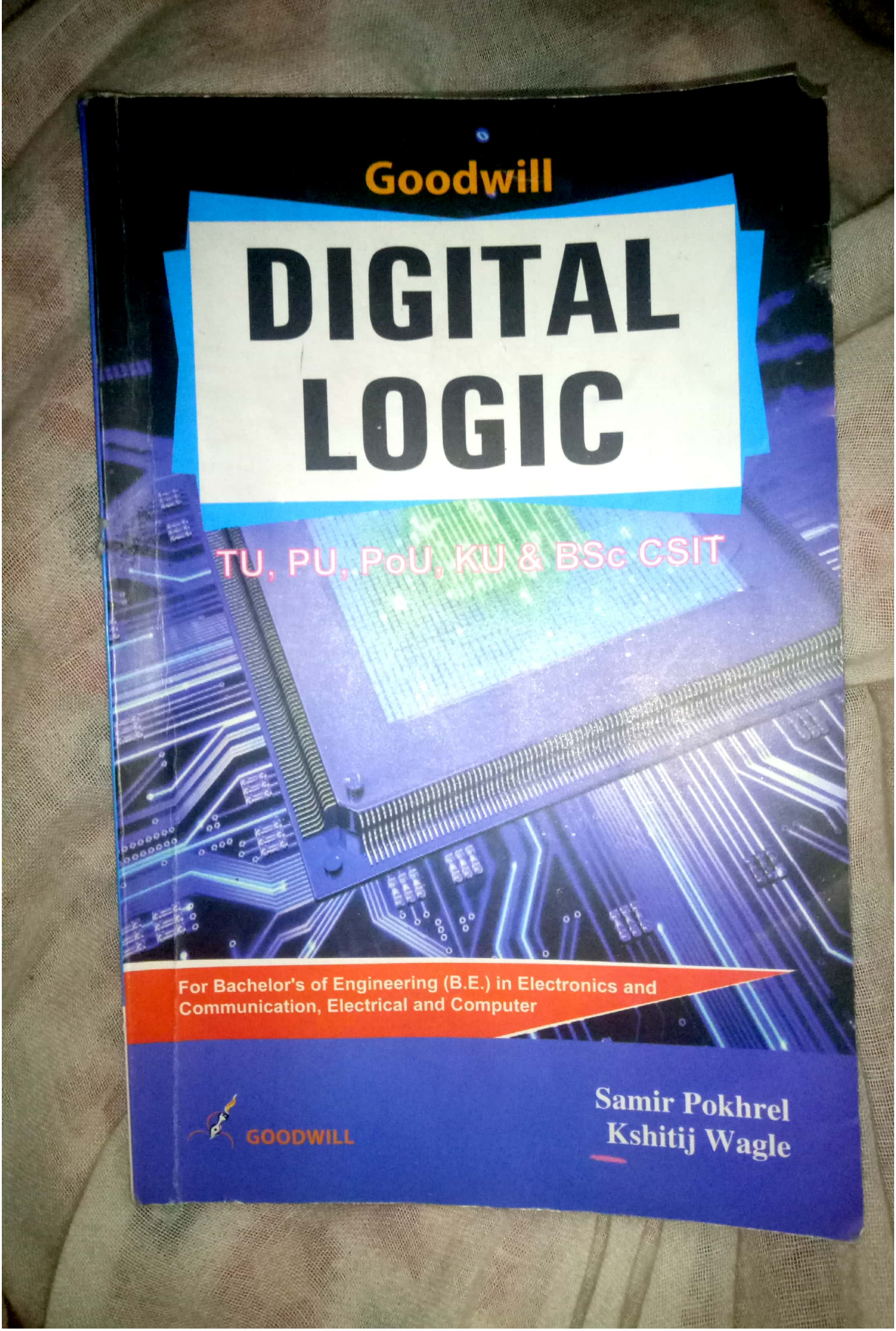


For Bachelor's of Engineering (B.E.) in Electronics and Communication, Electrical and Computer

Digital Logic

TU, PU, PoU, KU & BSc CSIT

Samir Pokhrel

Kshitij Wagle

GOODWILL

(Publisher & Distributor)

CONTENT

Chapter 1
Introduction

Chapter 2
Digital Logic

Chapter 3
Combination Logic Circuits

Chapter 4
Data Processing Circuits

Chapter 5
Arithmetic Circuits

Chapter 6
Flip Flops

Chapter 7
Registers

Chapter 8
Counters

Chapter 9
Sequential Machine

Chapter 10
Digital Integrated Circuits

Chapter 11
Applications

Chapter 12
VHDL

Chapter 1

Introduction

A digital circuit is one that is built with devices with two well-defined states. Such circuits can process information represented in binary form. Systems based on digital circuits touch all aspects our present day lives. The present day home products including electronic games and appliances, communication and office automation products, computers with a wide range of capabilities, and industrial instrumentation and control systems, electromedical equipment, and defense and aerospace systems are heavily dependent on digital circuits. Many fields that emerged later to digital electronics have peaked and levelled off, but the application of digital concepts appears to be still growing exponentially. This unprecedented growth is powered by the semiconductor technology, which enables the introduction of more and complex integrated circuits. The complexity of an integrated circuit is measured in terms of the number of transistors that can be integrated into a single unit. The number of transistors in a single integrated circuit has been doubling every eighteen months (Moore' Law) for several decades and reached the figure of almost one billion transistors per chip. This allowed the circuit designers to provide more and more complex functions in a single unit.

The introduction of programmable integrated circuits in the form of microprocessors in 70s completely transformed every facet of electronics. While fixed function integrated circuits and microprocessors coexisted for considerable time, the need to make the equipment smaller and portable lead to replacement of fixed

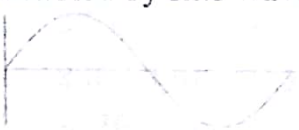
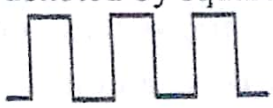
function devices with programmable devices. With the all pervasive presence of the microprocessor and the increasing usage of other programmable circuits like PLDs (Programmable Logic devices), FPGAs (Field Programmable Gate Arrays) and ASICs (Application Specific Integrated Circuits), the very nature of digital systems is continuously changing.

The central role of digital circuits in all our professional and personal lives makes it imperative that every electrical and electronics engineer acquire good knowledge of relevant basic concepts and ability to work with digital circuits.

1.1. Definitions of Digital Signal

Analog and digital signals are used to transmit information, usually through electric signals. In both these technologies, the information, such as any audio or video, is transformed into electric signals. The difference between analog and digital technologies is that in analog technology, information is translated into electric pulses of varying amplitude. In digital technology, translation of information is into binary format (zero or one) where each bit is representative of two distinct amplitudes.

Analog vs. Digital comparison chart

	Analog	Digital
Signal	Analog signal is a continuous signal with infinite range of values however the values are more exact.	Digital signals are discrete time signals with finite range of values generated by digital modulation technique.
Waves	Analog signals are denoted by sine wave. 	Digital signals are denoted by square wave. 
Example	Old radios, transfer of audio wave from microphone to speaker, megaphones, human	Digital voice telephone system, Computers, CDs, DVDs etc.

Analog vs. Digital comparison chart		
	Analog	**Digital**
	voice in air etc.	
Circuits	Made by fundamental electronic components like resistor, capacitor, inductor, diode, transistor and operational amplifier.	Made by combination of transistors and logic gates and at higher levels: microcontroller or other computing chips.
Response to Noise	In analog system any distortion or noise, no matter how small, will distort the received signal	Digital system can withstand channel noise and distortion much better than analog as long as the noise and the distortion are within limits.
Flexibility	Analog hardware is not flexible.	Digital hardware is flexible in implementation.
Uses	Can be used in analog devices only. Best suited for audio and video transmission.	Best suited for Computing and digital electronics.
Applications	• Thermometer, photocopiers, old land-line telephones, audio tapes, VCRs(same as TV) • Communication satellite • Telecommunication • Fiber optic communication • 3G communication	• PCs, PDAs, Mobile Phones • Audio recording: from tape to music CD to MP3(MPEG Layer 3) player • Image processing: from silver-halide film to digital camera • Telephone switching networks • Combustion control in car engines

Analog vs. Digital comparison chart		
	Analog	**Digital**
Bandwidth	Analog signal processing can be done in real time and consumes less bandwidth.	There is no guarantee that digital signal processing can be done in real time and consumes more bandwidth to carry out the same information.
Memory	Stored in the form of wave signal	Stored in the form of binary bit
Power	Analog instrument draws large power	Digital instrument draws only negligible power
Cost	Low cost and portable	Cost is high and not easily portable
Impedance	Low	High order of 100 MΩ
Errors	Analog instruments usually have a scale which is cramped at lower end and give considerable observational errors.	Digital instruments are free from observational errors like parallax and approximation errors.

1.2 Digital Waveforms

1.2.1 Digital States

In digital devices, there are only two states: on and off. Using only these two states, devices can communicate a great deal of data and control various other devices. In binary, these states are represented as a 1 or 0. Binary 1 is typically considered a logic high, and 0 is a logic low.

Voltage Levels

Digital devices, however, are often driven by analog devices with an infinite number of states. How do you turn an infinite number of

states into only two? The answer is by creating voltage logic levels, which define the voltage to represent a logic high or logic low.

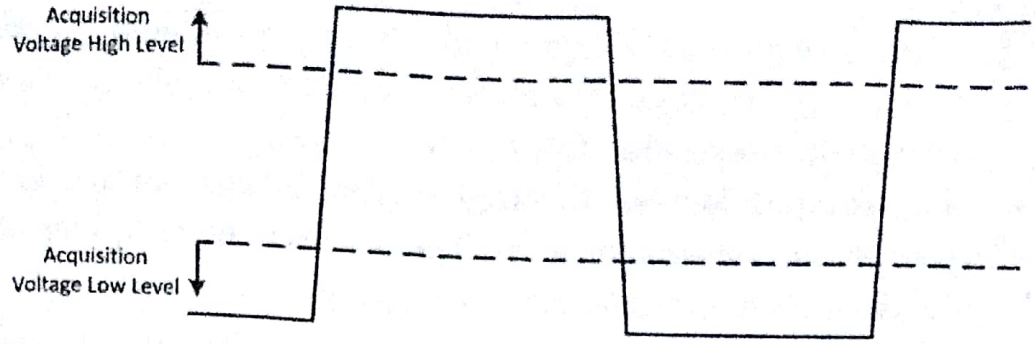


Fig: Voltage levels define the analog voltage that represents a logic high or low.

A system can define the voltage logic levels at any value it chooses, but many circuits represent a logic high by +5 V or +3.3 V to ground and a logic low as ground or 0 V. This type of system is called a positive or active-high. It describes how the pin is activated for an active-high pin, you connect it to your high voltage.

A negative or active-low system is the reverse. The higher voltage represents a logic low and the lower voltage represents a logic high. For an active-low pin, you must pull that pin low by connecting it to ground. Data sheets often denote a pin is active-low by putting a line above the pin name, such as $\overline{EN}$.

Although a high and low are specified, in most systems there is actually a range so as to be more practical. For example, a logic high might be any value between 2 V and 5 V and a low might be any value from 0 V to 1 V. Voltages outside those ranges are considered invalid and occur only in a fault condition or during a logic-level transition.

1.2.2 Logic Families

Standardized logic families make it easier to work with circuits and components. They provide a standardized voltage level that constitutes a logic high or logic low. All circuits within a logic family are compatible with other circuits within that same family because they share the same characteristics.

Single-Ended Logic Families

Single-ended logic families specify voltage levels in relation to ground. The four levels are defined as:

- **V_{OH} (output high-level voltage):** This is also known as the generation voltage high level. When configured for active drive generation, this is the voltage produced by the device when it generates a logic high. When configured for open collector generation, this is the equivalent to setting the data channel to a high-impedance state.
- **V_{OL} (output low-level voltage):** This is also known as the generation voltage low level. This is the voltage produced by the device when it generates a logic low.
- **V_{IH} (input high-level voltage):** This is also known as the acquisition voltage high level. This is the voltage level necessary to send to the device for it to read a logic high.
- **V_{IL} (input low-level voltage):** This is also known as the acquisition voltage low level. This is the voltage level necessary to send to the device for it to read a logic low.

