

For the same ADC of problem 8 determine the approximate range of analog input voltages that will produce the same digital result of $0101110101_2 = 373_{10}$.

Solution

Table 7.2 shows the ideal DAC output voltage, V_{AX}, for several of the steps on and around the 373rd. If V_A is slightly smaller than 3.72 V (by an amount < V_T),

Step	V_{AX} (V)
371	3.71
372	3.72
373	3.73
374	3.74
375	3.75

Table 7.2

Then $\overline{EOC}$ won't go LOW when V_{AX} reaches the 3.72-V step, but will go LOW on the 3.73-V step. If V_A is slightly smaller than 3.73 V (by an amount < V_T), then $\overline{EOC}$ won't go LOW until V_{AX} reaches the 3.74-V step. Thus, as long as V_A is between approximately 3.72 V and 3.73-V, $\overline{EOC}$ will go LOW when V_{AX} reaches the 3.73-V step. The exact range of V_A values is

$$3.72 \text{ V} - V_T \text{ to } 3.73 \text{ V} - V_T$$

but since V_T is so small, we can simply say that the range is approximately 3.72 V to 3.73 V - a range equal to 10 mV, the DAC's resolution.

A/D Resolution and Accuracy

Resolution of the ADC is qual to the resolution of he DAC that it contains.

Resolution of the ADC is equal to the resolution of the DAC that it contains. The DAC output voltage V_{AX} is a staircase waveform that goes up in discrete steps until it exceeds V_A. Thus, V_{AX} is an approximation to the value of V_A, and the best we can expect is that V_{AX} is within 10 mV of V_A if the resolution (step size) is 10 mV. We can think of the resolution as being a built-in error that is often referred to as

quantization error. This quantization error, can be reduced by increasing the number of bits in the counter and DAC.

Problem 10

Problem 10 and Solution

Acertain 8-bit ADC hs a full-scale input of 2.55 V (i.e., V_A = 2.55 V produces a digital output of 11111111). It has a specified error of 0.1% F.S. Determine the maximum amount by which the V_{AX} output can differ from the analog input.

Solution

The step size is 2.55 V/ (2^8- 1), which is exactly 10 mV. This means that even if the DAC has no inaccuracies, the V_{AX} output could be off by as much as 10 mV because V_{AX}. can change only in 10-mV steps; this is the quantization error. The specified error of 0.1% F.S. is 0.1% × 2.55 V = 2.55 mV. This means that the V_{AX} value can be off by as much as 2.55 mV because of component inaccuracies. Thus, the total possible error could be as much as 10 mV + 2.55 mV = 12.55 mV.

Conversion Time, TC

Conversion Time, TC

The conversion time is the time interval between the end of the START pulse and the activation of the $\overline{EOC}$ output. The counter starts counting from zero and counts up until V_{AX} exceeds V_A, at which point $\overline{EOC}$ goes LOW to end the conversion process. It should be clear that the value of conversion time, to, depends on V_A. A larger value will require more steps before the staircase voltage exceeds V_A.

The maximum conversion time will occur when V_A is just below full scale so that V_{AX} has to go to the last step to activate $\overline{EOC}$. For an N-bit converter this will be

$$tc\ (max) = 2^N - 1 \text{ clock cycles}$$

Sometimes, average conversion time is specified; it is half of the maximum conversion time.

$$tc(avg) = \frac{tc(\max)}{2} \approx 2^{N-1}\ clock\ cyles$$

The major disadvantage of the digital-ramp method is that conversion time essentially doubles for each bit that is added to the counter, so that resolution can be improved only at the cost of a longer tc. Applications,

however, the relative simplicity of the digital-ramp converter is an advantage over the more complex, higher-speed ADCs.

Applications

Applications

Almost any measurable quantity present as a voltage can be digitized by an A/D converter and displayed. A/D converters are the heart of digital voltmeters and digital MultiMate's. Analog voice signals are converted to digital form for transmission over long distances. At their destination they are reconverted to analog. In digital audio record- the analog audio signal produced by a microphone is digitized (using an ADC), then stored on some medium such as magnetic tope, magnetic disk or optical disk. Later the stored data are played back by sending them to a DAC to reconstruct the analog signal, which is fed to the amplifier and speaker system to produce the recorded sound.

Exercise

Multiple choice question

a) The number of total steps of a 9-bit ADC is,

i) 255
ii) 256
iii) 511
iv) 512.