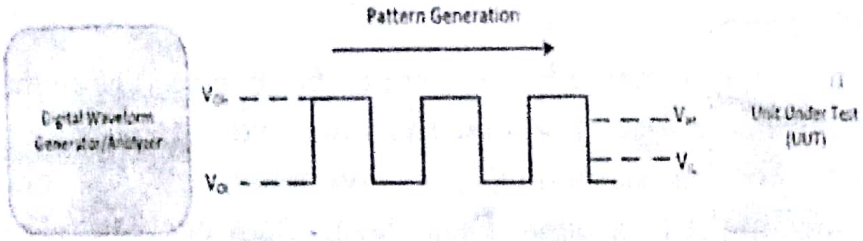


Fig: Single-ended logic levels are specified for output and input.

To accurately communicate with a device, be sure to configure the digital device such that the following conditions are met:

- $V_{OH} \geq$ DUT V_{IH}
- $V_{OL} \leq$ DUT V_{IL}
- $V_{IH} \leq$ DUT V_{OH}
- $V_{IL} \geq$ DUT V_{OL}
- $V_{IH} > V_{IL}$

There is usually a cushion between the output voltage of one device and the input of another. This is referred to as the noise margin or the noise immunity level (NIM). If you are in a noisy environment and having difficulty with incorrect data bits, consider increasing this value.

There are several single-ended logic families. Transistor-transistor logic (TTL) is very common for integrated circuits and is used in many applications such as computers, consumer electronics, and

test equipment. Circuits built from bipolar transistors achieve switching and maintain logic states. A TTL must also meet specific current specifications and rise/fall times.

Another common IC family is CMOS. These devices have high noise immunity, require less power consumption, and have a lower based voltage. Most of the voltage levels are similar to TTL devices for greater compatibility. This makes it easy to switch from a TTL to a CMOS device, but going the other direction can be trickier. Too high a voltage to a CMOS could damage the chip. In this case, you can use a voltage divider to reduce the voltage.

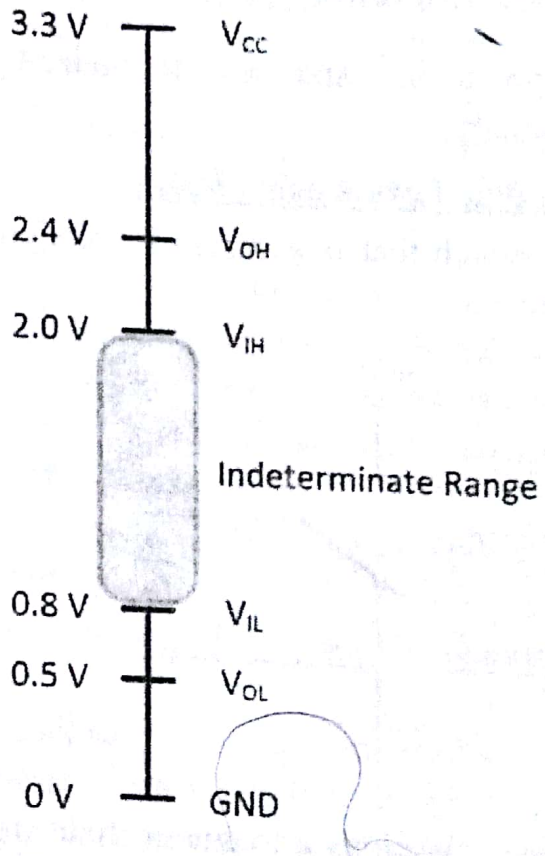


Fig: Standard CMOS Voltage Levels

1.3 Digital Logic

1.3.1 Generating Logic Levels

Generating Logic Levels using witch switch

The digital logic levels can be produced using switches as shown:

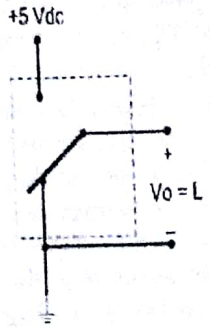


Fig: Switch is DOWN, V_o=L

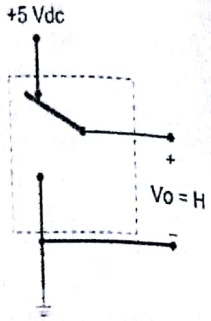
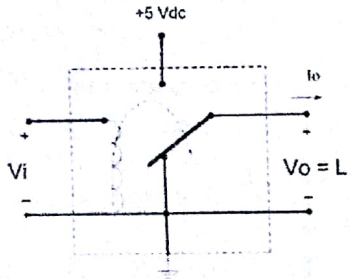


Fig: Switch is UP, V_o=H

Switch is easy to use and easy to understand, but it must be operated manually.

Generating Logic Levels using Relay

A relay is a switch that is actuated by applying a voltage V_i to a coil as shown:



The coil current develops a magnetic field that moves the switch arm from one contact to another. Switches and relays were useful in the construction of early machines used for calculation and/or logic operations.

Disadvantages:

- Bulkier
- Cannot switch rapidly

They are replaced by digital ICs.

1.3.2 The Buffer

A buffer is an electronic switch. It is actuated by the input voltage V_i. Its operation is similar to the relay. The buffer is capable of delivering additional current to a load, hence the name buffer amplifier.

The buffer operation

V_i	V_o
0	0
1	1

Fig: Truth table

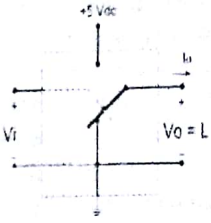


Fig: Model

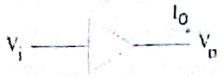


Fig: Symbol

The Tri-state Buffer

Consider a digital system with the possibilities: There may be more than one input signal and it is necessary to connect only one signal at a time. The output may need to be directed to more than one destination, one at a time. It is a simple buffer with an additional switch controlled by and input G.

When G is low (0) this switch is open and the output is disconnected from the buffer.

When G is high, the switch is closed and the output follows the input.

In effect, the control signal G connects the buffer to the load or disconnects the buffer from the load. Since it generates three types of signals it is called three-state buffer or tri-state buffer.

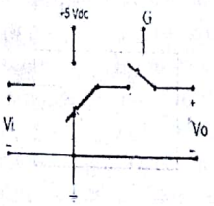


Fig: Model

Fig: Symbol

V_i	G	V_o
0	0	Open
1	0	Open
0	1	0
1	1	1

Fig: Truth Table

1.4 Moving and Storing Digital Information

1.4.1 Digital Signals

Digital systems are used to process discrete elements of information. A digital electronic system processes discrete electrical signals. They can be designed to perform simple logic and arithmetic operations and can therefore be used to construct the basic operational parts of a calculator. A digital electronic system can also be used to hold or store discrete elements of information and this gives the system a memory capability. The ability to store information or data and to process the data by logical or arithmetic operations is central to the design of nearly all digital information processing systems including digital computers.

The function of a digital system is determined by the sequence of operations which are performed on the information or data being processed. A digital system can be classified by the way in which its sequence of operations is implemented.

Digital electronic systems operate on physical quantities such as a level of a voltage or current. These quantities or signals are used to represent discrete elements of information. The signals within a digital electronic system are normally allowed to take only two discrete values because two-valued, or binary, signals can be represented unambiguously and reliably by electrical circuits or magnetic storage devices which have two clearly defined states. These states are often referred to as the binary coefficients 0 and 1 or the logic values 'false' and 'true'. It follows that all discrete element of information in the system must be represented by groups of binary digits.

The number of binary digits needed to encode the various discrete elements of information in a system has a significant effect on the design of a digital system. The system must be able to move, process and store discrete elements of information using either simulations (parallel) operations on a group of binary digits or sequential (serial) operations on a stream of binary digits. Since one binary digit or bit of information can be used to encode only two data objects, it is common to find digital systems operating on groups of 8, 16 or 32 bits of information at once. The range of different objects which can be encoded by a group of n bits is 2^n.

Systems which operate on larger groups of bits can, therefore, processes a wider range of encoded information.

1.4.2 Digital Representation of Information

Digital systems may be used to process textual, numerical or logical information. This information usually takes the form of a sequence or string of alphabetic characters, punctuation symbols, decimal digits and the symbols representing arithmetic operators, logical values and logical operators. It also includes the spaces which mark the boundaries between various words or quantities. This sort of information is represented in a digital system by codes of binary digits. Since the information must be input, processed, and output in coded form, the encoding must be reversible and assign a unique representation to each element of information. Each item of information or data whose value can change during processing is known as a variable and can be classified by the type of information which it represents.

1.4.3 Binary Storage, Registers, Switching Circuits & Logic Gates

Binary Cell

A binary cell is a device that can store one bit information and that bit is either a 0 or 1. These states are achieved by the virtue of receiving the excitation signals from the input. The output of binary cells is a physical quantity that can recognize or separate the two states.

Examples of binary cells are flip-flop circuits, ferrite cores etc.

Register

A register is group of binary cells that can store n-bits of data for n binary cells.

So we can say that the content of registers is a combination of 1's and 0's since a binary cell has either a 0 or 1. A digital computer is characterized by its registers used in memory unit, processor unit, control unit etc.

Register Transfer

An inter-register transfer operation transfers information from one register to another. To process the information, registers and digital logic circuits are required. The diagram below shows an

example of processing binary information where we want to add two bit binary numbers. In this particular example, the memory unit usually having thousands of registers contains only three memory registers to make the diagram and process more clear. In the processing unit, we have three processor registers named R1, R2, R3 and digital logic circuits to add the given data.

First data is input through keyboard to the memory registers. As memory register can only store and transfer the data so it sends the two operands to the processor register where the task is performed by the digital logic design circuit and the answer is stored in processor register R3. From R3 the answer is transferred to a new memory register again and the process gets completed. This process can be easily understood from the diagram given below:

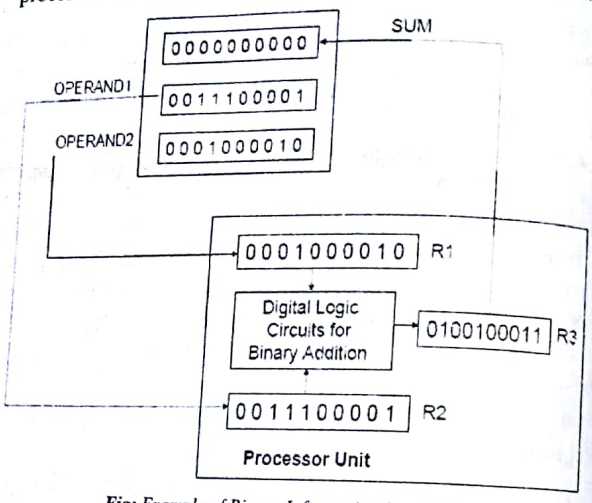


Fig: Example of Binary Information Processing

1.4.4 Magnetic and Optical Memory

There are three main categories of storage devices: optical, magnetic and semiconductor. The earliest of these was the magnetic device. Computer systems began with magnetic storage in the form of tapes (yes, just like a cassette or video tape). These graduated to the hard disk drive and then to a floppy disk. All magnetic media use the same general process of a read/write head magnetizing material. On a hard drive, the materials are magnetized on a glass or aluminum disk.

An optical storage device is written and read with a laser. It is strong and can handle temperature fluctuations much better than magnetic media. The disks used for storage (like CDs, DVDs and Blu-rays) were more expensive than floppies but held a lot of data. A compact disc (CD) can hold 700 MB of data, or roughly a little over an hour of music. Digital video discs (DVDs) began being issued for consumer-released movies. A single-sided DVD holds 4.7 gigabytes (GB) of data, so a normal, not overly-computerized two-hour movie will fit.

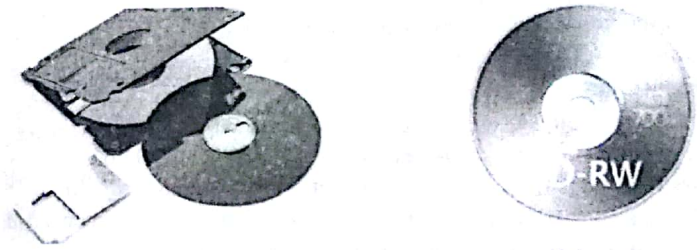


Fig: (a) Internal of floppy disk (b) Optical disk

1.5 Digital Operations

Counting is frequently required in digital computers and other digital systems to record the number of events occurring in a specified interval of time. Normally an electronic counter is used for counting the number of pulses coming at the input line in a specified time period. The counter must possess memory since it has to remember its past states. As with other sequential logic circuits counters can by synchronous or asynchronous.