Questions for short answers

a) What do you know about quantization error?
b) Define conversion time.
c) What is the major disadvantage of the digital ramp type ADC?
d) What is the function of the comparator in the ADC?
e) What is the function of the comparator in the ADC?
f) Where is the approximate digital equivalent of V_A when the conversion is complete?
g) What is the function of the $\overline{EOC}$ signal?

Analytical question

a) Draw digital ramp ADC and write down its operation.

Semiconductor Memories: RAMs and ROMs

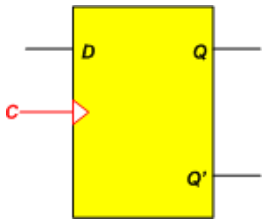
Lesson Objectives:

In this lesson you will be introduced to:

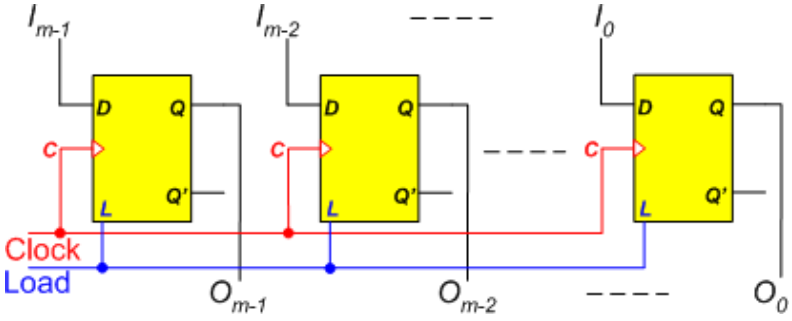
- Different memory devices like, ***RAM, ROM, PROM, EPROM, EEPROM,*** etc.
- Different terms like: ***read, write, access time, nibble, byte, bus, word, word length, address, volatile, non-volatile etc.***
- How to implement combinational and sequential circuits using ROM.

Introduction:

The smallest unit of information a digital system can store is a ***bit***, which can be stored in a ***flip-flop*** or a ***1-bit register***.



To store *m* bits of data, an ***m-bit register*** with parallel load capability may be used. Data available on the ***m***-bit input lines (I_0 to I_{m-1}) may be stored/*written* into this register under control of the clock by asserting the "*Load*" control input. The stored ***m*** bits of data may be *read* from the register outputs (O_0 to O_{m-1}).

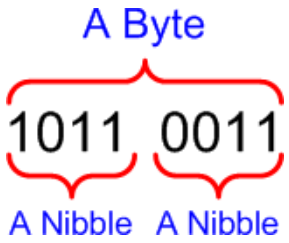


The ***m*** bits of data stored in a register make up a ***word***. It is simply a number of bits operated upon or considered by the hardware as a group. The number of bits in the word, ***m,*** is called ***word length***.

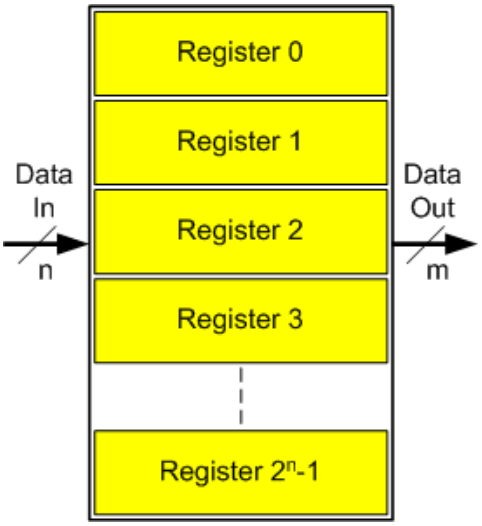
The *m* inputs of the register are provided through an *m*-bit input data *bus* and *m* outputs by an *m*-bit output data *bus*.

A ***bus*** is a number of signal lines, grouped together because of similarity of function, which connect two or more systems or subsystems.

A unit of 8-bits of information is referred to as a ***byte***, while 4-bits of information is referred to as a ***nibble***.



A ***memory*** device can be looked at as consisting of a number of equally sized registers sharing a *common set of inputs*, and a *common set of outputs*, as shown in the Figure.



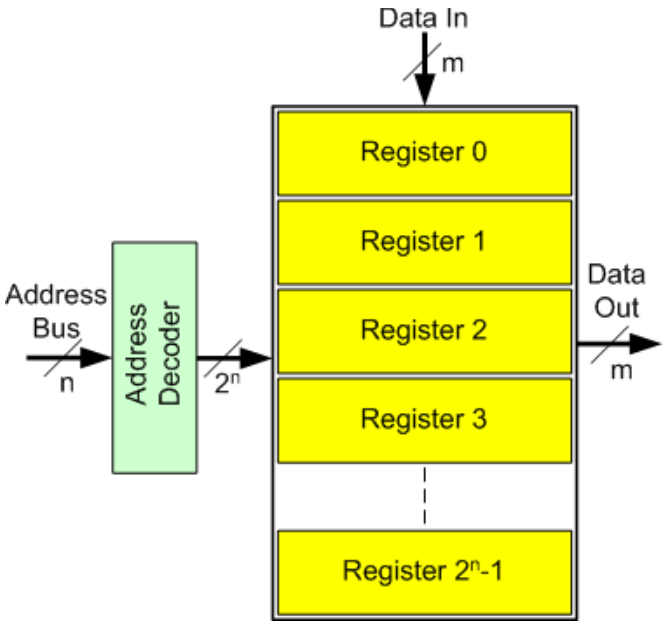
Storing data in a memory register is referred to as a memory ***write*** operation and looking up the contents of a memory register is referred to as a memory ***read*** operation.

In case of a write operation, the input data need to be written into one *particular* register in the memory device.

Since the input data lines are common to all registers of the memory device, only the selected register should have its ***load*** control signal asserted while the other registers should not.

If the number of registers is 2^n, n lines will be required to select the register to be written into. The n-lines are used as an input to a decoder where the decoder's 2^n outputs may be used as the ***load*** control inputs to the 2^n registers.

The load control signal of a particular register is asserted by a unique combination of the n-select lines. This unique combination is considered as the ***address*** for that particular register.

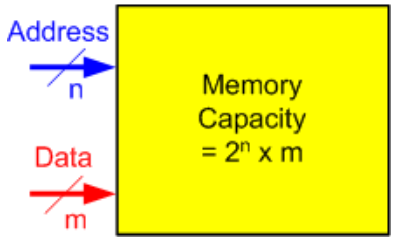


Thus, a memory device can be thought of as a collection of **addressable** registers.

A read or a write operation into the memory device has to specify the address of the particular register to be read or written into.

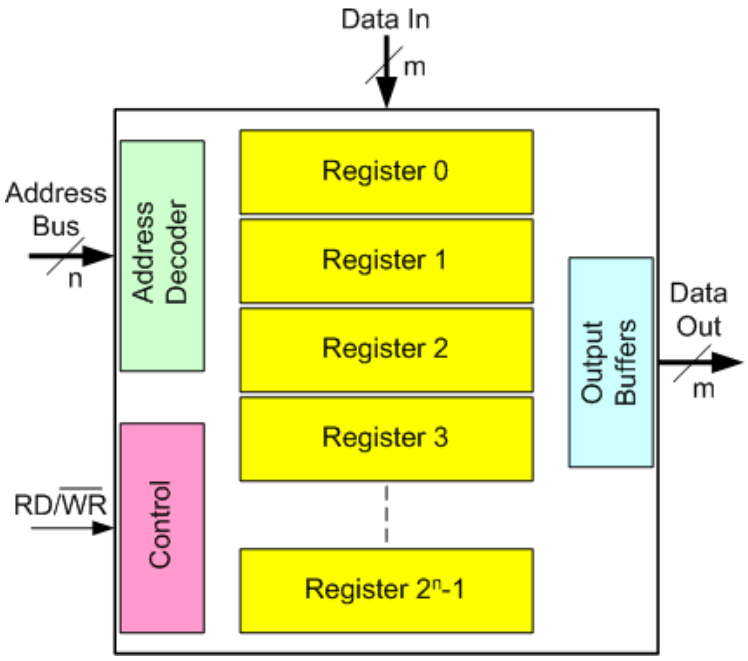
The ***capacity*** of the memory is specified in terms of the number of bits or the number of words available in this memory device.

For a memory device with ***n***-bit address lines and word (register) size of ***m***-bits, the memory has 2^n words (storage locations/registers) each having ***m*** bits for a total capacity of $2^n \times m$ bits.



For example, if $n = 10$ and $m = 8$, the memory is a **"1024 x 8"** bit memory. Alternatively, it is said that the memory has 1K bytes.

A block diagram of the memory device is shown in the figure. The address inputs are decoded by **address decoder** to select one, and only one, of the memory words (registers), either for reading or writing.



The $RD/\overline{WR}$ line is a control signal that determines the type of operation to be performed; a read operation or a write operation.