An **arithmetic logic unit (ALU)** is a digital circuit used to perform arithmetic and logic operations. It represents the fundamental building block of the **central processing unit (CPU)** of a computer. Modern CPUs contain very powerful and complex ALUs. In addition to ALUs, modern CPUs contain a control unit (CU). An ALU performs basic arithmetic and logic operations.

Examples of arithmetic operations are addition, subtraction, multiplication, and division. Examples of logic operations are comparisons of values such as NOT, AND, and OR.

In the digital system there is the provision of entering the data in the system and extraction of data from the system. In case of a computer, information is frequently entered by typing on a keyboard or perhaps by using a magnetic floppy disk. There is a requirement to connect multiple input devices, one at a time, to the system. The digital circuit used for this operation is a multiplexer. Likewise, there is a need to connect the system output to a number of different destinations, one at a time. The digital circuit used for this purpose is a demultiplexer. The multiplexer have n input lines. Each line is used to shift digital data serially. There is a single output line which is connected to the computer input port. The principle of demultiplexer is just opposite of the multiplexer (one input and many output).

Also, encoding is the process of putting a sequence of characters (letters, numbers, punctuation, and certain symbols) into a specialized format for efficient transmission or storage. Decoding is the opposite process. I.e. the conversion of an encoded format back into the original sequence of characters.

1.6 Digital Computer

Digital computer is one kind of device which is capable of solving problems by processing information in the discrete form. The computer operated on discrete or digital data is known as digital computer. The digital computer is also operated on magnitudes, letters and symbols by representing it in binary form or by converting in digital form. It can also process the business data which is in discrete form. The digital computer is generally used in education and scientific computation.

Digital computer has five basic functional elements which are shown in the block diagram of digital computer below.

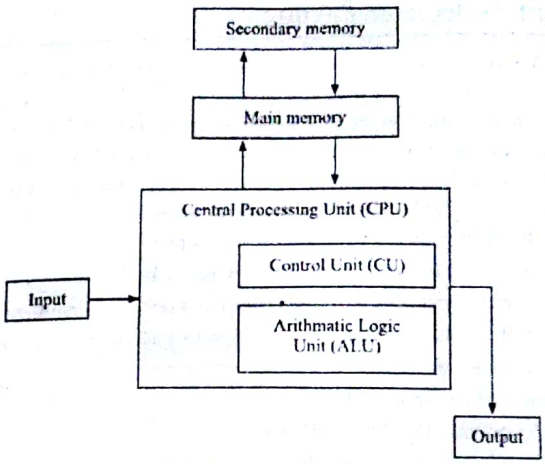


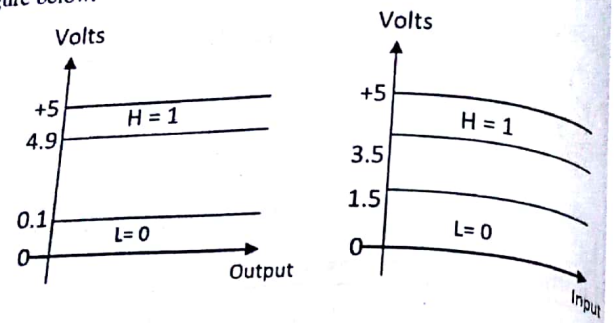
Fig: Block diagram of digital computer

1. Input
2. Output
3. Main memory
4. Control unit
5. Arithmatic logic unit (ALU)

1. **Input:** This device is used to enter data, programme, and instruction into computer. The examples of input devices are keyboard, mouse, scanner, etc.
2. **Output**: Devices produce the output after processing operation by the computer. The examples of output devices are printer, CRT monitor, LCD monitor, etc.
3. **Main memory:** The device used to store the input data as well the result produced by computer. The example of main memory is RAM.
4. **Control unit:** It selects and calls up instructions from the memory in appropriate sequence and relays the proper commands to the appropriate unit for processing.
5. **Arithmatic logic unit (ALU):** This unit is used to perform arithmatic as well as logical operation.

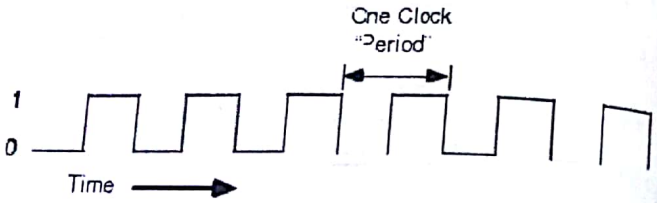
CMOS Logic Levels

The input and output profile of CMOS logic level is shown in figure below.



1.9 Clock Waveform

A clock signal is produced by a clock generator. A symmetrical signal which is frequently used as a basis for timing all operations in digital system are called clock signal. Which is shown in figure below:



Although more complex arrangements are used, the most common clock signal is in the form of a square wave with a 50% duty cycle, usually with a fixed, constant frequency. Circuits using the clock signal for synchronization may become active at either the rising edge, falling edge, or, in the case of double data rate, both in the rising and in the falling edges of the clock cycle.

Clock signals are used to synchronize digital transmitters and receivers during data transfer. For example, the transmitter can use each rising edge of the clock signal to send each bit of data and the receiver can use the same clock to read the data.

1.10 Coding

Computers and other digital circuits process data in binary format. Various binary codes are used to represent data which may be numeric, alphabetic or special characters. Codes are also used for error detection and error correction in digital systems. When numbers, letters or words are represented by a special group of symbols, this is called encoding or coding. Codes are broadly classified as *weighted*, *non-weighted* and *alphanumeric* codes.

Weighted code

If each position of a number represents a specific weight then the coding scheme is called weighted binary code. In such coding the bits are multiplied by their corresponding individual weight, and then the sum of these weighted bits gives the equivalent decimal digit.

Eg.: BCD code ⇒8421 BCD code, 4221 BCD code, 5421 BCD code

Since the 8421 code is the most popular of all the BCD codes, it is simply referred to as the BCD code.

Non-weighted code

These codes are not positionally weighted. It basically means that each position of the binary number is not assigned a fixed value.

Eg.: Excess-3 code, Gray code

Alphanumeric code

An alphanumeric code is a binary code of a group of elements consisting of ten decimal digits, the 26 letters of the alphabet (both in uppercase and lowercase), and a certain number of special symbols such as #, /, &, %, etc. The total number of elements in an alphanumeric code is greater than 36. Therefore it must be coded with a minimum number of 6 bits ($2^6 = 64$, but $2^5 = 32$ is insufficient).

Eg.: ASCII, EBCDIC

1.10.1 ASCII Code

The full form of ASCII (pronounced "as-kee") is "American Standard Code for Information Interchange," used in most microcomputers. It is actually a 7-bit code, where a character is

represented with seven bits. The character is stored as one byte with one bit remaining unused. But often the extra bit is used to extend the ASCII to represent an additional 128 characters.

E.g.: A → $(41)_{16}$ is represented by 1 0 0 0 0 0 1

1.10.2 EBCDIC Code

EBCDIC (Extended Binary Coded Decimal Interchange Code) (pronounced either "ehb-suh-dik" or "ehb-kuh-dik") is a binary code for alphabetic and numeric characters that IBM developed for its larger operating systems. It is the code for text files that is used in IBM's OS/390 operating system for its S/390 servers and that thousands of corporations use for their legacy applications and databases. In an EBCDIC file, each alphabetic or numeric character is represented with an 8-bit binary number (a string of eight 0's or 1's). 256 possible characters (letters of the alphabet, numerals, and special characters) are defined.

IBM's PC and workstation operating systems do not use IBM's proprietary EBCDIC. Instead, they use the industry standard code for text, ASCII. Conversion programs allow different operating systems to change a file from one code to another.

1.10.3 BCD Code:

The full form of BCD is 'Binary-Coded Decimal.' i.e. it is the simplest binary code to represent a decimal number.

The code is also known as ***8-4-2-1 code***. This is because 8, 4, 2, and 1 are the weights of the four bits of the BCD code. So, 16-numbers (2^4) can be represented by 4-bits and *only ten of these are valid*. Remaining binary codes 1010, 1011, 1100, 1101, 1110, 1111, representing 10, 11, 12, 13, 14, and 15 in decimal are never being used in BCD code. So these six codes are called *forbidden codes* and the group of these codes is called the *forbidden group* in BCD code. BCD code for decimal digits 0 to 9 is shown in table.

Decimal	BCD(8421)
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1

Merits and demerits of BCD in digital system.

The users give the data in digital form. The input decimal numbers are stored internally by the means of decimal code. For integers from 0 to 9, the BCD has chosen to be both a code and a direct conversion.

For integers greater than 9, BCD occupies more space than the corresponding binary. E.g.

Decimal → 13
Binary → 1101
BCD code → 0001 0011

Example

Give the BCD equivalent for the decimal number 589.

→ The decimal number is 589
BCD code is 0101 1000 1001
Hence, $(589)_{10} = (010110001001)_{BCD}$

BCD addition

There are certain rules to be followed in BCD addition as given below.

(i) First add the two numbers using normal rules for binary addition.

(ii) If the 4-bit sum is equal to or less than 9, it becomes a valid BCD number.

(iii) If the 4-bit sum is greater than 9, or if a carry-out of the group is generated, it is an invalid result. In such a case, add $(0110)_2$ or $(6)_{10}$ to the 4-bit sum in order to skip the six invalid states and return the code to BCD. If a carry results when 6 is added, add the carry to the next 4-bit group.

Example:

Add the following BCD numbers:

(i) 0 1 1 1 and 1 0 0 1

(ii) 1 0 0 1 0 0 1 0 and 0 1 0 1 1 0 0 0.

Solution:

(*i*)

```
   0 1 1 1
 + 1 0 0 1
 ---------
   1 0 0 0 0      → Invalid BCD number
 + 0 1 1 0        → Add 6
 ---------
 0 0 0 1 0 1 1 0  → Valid BCD number
   1     6
```

(*ii*)

```
     1 0 0 1  0 0 1 0
   + 0 1 0 1  1 0 0 0
   ---------------------
     1 1 1 0  1 0 1 0   → both groups are invalid
   + 0 1 1 0  0 1 1 0   → Add 6 on each group
   ---------------------
   0 0 0 1  0 1 0 1  0 0 0 0   → Valid BCD number
     1        5        0
```

1.10.4 The Excess-3 Code

Excess-3, also called XS3, is a non-weighted code used to express decimal numbers. It is another important binary code. It is particularly significant for arithmetic operations as it overcomes the shortcomings encountered while using the 8421 BCD code to add two decimal digits whose sum exceeds 9. This code is used in some old computers and it was used in hand-held portable electronic calculators of the 1970s. This code assignment is obtained from the corresponding value of 4-bit binary code after adding 3 (0 0 1 1) to the given decimal digit. Here the maximum value may be 1 1 0 0. Since the maximum decimal digit is 9 we have to add 3 to 9 and then get the BCD equivalent. Like 8 4 2 1 and 2 4 2 1 codes, Excess-3 is also a ***self-complementary code***. BCD code for decimal digits 0 to 9 is shown in table.

Decimal	BCD(8421)	Excess-3
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100

Here 10 of 16 possible combination are used in excess-3 code. Thus, 6 invalid combinations are: 0000, 0001, 0010, 1101, 1110 and 1111.

Examples:

i) Convert $(367)_{10}$ into its Excess-3 code.

Solution:

The decimal number is	3	6	7
Add 3 to each bit	+3	+3	+3
Sum	6	9	10

Converting the above sum into 4-bit binary equivalent, we have a 4-bit binary equivalent of 0110 1001 1010.

Hence, the Excess-3 code for $(367)_{10}$ = 0110 1001 1010

ii) Convert (58.43)10 into its Excess-3 code.

Solution:

The decimal number is	5	8	4	3
Add 3 to each bit	+3	+3	+3	+3
Sum	8	11	7	6

Converting the above sum into 4-bit binary equivalent, we have a 4-bit binary equivalent of 1000 1011 0111 0110.

Hence, the Excess-3 code for $(367)_{10}$ = 10001011.0111011

1.10.5 Gray Code

This is a variable weighted (i.e. unweighted) code and is cyclic. This means that it is arranged so that every transition from one

value to the next value involves only one bit change. The gray code is sometimes referred to as reflected binary, because the first eight values compare with those of the last 8 values, but in reverse order.

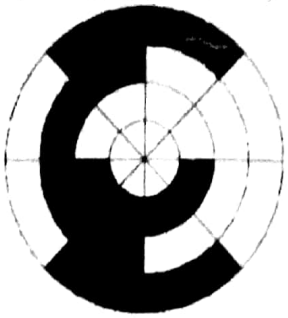


Fig: gray code in rotary encoder

Application of Gray code

A common application is for finding position in rotary encoders. With normal binary code there will be some glitches at the edge of the region because there may be multiple transitions in the code at once but in the rotary disk those bits may not be read at the same time, therefore the reading will be incorrect.

It has application also in input/output devices, some analog to digital converters and designation of rows and columns in Karnaugh map.

Decimal	BCD	Gray
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101

10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

Binary to Gray conversion

(i) The most significant digit (leftmost) in the gray code is same as the corresponding digit in binary code.

(ii) Going from left to right, add each adjacent pair of binary digits to get next gray code digit and neglect carry.

Example

Convert (1 0 1 1 0)$_2$ into its Gray code.

Solution:

$$\Rightarrow 1\ 0\ 1\ 1\ 0$$
$$= 1 \quad 1+0 \quad 0+1 \quad 1+1 \quad 1+0$$
$$= 1 \quad 1 \quad 1 \quad 0 \quad 1$$
$$\therefore (10110)_2 = (11101)_{gray}$$

Gray to Binary conversion

(i) The most significant digit (leftmost) in the binary code is same as the corresponding digit in binary code.

(ii) Add each binary digit generated to Gray digit in next adjacent position and neglect carry.

Example

Convert (1 0 1 1)$_{gray}$ into its Binary code.

Solution:

$$\Rightarrow \quad 1 \quad 0 \quad 1 \quad 1$$
$$\downarrow \quad \nearrow \oplus \quad \downarrow \quad \nearrow \oplus \quad \downarrow \quad \nearrow \oplus \quad \downarrow$$
$$1 \quad 1 \quad 0 \quad 1$$
$$= \quad 1 \quad 1 \quad 0 \quad 1$$
$$\therefore \quad (1011)_{gray} = (1101)_2$$

Important Questions

1) Define digital operations. What is Excess-3 Code explain with example. [IOE 2071 Shrawan]
2) What are major difference between Binary code and BCD code? [IOE 2072 Kartik]
3) Explain different coding system used to represent data. [IOE 2070 Ashad]
4) Describe in your words the characteristics of an analog and a digital signal. [IOE 2069 Ashad]
5) Define digital IC signal levels. What is Gray Code? Explain with example. [IOE 2069 Chaitra]
6) What are BCD numbers? State their merits and demerits when used in digital system. [IOE 2066 Bhadra]
7) What is weighted code and non-weighted code? What will be the BCD, Excess-3 and Gray code for the decimal number 15? [IOE 2071 Chaitra]
8) Compare analog and digital system. Which one is better and why? [PU 2011 Fall]
9) What do you mean by alphanumerical code? "Excess- 3 codes are called self-complementing code". Explain it. [PU 2011 Fall]
10) Perform the following conversion.
    i. $(5849)10 = (\quad)_{Excess\text{-}3}$
    ii. $(8412)10 = (\quad)_{2421}$
    iii. $(10101111)_{GRAY} = (\quad)_2$
    [PU 2014 Fall]
11) What are the differences between digital and analog system? Why digital systems are preferred rather than analog system? [PU 2014 Fall]

●●●

Chapter 2

Digital Logic

A Digital Logic Gate is an electronic device that makes logical decisions based on the different combinations of digital signals present on its inputs. Logic gates are the basic building blocks of any digital system. It is an electronic circuit having one or more than one input and only one output. The relationship between the input and the output is based on a certain logic. There are three basic logic gates, namely the OR gate, the AND gate and the NOT gate. Other logic gates that are derived from these basic gates are the NAND gate, the NOR gate, the Exclusive - OR gate and the Exclusive - NOR gate. In digital logic design only two voltage levels or states are allowed and these states are generally referred to as Logic "1" and Logic "0", High and Low, or True and False. These two states are represented in Boolean algebra and standard truth tables by the binary digits of "1" and "0" respectively.

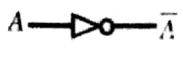
2.1 The Basic Gates

There are mainly three basic gates. They are NOT, OR and AND gate. We can derive various other gates using basic gates.

a. NOT gate:

It is the gate which gives high output when we provide low input and vice versa. It takes single inputs and gives single output.

Symbol:



Truth table:

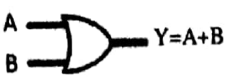
A	Y
1	0
0	1

Boolean expression: $Y = \overline{A}$

b. OR gate:

It is the gate having two or more input and only one output. It gives high output if any of input is high.

Symbol:



Truth table:

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

Boolean expression
Y=A+B

c. AND gates:

It gives high output only when all input are high.

Symbol:

Truth table:

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

Boolean expression:
Y=AB

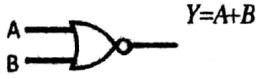
2.2 Universal gates

AND, OR and NOT gates can be used to perform various different logic functions. When we use only one type of gate for various logic function it will become quite easier. So, gates which can perform those tasks are called universal gates. There are two universal gates. They are:

a) NOR gates:

It is the circuit which we can obtain by inverting the output of OR gates.

Symbol:



Truth table:

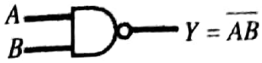
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

Boolean expression:
$Y=\overline{A+B}$

b) NAND Gate:

We can obtain NAND gates by inverting the output of AND gates.

Symbol:



Truth table:

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

Boolean expression:
$Y=\overline{AB}$

De-Morgan's Theorem

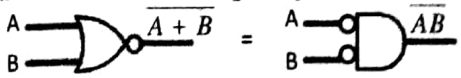
Theorem 1:

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

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

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

In term of circuit a NOR gate equals a bubbled AND gate



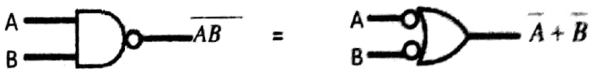
Theorem 2:

The complement of product is equal to the sum of individual complement.

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

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

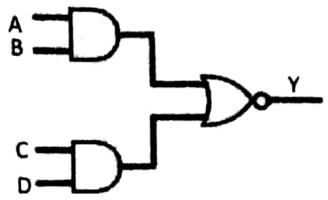
In term of circuit a NAND gate is equivalent to a bubbled OR



2.3 AND-OR-INVERT Gates

When all those three gates i.e. AND, OR and NOT are used to build any designed logic circuit then it is called as AND-OR-INVERT gates.

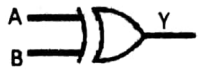
Eg:- $Y=\overline{(AB+CD)}$



Exclusive OR gate or XOR gate:

It gives low output when both of input is either high or low.

Symbol:



Truth table:

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

Boolean expression:

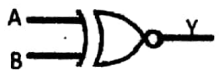
$Y = A \oplus B$

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

Exclusive NOR gate or XNOR gate:

It gives high output when both of input is either high or low.

Symbol:



Truth table:

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

Boolean expression:

$Y = A \odot B$

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

IEEE/ANSI Standard Symbols for basic gates:

The symbols used thus far in the chapter for representing different types of gate are the ones that are better known to all of us and have been in use for many years. Each logic gate has a symbol with a distinct shape. However, for more complex logic devices, e.g. sequential logic devices like flip-flops, counters, registers or arithmetic circuits, such as adders, subtractors, etc., these symbols do not carry any useful information. A new set of standard symbols was introduced in 1984 under IEEE/ANSI Standard 91–1984. The logic symbols given under this standard are being increasingly used now and have even started appearing in the literature published by manufacturers of digital integrated circuits. The utility of this new standard will be more evident in the following

paragraphs as we go through its salient features and illustrate them with practical examples.

IEEE/ANSI Standards – Salient Features

This standard uses a rectangular symbol for all devices instead of a different symbol shape for each device. For instance,

a) All logic gates (OR, AND, NAND, NOR) will be represented by a rectangular block.

b) A right triangle is used instead of a bubble to indicate inversion of a logic level. Also, the right triangle is used to indicate whether a given input or output is active LOW. The absence of a triangle indicates an active HIGH input or output. As far as logic gates are concerned, a special notation inside the rectangular block describes the logic relationship between output and inputs.

c) A '1' inside the block indicates that the device has only one input. An AND operation is expressed by '&', and an OR operation is expressed by the symbol '≥1'. Figure below shows the ANSI counterparts of various logic gates.

d) A '≥1' symbol indicates that the output is HIGH when one or more than one input is HIGH.

e) An '&' symbol indicates that the output is HIGH only when all the inputs are HIGH. The two-input

f) EX-OR is represented by the symbol '=1' which implies that the output is HIGH only when one of its inputs is HIGH.

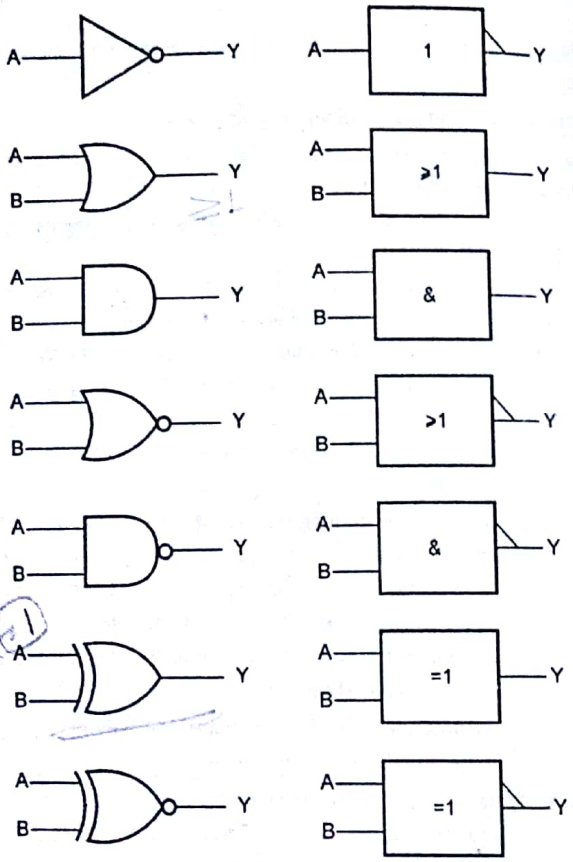


Fig: IEEE / ANSI symbols.

Application of logic gates:

Logic gates are the building block of digital electronic. Basically they are found in various digital devices like Arithmetic and Logic Unit, memory, multiplexer, registers etc. Some common application of logic gates are tabulated below:

Gates	Application
NOT gate	➢ Suppose its use in cooling system. When the temperature exceeds 25 degree C then turn on the cooler if less than 25 then turn off. ➢ Basically used to build square-wave oscillators for generating clock signals. These clock generators offer good stability, operation over a wide supply voltage range (3–15 V) and frequency range (1 Hz to in excess of 15 MHz), low power consumption and an easy interface to other Logic families.
OR gate	➢ It is used in those situations where the occurrence of any one or more than one event needs to be detected. Suppose a doorbell to ring when someone presses either the front door switch or the back door switch.
AND gate	➢ Suppose its application in fire alarm system. When both heat detector and smoke detector sensor passes high output then siren will be ON otherwise it will be off. ➢ It is commonly used as an ENABLE or INHIBIT gate to allow or disallow passage of data from one point in the circuit to another.
Ex-OR / Ex-NOR	➢ They are commonly used in parity generation and checking circuits

Realization of various gates from NAND and NOR gates.