$RD/\overline{WR} = 1$ indicates a read operation, while $RD/\overline{WR} = 0$ indicates a write operation.

To **read** the memory contents stored in a particular word, the address of this word is applied, and logic 1 is applied to the $RD/\overline{WR}$ line that enables the output buffers of the memory.

To **write** at a location, the address of the location to be written is provided at the address inputs, data is provided at the data inputs, and logic 0 is applied to the $RD/\overline{WR}$ line.

There is a time delay between the application of an address and the appearance of contents at the output, this is called the memory ***access time***. This depends both on the technology and on the structure used to implement the memory.

Random Access Memory (RAM):

For the shown above memory structure, the access time is independent of the sequence in which addresses are applied.

Such a memory is called ***random access memory (RAM).*** Thus, the contents of any one location can be accessed in essentially the same time as can the contents of any other location chosen at random.

RAMs are ***volatile*** memories that will only retain the stored data as long as power is **ON** but will lose this data when power is turned **OFF**.

RAMs are classified into two main categories: Static RAM ***(SRAM)*** and Dynamic RAM ***(DRAM).*** These will be studied in greater details in future courses.

Read Only Memory (ROM):

Read Only Memory (ROM) is memory whose stored data can only be read but cannot be re-written (altered).

It is a device in which "permanent" binary information has been stored.

ROMs are ***nonvolatile*** where stored data are not lost even when power is turned **OFF**.

The Figure shows a block diagram of a ROM.

Like RAMS, a ROM has ***n*** *address* inputs and ***m*** outputs. This corresponds to 2^n memory words each of *m* storage bits for a total capacity of $2^n \times m$ bits.

Unlike RAMs, ROMs do not have data input lines, because they do not have a write operation.

ROMs are common to use in storing system-level programs that should be available at all times.

The most common example is the PC system BIOS (Basic Input Output System), which is stored in a ROM called the *system BIOS ROM*.

Several classes of ROMs are in common use. These may be categorized according to their fabrication technologies that influence the way data are introduced into the ROM. The process of storing the desired data into the ROM is referred to as *ROM programming*.

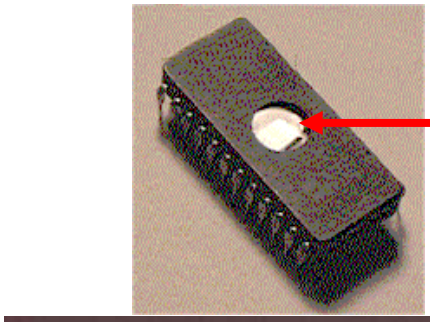
Types of ROMs:

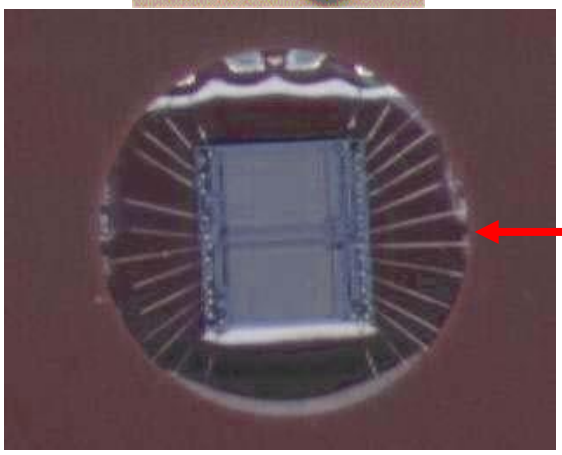
Following are the different types of ROMs.

1. Programming is done by the manufacturer during the last fabrication steps according to the truth table provided by the customer. This type is known as mask programmable ROMs or simply ***ROM***. Data stored this way can never be altered.

2. ROM is provided with fuses to allow users to introduce the desired data by electrically blowing some of these fuses. This type is referred to as a *programmable ROM*, or ***PROM***. Fuse blowing is irreversible and, once programmed the ROM stored pattern cannot be altered.

3. The ROM uses *erasable floating-gate* memory cells that allow erasure of the stored data by Ultra-Violet light. In this type, programming is performed *electrically* by the user using special hardware programmers. Data, thus stored, can later be erased globally (all memory bits = 1) by exposing the memory array to UV-light. This ROM type is referred to as UV-*erasable, programmable ROM*, or simply ***EPROM***. The EPROM IC package is provided with a quartz window to allow UV-light penetration to the memory array.