1) From NOR gate

For realization of various gate from NOR we have to remember De-Morgan's 2nd law, Boolean laws & theorem.

a) NOT gate from NOR gate

For NOT gate

$Y = \overline{A}$

In symbol

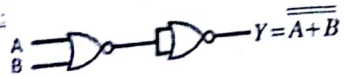
b) OR gate from NOR gate:

For OR gate:

$$Y = A + B$$

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

In symbol:



c) AND gate from NOR gate:

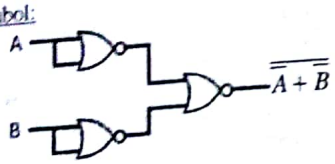
For AND gate:

$$Y = AB$$

$$Y = \overline{\overline{AB}}$$

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

In symbol:



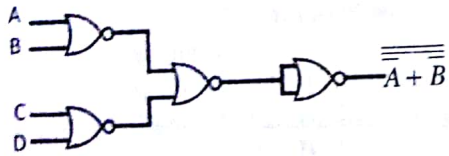
d) NAND gate from NOR gate:

$$Y = \overline{AB}$$

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

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

In symbol:



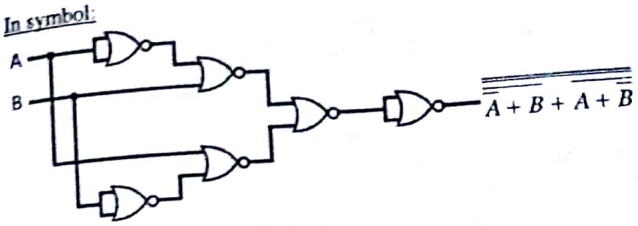
e) XOR gate from NOR gate:

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

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

$$= \overline{\overline{A} + B} + \overline{A + \overline{B}} \quad [De - Morgan's\ 2^{nd}\ law]$$

In symbol:



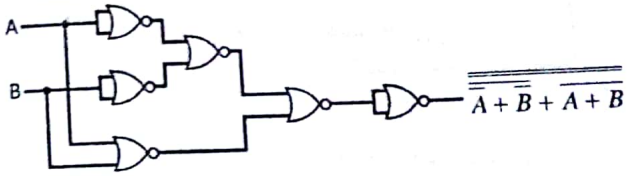
f) XNOR gate from NOR gate:

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

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

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

In symbol:



2. From NAND gate:

a. NOT gate from NAND gate:

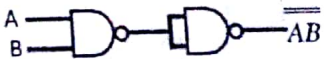
$$Y = \overline{A}$$

In symbol:

b. AND gate from NAND gate:

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

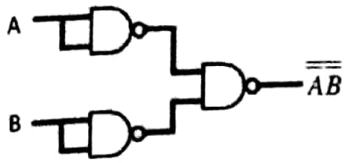
In symbol:



c. OR gate from NAND gate:

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

In symbol:

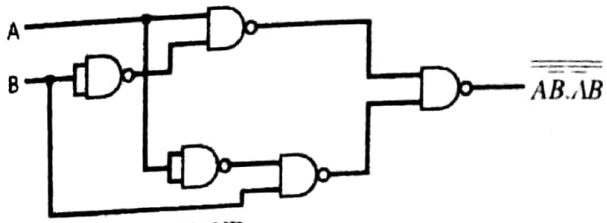


d. XOR from NAND:

$Y = A \oplus B$

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

In symbol:



e. XNOR from NAND:

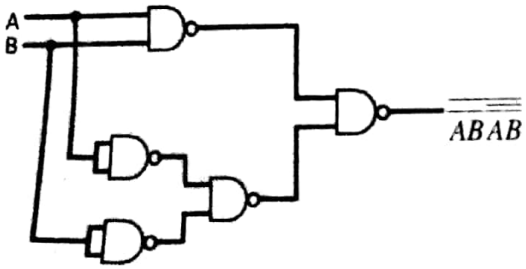
$Y = A \odot B$

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

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

$= \overline{\overline{AB}.\overline{AB}}$

In symbol:



Q) Define universal gates with example. Realize EX-OR gate using NAND gate only. *[IOE 2071 Shrawan]*

2.4 Positive and negative logic

a. **Positive logic:**
When we use binary 1 for high voltage i.e. +5 V and binary 0 for low voltage i.e. 0 V, then it is called positive logic.

b. **Negative logic:**
When we use binary 0 for high voltage i.e. +5 V and binary 1 for low voltage i.e. 0 V, then it is called negative logic.

Remember that positive true represents a high voltage & negative true always represents a low voltage.

1) **Consider a gate which outputs high voltage only when all inputs are high and outputs low voltage otherwise. How this gate does behaves in positive and negative logic system?**

Consider the two inputs A and B and one output Y. 'H' be high and 'L' be low. According to the question we can draw the truth table as below.

Truth Table (a):

A	B	Y
L	L	L
L	H	L
H	L	L
H	H	H

Positive logic system:

In positive logic system high is represented by 1 and low is represented by 0. Now implementing this system in truth table (a) we can get truth table as below.

Truth table (b)

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

This is the truth table of positive logic AND gate

Negative logic system:

In negative logic system high is represented by 0 and low is represented by 1. Now implementing this system in truth table (a) we can get truth table as below:

Truth table (c)

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

This is the truth table of negative logic AND gate.

2) **Consider a gate which outputs high voltage when any of the inputs are high and outputs low voltage otherwise. How this gate does behaves in positive and negative logic system.**

Consider the two inputs A and B and one output Y. 'H' be high and 'L' be low. According to the question we can draw the truth table as below.

Truth Table (a):

A	B	Y
L	L	L
L	H	H
H	L	H
H	H	H

Positive logic system:

In positive logic system high is represented by 1 and low is represented by 0. Now implementing this system in truth table (a) we can get truth table as below.

Truth table (b)

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

This is the truth table of positive logic OR gate

Negative logic system:

In negative logic system high is represented by 0 and low is represented by 1. Now implementing this system in truth table (a) we can get truth table as below.

Truth table (c)

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

This is the truth table of negative logic OR gate.

From the above questions 1 and 2 we can say following things:

a) **Positive logic AND gate is equivalent to negative logic OR gate.**

Positive logic AND

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

Negative logic OR gate

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

b) **Positive logic OR gate is equivalent to negative logic AND gate.**

Positive logic OR gate

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

Negative logic AND gate

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

Equivalence of positive and negative logic:

As above equivalence we can derive various other which are as below.

- Positive logic **NAND gate** is equivalent to Negative logic **NOR gate**.

Positive logic NAND gate

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

Negative logic NOR gate

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

- Positive logic **NOR gate** is equivalent to negative logic **NAND gate**.
- Positive logic **XOR gate** is equivalent to negative logic **XNOR gate**.
- Positive logic **XNOR gate** is equivalent to negative logic **XOR gate**.
- Positive logic **XOR gate** is equivalent to negative logic **XNOR gate**.

Solved Problems

1) **Prove that positive XOR gate is equivalent to negative XNOR gate.** [IOE 2070 Chaitra]

Here first of all we have to show the XOR gate in the positive and negative logic system and similarly we have to show the XNOR gate in positive and negative logic system and in last we have to prove positive XOR gate is equivalent to negative XNOR gate as below.

Positive XOR gate

Positive XOR gate

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

Negative XNOR gate

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

From the above truth table we can say that positive XOR gate is equivalent to negative XNOR gate

2) **When FF_H is ANDed with CO_H what is the resulting number?** [IOE 2070]

Given:

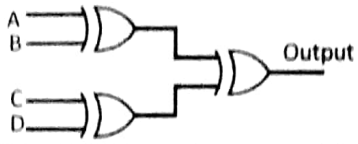
$FF_H = (1111\ 1111)_2$ $CO_H = (1100\ 0000)_2$

Truth table of AND gate:

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

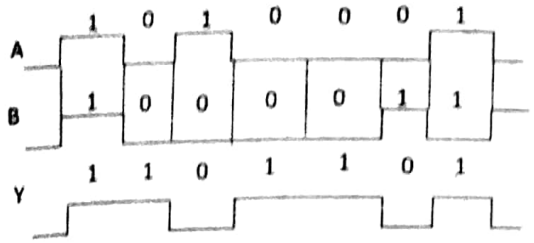
Hence, when FF_H ANDed with CO_H we get 1100 0000.

3) **Construct truth table of 4 output XOR gate using two input three XOR gate**



Input				Output		
A	B	C	D	A ⊕B	C ⊕D	(A ⊕B) ⊕(C ⊕D)
0	0	0	0	0	0	0
0	0	0	1	0	1	1
0	0	1	0	0	1	1
0	0	1	1	0	0	0
0	1	0	0	1	0	1
0	1	0	1	1	1	0
0	1	1	0	1	1	0
0	1	1	1	1	0	1
1	0	0	0	1	0	1
1	0	0	1	1	1	0
1	0	1	0	1	1	0
1	0	1	1	1	0	1
1	1	0	0	0	0	0
1	1	0	1	0	1	1
1	1	1	0	0	1	1
1	1	1	1	0	0	0

4) If the following two input waveforms are applied to an XNOR gate what is the resulting output waveform.



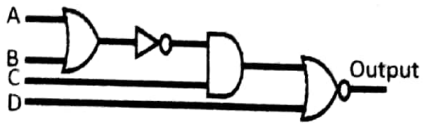
The truth table of XNOR gate is:

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

Important Questions

1) Using De Morgan's theorem express the function F = ABC+A'C'+A'B' with only OR and complement operation. [IOE 2067]
2) State De Morgan's theorem? Why NAND and NOR gates are called universal logic gates? [IOE 2067]
3) List out the name of universal gates and way they are called universal gates? Realize EX-OR gate using only NAND gates? [IOE 2068]
4) Define positive and negative logic with example. [IOE 2069]
5) Construct the basic gates using only universal gates. [IOE 2069]
6) What is the main significance of IEEE/ANSI symbols when compared with the conventional ones? Draw the ANSI symbols for four-input OR, two-input AND, two-input EX-OR and two-input NAND gates.

7) Explain the operation of NAND, NOR, XOR and NOT gates with Boolean expression and truth table. [IOE 2070 Ashad]
8) How do you distinguish between positive and negative logic systems? Prove that an OR gate in a positive logic system is an AND gate in a negative logic system.
9) What is the only input combination that:
   (a) Will produce a logic '1' at the output of an eight-input AND gate?
   (b) Will produce a logic '0' at the output of a four-input NAND gate?
   (c) Will produce a logic '1' at the output of an eight-input NOR gate?
   (d) Will produce a logic '0' at the output of a four-input OR gate?
10) Draw the truth table of the logic circuit shown in Fig.



11) Define universal gate. Design the three bit EX-OR circuit using only Universal gates. [PU Spring 2014]
12) Explain the universal property of NOR gate with appropriate logic gates. [PU Fall 2013]
13) Describe NAND gate as universal gate. [Purbanchal 2008]
14) State and prove De Morgan's 1st and 2nd law with logic gates and truth table. [Purbanchal 2008]

Chapter 3

Combinational Logic circuits

A combinational logic circuit consists of logic gates whose output at any time are determined directly from the present combination of input without regard to previous inputs. A combinational circuit perform a specific information - processing operation fully specified logically by the set of Boolean function. A combinational circuits consists of input variables, logic gates, and output variables. The logic gates accepts signals from the inputs and generates signals to the outputs. This process transforms binary information from the given input data to the required output data.

3.1 Boolean laws and theorem

There are various laws & theorem which helps us to rearrange Boolean equations in simpler logic circuits.

Basic law

Commutative law:
- A+B=B+A
- AB=BA

Associative law:
- A+(B+C)=(A+B)+C
- A(BC)=(AB)C

Distributive law
- A(B+C)=AB+BC

OR operation
- A+0=A
- A+A=A
- A+1=1
- $A+\overline{A}=1$

AND operation
- A.1=A
- A.A=A
- A.0=0
- $A.\overline{A}=1$

Double inversion
- $\overline{\overline{A}}=A$

Dual theorem:

Starting with a Boolean relation, we can derive another Boolean relation, called its dual by the following steps:

a) Changing each OR sign to an AND sign.
b) Changing each AND sign to an OR sign.
c) Complementing all 0's and 1's.

e.g. Dual of A.0=0 is written as A+1=1.

We can assume variable 'A' and 'B' either 0 or 1 and can understand the rules are true or not as well as ideas to remember easily.

Boolean algebraic theorem/laws/expression.

1)	A + 0 = A	2)	A.1 = A
3)	A + 1 = 1	4)	A. 0 = 0
5)	A + A = A	6)	A. A = A
7)	$A + \overline{A} = 1$	8)	$A. \overline{A} = 0$
9)	A. (B + C) = A. B + A. C	10)	A + (B. C) = (A + B). (A + C)
11)	A + (A. B) = A	12)	A. (A + B) = A
13)	$A + \overline{A}B = A + B$	14)	$A. (\overline{A} + B) = A. B$
15)	$A. B + A. \overline{B} = A$	16)	$(A + B). (A + \overline{B}) = A$
17)	$A. B + \overline{A} . C = (A + C). (\overline{A} + B)$	18)	$(A + B). (\overline{A} + C) = (A. C) + (\overline{A} . B)$
19)	$(A. B) + (\overline{A} . C) + (B. C) = (A. B) + (\overline{A} . C)$	20)	$(A + B). (\overline{A} + C). (B + C) = (A + B). (\overline{A} + C)$

Proof of expression 13.

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

Proof of expression 17.

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

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

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

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

$= C(B + \overline{A}\overline{B}) + AB\overline{C}$ $[A+\overline{A}B=A+B]$

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

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

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

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

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

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

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

Proof of expression 22.

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

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

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

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

$= \overline{A}B + BC + AC$

$= \overline{A}B + BC + AC + A\overline{A}$

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

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

Q) Prove that:

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

L.H.S

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

$= AC(\overline{A}B + C)(\overline{A}BC + \overline{C})$ [Distributive law]

$= (AC\overline{A}B + ACC)(\overline{A}BC + \overline{C})$ $[X\overline{X} = 0]$

$= AC(\overline{A}BC + \overline{C})$

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

$= 0$

Q) Prove that:

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

L.H.S

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

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

$= A(B + C) + \overline{A}BC$ $[A + \overline{A}B = A + B]$

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

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

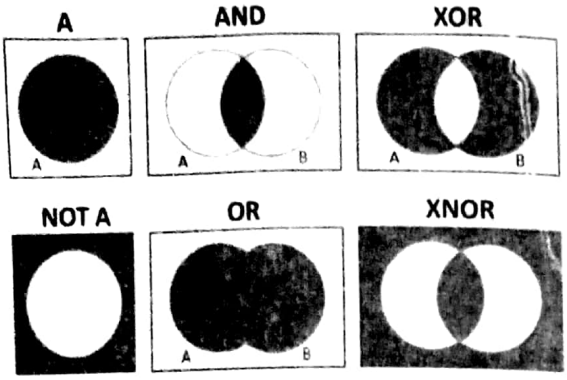
$= AB + C(A + B)$

$= AB + AC + BC$

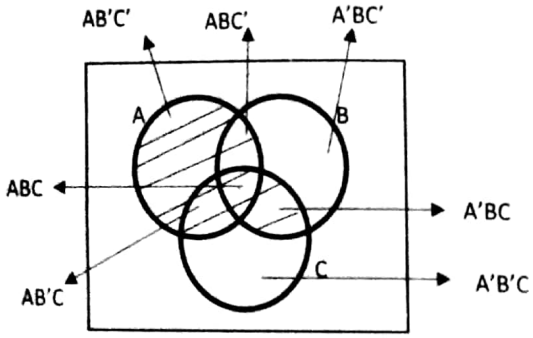
3.2 The Venn diagram

The perfect induction can be used to verify the theorems and properties. This procedure is quite tedious and not very informative from the conceptual point of view. A simple visual aid that can be used for this purpose also exists. It is called the Venn diagram.

- In Venn diagram the entire space of logical possibilities – every row of the truth table – is represented within that rectangle.
- The logical values of rectangle (which represent all possibilities) is 1 or "true".
- The two circle represent A and B. Everything inside the 'A' circle represents the entries in the truth table in which A is true.
- Everything inside the 'B' circle represents the entries in the truth table in which B is true.
- Everything outside the A and B circle represents the entries in which the value of A and B are false.
- Overlaps between A and B circle represents the entries in truth table for which both A and B are true. The AND of both A and B.



Q) Find the Boolean function from the following Venn diagram.



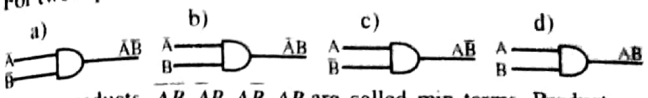
Solution:

$$Y = AB'C' + AB'C + ABC + ABC'$$
$$= AB'(C + C') + AB(C + C') \qquad [C + C' = 1]$$
$$= AB' + AB$$

3.3 Sum of product (SOP) method

We have four method to AND two input variable and eight ways to AND three input variable. So, when we get the output from two or three input variable then it is called fundamental product or min terms.

For two input A and B



Here, products $\bar{A}\bar{B}, \bar{A}B, A\bar{B}, AB$ are called min terms. Product are represented by m_0, m_1, m_2 and m_3 respectively.

Consider a table

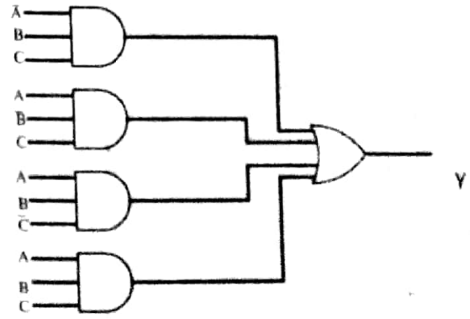
A	B	C	Y	
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	$\bar{A}BC$
1	0	0	0	
1	0	1	1	$A\bar{B}C$
1	1	0	1	$AB\bar{C}$
1	1	1	1	ABC

Consider a equation:

$$Y = \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$$
$$Y = F(ABC) = \sum_M (3,5,6,7)$$

Here Σ symbolize summation or logic OR operation that is performed in corresponding min terms Y=F (A, B, C) means Y is the function of three Boolean variables A,B and C. This kind of representation of truth table is called conical sum form.

Logic circuit



3.4 Truth table to karnaugh map

A karnaugh map is the visual display of the fundamental products needed for the sum of the product solution. For n variable problem it requires 2^n location in karnaugh map.

For three variable problem it requires 2^3 location.

	$\overline{A}\overline{B}$	$\overline{A}B$	AB	$A\overline{B}$
$\overline{X}$	0	1	3	2
X	4	5	7	6

Similarly for 4 variable problem it requires 2^4 location.

	$\overline{C}\overline{D}$	$\overline{C}D$	CD	$C\overline{D}$
$\overline{A}\overline{B}$	0	1	3	2
$\overline{A}B$	4	5	7	6
AB	12	13	15	14
$A\overline{B}$	8	9	11	10

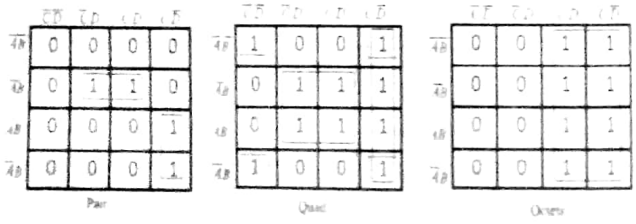
3.5 Pair, quads and octets

Two adjacent 1 is called pair

Four adjacent 1 is called quads

Eight adjacent 1 is called octets

First of all we have to observe octets, if there is octets we have to group them, secondly we have to group quads and in last we have to group pair



Pair Quad Octets

3.6 Karnaugh simplification

A pair eliminate one variable and its complement, a quad eliminate two variable and their complement and octets eliminate three variable and their complement. Because of this we have to use octets first then quad and in last pair.

Suppose a truth table as:

	$\overline{C}\overline{D}$	$\overline{C}D$	CD	$C\overline{D}$
$\overline{A}\overline{B}$	0	0	1	1
$\overline{A}B$	0	0	0	0
AB	0	0	1	1
$A\overline{B}$	0	0	1	1

First look for octets. In above k-map there is no any octet. Next look for quads. When you find them encircle them. Finally look for the pair if there is pair then encircle pair.

By ORing these simplified products we get the Boolean equation corresponding to the entire karnaugh map.

$Y = AC + \overline{A}\overline{B}C$

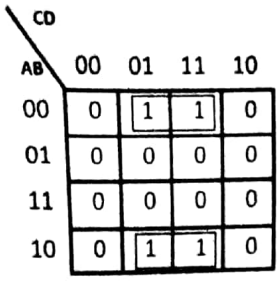
Overlapping group:

We can use same 1 more than once. So always overlap the group if possible.

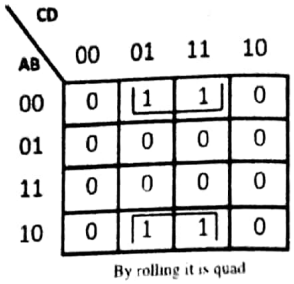
AB \ CD	00	01	11	10
00	0	0	0	0
01	0	0	1	0
11	1	1	1	1
10	1	1	1	1

Rolling group:

We can roll the map if it is possible. If there is a pair and rolling is possible at that place to form a quad then we follow a rolling method.



Pair

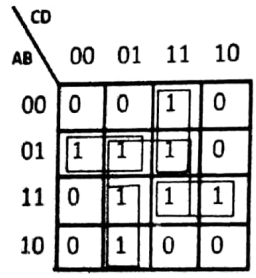


By rolling it is quad

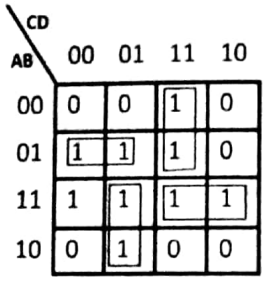
Note: we use overlap and rolling to get largest group as possible.

Redundant group:

When there is redundant group eliminate it



a) Redundant group



b) Elimination of Redundant group

In above k-map all the 1s of quad are used by pair. Because of this, quad is redundant and can be eliminated as b.

3.7 Don't care condition

When the certain input condition in some digital system never occurs during the normal operation so that output never occurs, then is called don't care conditions. It is indicated by 'x'. 'x' represent either 0 or 1. We can make pair, quad and octet from 'x' with 1 or with 0.

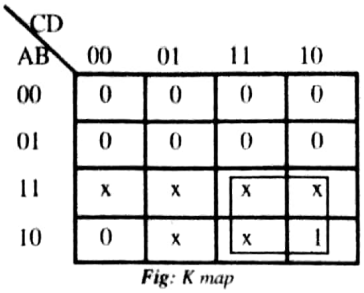


Fig: K map

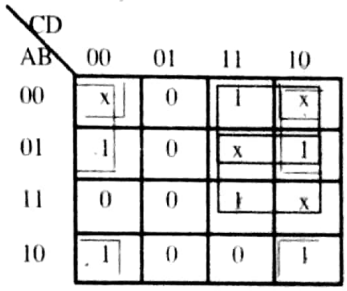
Suppose a truth table and draw a karnaugh map as above with 0's, 1's and don't care i.e. x. encircle the actual 1's on the karnaugh map in the larger group you can find by treating don't care as 1. After the actual 1's have been included in group, neglect the remaining don't care by visualizing them as 0's.

Q) Simplify the following using k map.

$\sum_m (3,4,6,8,10,15) + d(0,2,7,14)$

= Here, $\sum_m$

Prepresent sum of the product terms and d represent don't care condition.



Expression in sum of product (sop)

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

3.8 Product of sum method

In product of sum method, the fundamental sum produces an output 0 from the corresponding input condition. But in the sum of product method the fundamental product produces an output 1 from the corresponding condition. A product-of-sums expression contains the product of different terms, with each term being either a single literal or a sum of more than one literal. It can be obtained

from the truth table by considering those input combinations that produce a logic '0' at the output. Each such input combination gives a term, and the product of all such terms gives the expression. Different terms are obtained by taking the sum of the corresponding literals. Here, '0' and '1' respectively mean the uncomplemented and complemented variables, unlike sum-of-products expressions where '0' and '1' respectively mean complemented and uncomplemented variables.

3.9 Product of sum simplification

Q) Simplify the expression Y= Π(0, 1, 2, 3, 4, 5, 6, 7, 9, 11) using k map.

= Here Π represent product of sum terms.

We can use same 1 more than once. So always overlap the group if possible. .

CD / AB	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	1	0	0	1

Simplified POS expression is

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

Q) Draw a k map and simplify Y= F(A,B,C,D)= Π(0,1,3,8,9,10,14,15)

= In k map

CD / AB	00	01	11	10
00	0	0	0	1
01	1	1	1	1
11	1	1	0	0
10	0	0	1	0

Simplified POS expression is

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

3.10 Hazard and hazard cover

Ideally when we provide input to gate then it generate output instantaneously. But practical circuit always offer finite propagation delay through very small in Nano second order. This gives raise to the several hazard. So, hazard cover are additional term in an equation that prevent occurring of them.