4. When special electrically erasable memory cells are used, the ROM can be electrically erased at the byte level. Thus individual bytes may be addressed and programmed or erased as desired. This type is referred to as *electrically erasable, programmable ROM*, or ***EEPROM*** or ***E^2PROM***. The ***E^2PROM*** technology is an expensive low-capacity technology and is thus not used for high density or low-cost applications.

5. The most recent ROM technology is the *flash* technology that combines the low-cost and high-density advantages of the UV-EPROM technology and the flexibility of electrical erase of ***E^2PROM*** technology. This technology is electrically erasable but the erasure is performed either globally (the full array) or partially on complete sub-arrays (sectors).

Combinational Circuit Implementation Using ROM:

ROM devices can be used to implement complex combinational circuits directly from truth tables without need for minimization.

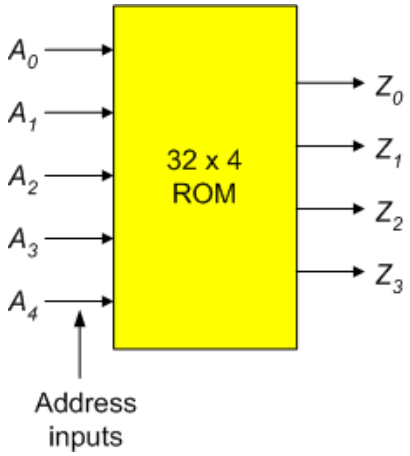
For an ***n***-input, ***m***-output combinational circuit, a 2^n **x** ***m*** ROM is needed (2^n words each of ***m*** storage bits). The designer needs only to specify a ROM table that gives the information stored in each of the 2^n words.

When a combinational circuit is implemented using a ROM, the function may either be expressed in the sum of minterms form, or using a truth table.

As an example, the ROM shown in the figure may be considered as a combinational circuit with four outputs, each a function of the five input variables.

Outputs $Z_0 – Z_3$ can be expressed as sum of minterms as follows:

$Z_0 (A_4, A_3, A_2, A_1, A_0) = \sum m\ (2, 3, 18, 21, 31)$
$Z_1 (A_4, A_3, A_2, A_1, A_0) = \sum m\ (0, 1, 17, 25, 31)$
$Z_2 (A_4, A_3, A_2, A_1, A_0) = \sum m\ (1, 6, 11, 29, 30)$
$Z_3 (A_4, A_3, A_2, A_1, A_0) = \sum m\ (7, 8, 16, 28, 29)$



Example 1:

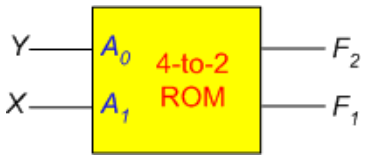
Consider a combinational circuit which is specified by the following two functions:

$F_1 (X, Y) = \sum m\ (1, 2, 3)$
$F_2 (X, Y) = \sum m\ (0, 2)$

The truth table for this circuit is as shown.

X	Y	F_1	F_2
0	0	0	1
0	1	1	0
1	0	1	1
1	1	1	0

In this example, the ROM that implements the two combinational functions must have two address inputs and two outputs. Thus, its size must be 4 x 2 (since 2^n **x** ***m*** is the size of ROM).



The ROM table for this example is as shown.

ROM Address		Stored Information	
A_1	A_0	F_1	F_2
0	0	0	1
0	1	1	0
1	0	1	1
1	1	1	0

Example 2:

Design a combinational circuit using a ROM. The circuit accepts a 3-bit number and generates an output binary number that is equal to the square of the input number.

The first step is to derive the truth table for the combinational circuit as shown. Three inputs and six outputs are needed to accommodate all possible numbers.

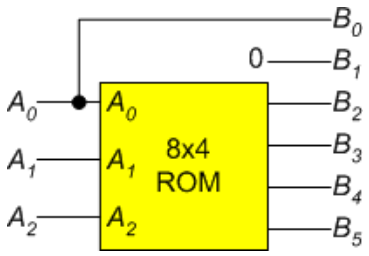
Inputs			Outputs						
A_2	A_1	A_0	B_5	B_4	B_3	B_2	B_1	B_0	Decimal
0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	1	1
0	1	0	0	0	0	1	0	0	4
0	1	1	0	0	1	0	0	1	9
1	0	0	0	1	0	0	0	0	16
1	0	1	0	1	1	0	0	1	25
1	1	0	1	0	0	1	0	0	36
1	1	1	1	0	0	0	0	1	49

By observation, we note that output B_0 is always equal to input A_0, and output B_1 is always 0. Thus, there is no need to store B_0 and B_1 in the ROM. We actually need to only store values of the four outputs (B_5 through B_2) in the ROM.