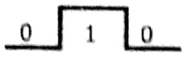
There are two types of hazard they are:

Static hazard:

A hazard in which change in a single input variable produces a instantaneous change in output when the output is expected to remain in a same state, it is called a static hazard. It is classified in to two types:

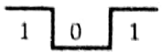
a) **Static 0 hazard:**

When the output instantaneous goes to a state 1 but the output should have to be in state 0, it is known as static 0 hazard. Input changes causes output goes from 0 to 1 to 1 to 0.



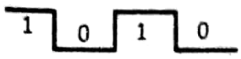
b) **Static 1 hazard:**

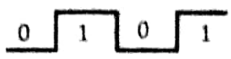
When the output instantaneous goes to a state 0 but the output should have to be in state 1, it is known as static 1 hazard. Input changes causes output goes from 1 to 0 to 0 to 1.



Dynamic hazard:

It occurs when the circuit output make multiple transition before it settle to a final value while the logic equation asks for only one transition.





Example of hazard and its removal:

Let us take an example

$F(A,B,C) = \sum m(0,1,3,4)$

A	B	C	Output (Y)
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	0
1	1	1	0

Map illustrating hazard

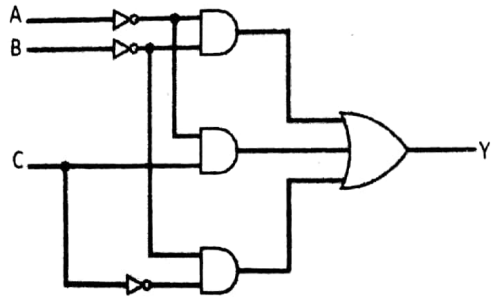
A \ BC	00	01	11	10
0	1	1	1	0
1	1	0	0	0

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

Hazard and its removal

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

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



This method of eliminating of static hazard is known as covering method.

Important Questions

1) Simplify Π(0, 4, 5, 8, 9, 11, 15) using k-map, write the standard sum of product(SOP) expression and realize the function using NAND gate only.

Solution:

F(A,B,C,D) = Π(0,4,5,8,9,11,15)

Here Π represent product of sum term

Using k- map

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

SOP expression for above is

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

Now for using NAND gate only

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

$= \overline{\overline{(AB\overline{C})}\,\overline{(\overline{A}\overline{B}D)}\,\overline{(\overline{A}C)}\,\overline{(C\overline{D})}}$

In circuit:

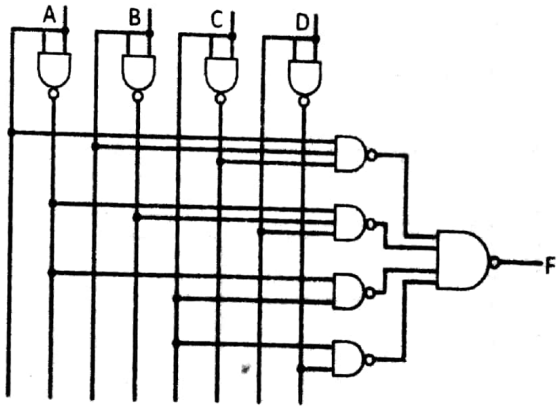


Fig: logic circuit using NAND gate only

2) Simplify $\sum_m(0, 1, 2, 8, 10, 11, 14, 15)$ using k- map, write its standard Product Of Sum(POS) expression and realize the function using NOR gate only.

= $F(A,B,C,D)= \sum_m(0,1,2,8,10,11,14,15)$

Here $\sum_m$ represent sum of product term

Using k-map

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

POS expression of above k-map is:

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

Now, for using NOR gate only

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

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

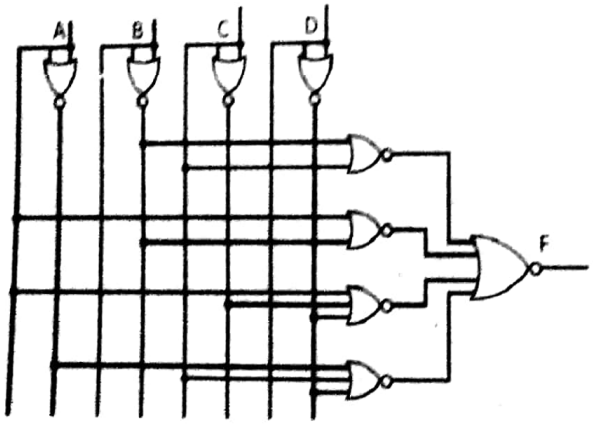


Fig: logic circuit using NOR gate only

3) Use the k map method to implement a following function and also draw a reduced circuit using NOR gate.

$F(A, B, C, D) = \sum m(0, 2, 4, 6, 8, 10, 15)$ and $d = \sum m(3, 11, 14)$ [BE IOE 2070 chaitra]

= Here

$\sum m$ represent sum of product term.

Using K-map

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

To draw a reduced circuit using NOR gate we have to write POS expression which is:

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

Using double inversion in above F we get the circuit using NOR gate only.

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

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

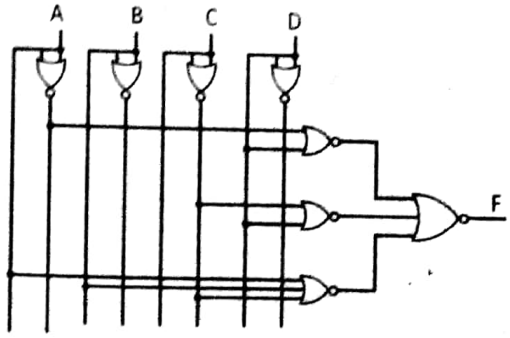


Fig: Logic circuit using NOR gates only

4) **Construct the given Boolean function F= (A+B) (C+D) E using NOR gate only.**

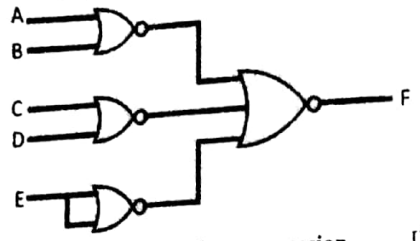
$F = (A + B)(C + D)E$

For using NOR gate only

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

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

Using NOR gate we can get



5) **Simplify the following Boolean expression.** [IOE 2064]

a) **B + AB' + AB'C +A'B + AB'CD' + A'BC'DE'**

= B + A'B + AB' + AB'C +AB'CD' + A'BC'DE'
= B (1+A') + AB' (1+C) +AB'CD'+A'BC'DE'
= B + AB' + AB'CD' + A'BC'DE'
= B + AB' (1+CD') + A'BC'DE'
= B+AB'+A'BC'DE'
= B (1+A'C'DE') + AB'
= B+AB'
= A+B

b) **[{(A+B)'+A}'+{(A+B)'+B}']'**

= {(A+B)'+A}'. {(A+B)'+B}'
= {(A+B)'+A}. {(A+B)'+B}
= (A+B)'+AB
= A'B'+AB
= AOB

6) **Convert the following term into standard minterm: A+B'C** [IOE 2070 Chaitra]

⇒ A+B'C
= A(B + B')(C + C') + B'C(A + A')
= (AB + AB')(C + C') + B'CA + B'CA'
= ABC + ABC' + AB'C + AB'C' + AB'C + A'B'C

The canonical form of the expression requires the terms to be listed in ascending order with no duplicates.

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

Note: Similarly for maxterm

⇒ A+B'C
= (A + B') (A + C)
= (A + B' + CC') (A + C + BB')
= (A + B' + C) (A + B' + C') (A + C + B) (A + C + B')
= (A + B' + C) (A + B' + C') (A + B + C) (A + B' + C)
= (A + B' + C) (A + B' + C') (A + B + C)
= (A + B + C) (A + B' + C) (A + B' + C')

Important Questions

1) Simplify Π(0, 4, 5, 8, 9, 11, 15) using a k-map and write its standard SOP expression. [IOE 2068]
2) Simplify the following Boolean function using k-map F=(W, X, Y, Z) = $\sum m$(0, 2, 5, 8, 9 11,12, 13) [IOE 2068 Shrawan]
3) F(W,X,Y,Z)= $\sum m$(0, 2, 5, 8, 9, 11, 12, 13) [IOE 2068 Shrawan]
4) Prove the following Boolean expression

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

5) AND simplify $\sum m$(1, 2, 3, 8, 9, 10, 11, 14) and d(0, 4, 12) by using k map and write its standard POS expression. [IOE 2067 Ashad]
6) Obtain the reduced Boolean expression for F(0,4) and construct your circuit using NOR gate. [IOE 2067 Magh]
7) Simplify Boolean expression [IOE 2064]

$$B + A\overline{B} + A\overline{B}C + \overline{A}B + A\overline{B}C\overline{D} + \overline{A}B\overline{C}D\overline{E}$$

[{(A+B)'+A}'+{(A+B)'+B}']'

8) Simplify the following using k-map F=$\sum_m$(0, 1, 4, 8, 10, 11, 12) and D=$\sum$(2,3,6,9,15) also convert the result in standard min term. [IOE 2068]

9) Simplify F(A,B,C,D) = $\Pi(0,2,5,8,10)+d(7,15)$ write its standard SOP and implement the simplified circuit using NOR gate only.

10) Write a simplified maxterm Boolean expression for $\Pi(0, 4, 5, 6, 7, 10, 14)$ using the Karnaugh mapping method.

11) What do you mean by Don't care condition? What is its role in Boolean expression? [Purbanchal 2005]

12) Design a logic circuit to implement the Boolean function

$F(A,B,C,D)= \sum(1,3,7,11,15)$

$D(A,B,C,D)=\sum(0,2,5)$ in the term of

a) Sum of products

b) Implement with NAND-NAND gate only

[PU Fall 2013]

13) What is Don't care condition? Simplify given function using K-map with circuit design. $F(W,X,Y,Z)=\Sigma(1,4,5,6,12,14,15)$ and don't care condition $D(W,X,Y,Z)= \Sigma(10,11)$. [PU Spring 2014]

●●●

Chapter 4

Data Processing Circuits

4.1 Multiplexers

A multiplexers is a circuit having number of data inputs, one or many select inputs and one output. It has many inputs but only one output so it is also called many in to one. Here large number of information units are transmitted over a small number of channel or line. The selection of a particular input line is controlled by a set of and selection line. Normally there are n input line, m selection lines and 1 output. Remember that m control signal can select at most 2^m input signal thus $n \leq 2^m$.

Block diagram:

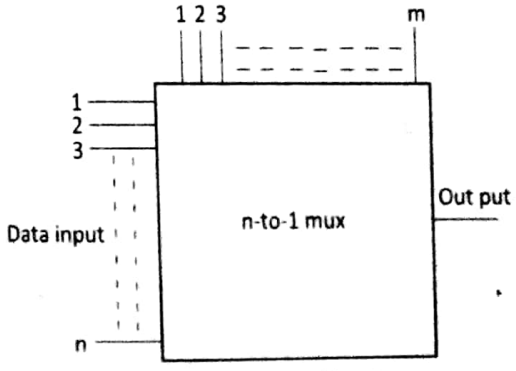


Fig: multiplexer block diagram

4 to 1 multiplexers:

It has four data input ($D_0 to D_3$), two select input ($S_0 and S_1$) and single output. Depending on control input $S_0 and S_1$ one of the four input $D_0 to D_3$ is directed to output Y.

Data select input		Input select	output
S_0	S_1	Y	
0	0	D_0	$\overline{S_0}\,\overline{S_1}D_0$
0	1	D_1	$\overline{S_0}S_1D_1$
1	0	D_2	$S_0\overline{S_1}D_2$
1	1	D_3	$S_0S_1D_3$

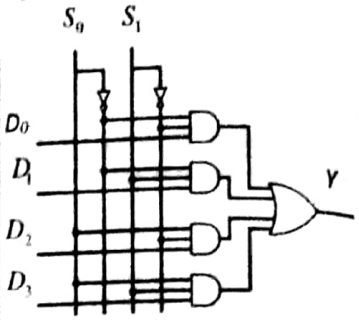


Fig: logic diagram of 4: 1 multiplexer

Universal logic circuits:

Multiplexer sometimes is called universal logic circuit because a 2^n to 1 multiplexer can be used as a design solution for any n variable truth table.