The table shown specifies all the information that needs to be stored in the ROM, and figure shows the required connections of the combinational circuit. The output B_1 is connected to logic 0 and output B_0 is connected to A_0 always to get $B_1 = 0$ and $B_0 = A_0$.

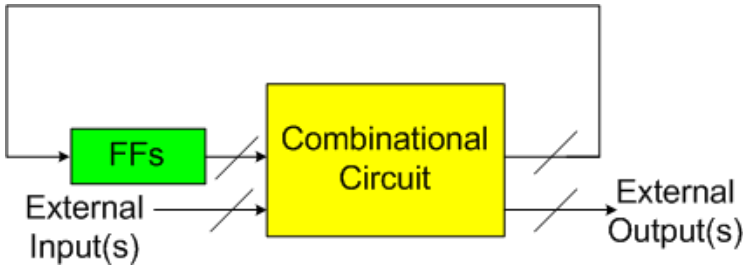
ROM Address			Stored Information			
A_2	A_1	A_0	B_5	B_4	B_3	B_2
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	1
0	1	1	0	0	1	0
1	0	0	0	1	0	0
1	0	1	0	1	1	0
1	1	0	1	0	0	1
1	1	1	1	0	0	0

The minimum size ROM needed must have three inputs and four outputs, for a total of 8 x 4 = 32 bits.

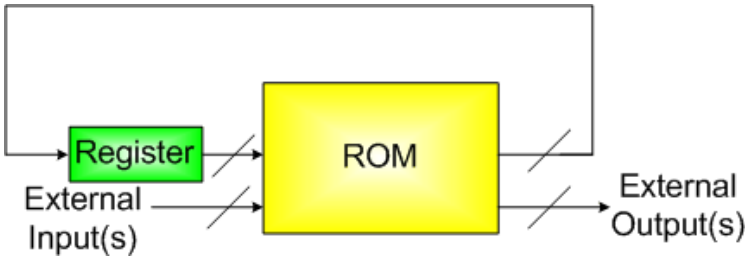


Synchronous Sequential Circuit Implementation Using ROM:

The block diagram of a sequential circuit is shown in the figure.



Since ROM can implement combinational logic, so this part can be replaced by a ROM and Flip-Flops can be replaced by a register as shown in the figure.



Example 3:

Design a sequential circuit whose state transition table is given, using a ROM and a register.

Present State		Input	Next State		Output
Q_2	Q_1	X	Q_2^+	Q_1^+	Y
0	0	0	0	0	0
0	0	1	0	1	0
1	0	0	0	1	0
1	0	1	0	0	1
0	1	0	1	0	0
0	1	1	0	1	0
1	1	0	1	1	0
1	1	1	0	0	1

The next-state and output information are obtained from the table as:

$Q_1^+ = \sum m\ (1, 2, 5, 6)$

$Q_2^+ = \sum m\ (4, 6)$

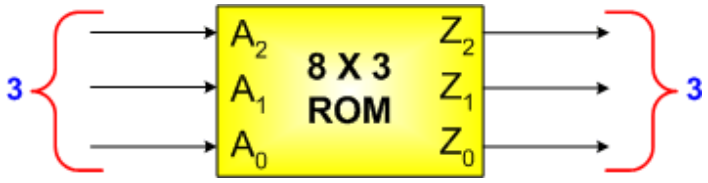
$Y\ (Q_2, Q_1, X) = \sum m\ (3, 7)$

The ROM can be used to implement the combinational circuit and register will provide the flip-flops.

The **number of address inputs** to the ROM is equal to the number of flip-flops plus the number of external inputs.

The **number of outputs** of the ROM is equal to the number of flip-flops plus the number of external outputs.

In this example, 3 inputs and 3 outputs of the ROM are required; so its size must be 8 x 3.



The ROM table is identical to the state transition table with Present State and Inputs specifying the **address** of ROM and Next State and Outputs specifying the ROM **outputs (stored information)**. It is shown below:

ROM Address			Stored Information		
A_2	A_1	A_0	Z_2	Z_1	Z_0
0	0	0	0	0	0
0	0	1	0	1	0
1	0	0	0	1	0
1	0	1	0	0	1
0	1	0	1	0	0
0	1	1	0	1	0
1	1	0	1	1	0
1	1	1	0	0	1

The next state values must be connected from the ROM outputs to the register inputs as shown in the figure below.