4.2 Demultiplexer

A demultiplexer is a combinational circuit in with one input and many output. It means one in to many. It receives information on a single line and transmits this information on one of 2^n possible output line. It is also known as data distributer.

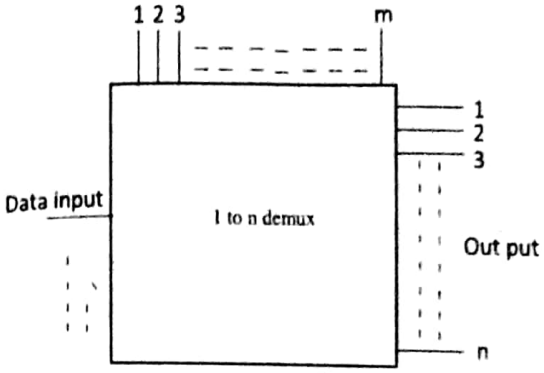


Fig: Block diagram of de-multiplexer

1 to 4 De multiplexer:

Input			Output			
D	S_0	S_1	Y_0	Y_1	Y_2	Y_3
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

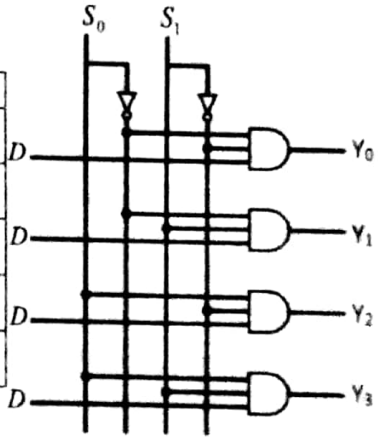


Fig: 1 4 De multiplexer

Boolean expression:

$Y_0 = D\overline{S_0}\,\overline{S_1}$

$Y_1 = D\overline{S_0}S_1$

$Y_2 = DS_0\overline{S_1}$

Implementing Boolean function with MUX:

One of the most common applications of a multiplexer is its use for implementation of combinational logic Boolean functions. The simplest technique for doing so is to employ a 2n-to-1 MUX to implement an n-variable Boolean function.

- The input lines corresponding to each of the Minterms present in the Boolean function are made equal to logic '1' state.
- The remaining Minterms that are absent in the Boolean function are disabled by making their corresponding input lines equal to logic '0'.

Implementing Boolean function with MUX:

One of the most common applications of a multiplexer is its use for implementation of combinational logic Boolean functions. The simplest technique for doing so is to employ a 2n-to-1 MUX to implement an n-variable Boolean function.

- The input lines corresponding to each of the Minterms present in the Boolean function are made equal to logic '1' state.
- The remaining Minterms that are absent in the Boolean function are disabled by making their corresponding input lines equal to logic '0'.

Implement Boolean function f(A, B, C) = ∑(1, 4, 7) using 8:1 MUX and 4:1 MUX.

Solution: Y_2

f(A, B, C) = ∑(1, 4, 7)

Implementing using 8:1 MUX:

- The input lines corresponding to each of the Minterms present in the Boolean function are made equal to logic '1' state. Here 3, 4, 7 are minterms.
- The remaining Minterms that are absent in the Boolean function are disabled by making their corresponding input lines equal to logic '0'.

A	B	C	Y	
0	0	0	0	D_0
0	0	1	1	D_1
0	1	0	0	D_2
0	1	1	0	D_3
1	0	0	1	D_4
1	0	1	0	D_5
1	1	0	0	D_6
1	1	1	1	D_7

Table 1

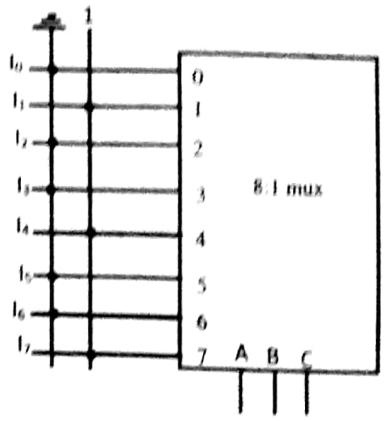


Fig: 8:1 mux

This can be implemented using 4:1 MUX as follow:

In terms of variable:

$f(A, B, C) = ABC' + AB'C' + ABC$ (1)

Steps:

- Chosen multiplexer with two selection lines.
- In the next step, two of the three variables are connected to the two selection lines, with the higher order variable connected to the higher-order selection line. For instance, in the present case, variables B and C are the chosen variables for the selection lines and are respectively connected to selection lines S_1 and S_0.
- In the third step, a table of the type shown below is constructed. Under the inputs to the multiplexer, minterms are listed in two rows, as shown. The first row lists those terms where remaining variable A is complemented in Table 1, and second row lists those terms where A is uncomplemented in Table 1.

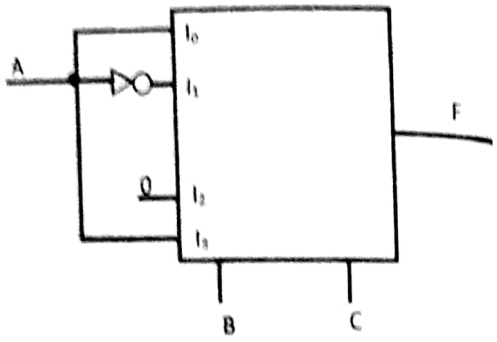
A′	I_0	I_1	I_2	I_3
A	0	1	2	3
A	4	5	6	7
	0	A′	A	

Table 2

The required minterms are identified or marked in the given Table 2, these entries have been highlighted. Each column is inspected individually.

- If neither minterm of a certain column is highlighted, a '0' is written below that.
- If both are highlighted, a '1' is written.
- If only one is highlighted, the corresponding variable (complemented or uncomplemented) is written.

The input lines are then given appropriate logic status. In the present case, I_0, I_1, I_2 and I_3 would be connected to A, A', 0 and A respectively. Below figure shows the logic implementation.



It is not necessary to choose only the leftmost variable in the sequence to be used as input to the multiplexer. Here we had chosen 'A' but either 'B' or 'C' can also be used.

Q) Realize Y=A'B+B'C+ABC using an 8 to 1 multiplexer.

Here,

$Y = A'B+B'C+ABC$

$Y = A'B(C'+C) + B'C(A'+A) + ABC \quad (x + x' = 1)$

$Y = A'BC'+A'BC+A'B'C+AB'C+ABC$

Now,

$Y = \sum(0,2,3,4,7)$

The input lines corresponding to the five minterms present in the given Boolean function are tied to logic '1'. The remaining three possible minterms absent in the Boolean function are tied to logic '0'.

A	B	C	Y	
0	0	0	1	D_0
0	0	1	0	D_1
0	1	0	1	D_2
0	1	1	1	D_3
1	0	0	1	D_4
1	0	1	0	D_5
1	1	0	0	D_6
1	1	1	1	D_7

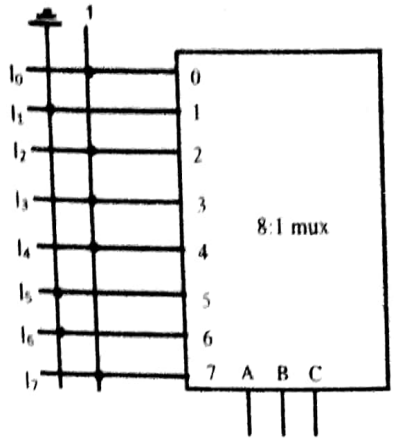


Fig: 8:1 mux

Q) Design a 32 to 1 mux using two 16 to 1 mux and one 2 to 1 mux.

We design 32 to 1 mux using 16 to 1 mux and one 2 to 1 mux as below figure:

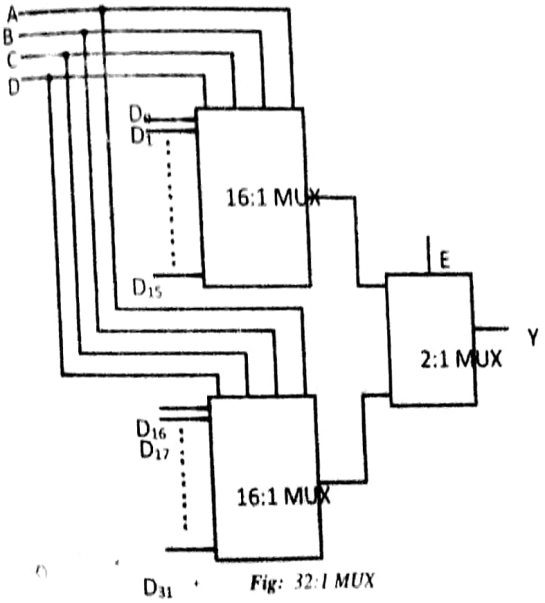


Fig: 32:1 MUX

Note: Similarly we can draw various mux like 16 to 1 mux, 4 to 1 mux using 4 to 1 mux and 2 to 1 mux respectively.

4.3 Decoder

A decoder is a combinational circuit that converts a binary information from n input lines to maximum of 2^n unique output lines. Data input is absent in decoder while inputs are the control bit. It converts coded input in to an active signal. This description of demultiplexer and decoder sounds similar, but a decoder is used to select among many devices while a demultiplexer is used to send a signal among many devices.

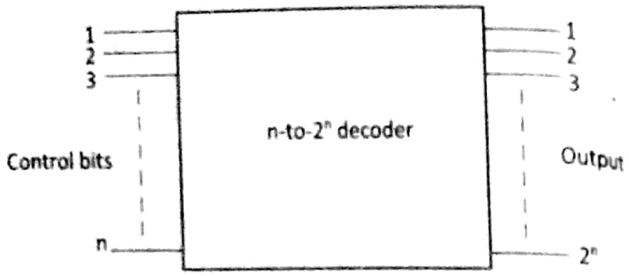


Fig: Block diagram of decoder

3 to 8 line decoder:

Truth table

A	B	C	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Boolean expression

$Y_0 = \bar{A}\bar{B}\bar{C}$
$Y_1 = \bar{A}\bar{B}C$
$Y_2 = \bar{A}B\bar{C}$
$Y_3 = \bar{A}BC$
$Y_4 = A\bar{B}\bar{C}$
$Y_5 = A\bar{B}C$
$Y_6 = AB\bar{C}$
$Y_7 = ABC$

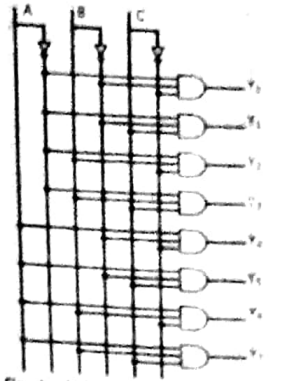


Fig: circuit diagram of 3 to 8 line decoder

4.4 BCD to decimal decoder

The BCD to decimal decoder converts each BCD code to its decimal equivalent. The technique employed is very similar to the one used in developing the 3-line-to-8 line decoder. It is referred as 4-line-to 10-line or 1 of 10 decodes.

Truth table:

Decimal digit	Binary input				Logic function
0	0	0	0	0	$\bar{A}\bar{B}\bar{C}\bar{D}$
1	0	0	0	1	$\bar{A}\bar{B}\bar{C}D$
2	0	0	1	0	$\bar{A}\bar{B}C\bar{D}$
3	0	0	1	1	$\bar{A}\bar{B}CD$
4	0	1	0	0	$\bar{A}B\bar{C}\bar{D}$
5	0	1	0	1	$\bar{A}B\bar{C}D$
6	0	1	1	0	$\bar{A}BC\bar{D}$
7	0	1	1	1	$\bar{A}BCD$
8	1	0	0	0	$A\bar{B}\bar{C}\bar{D}$
9	1	0	0	1	$A\bar{B}\bar{C}D$

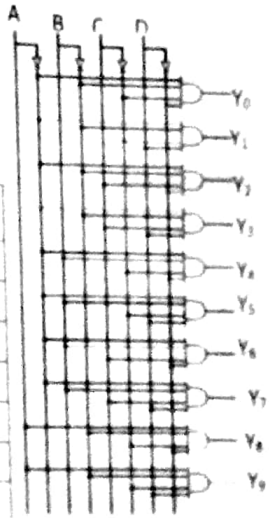
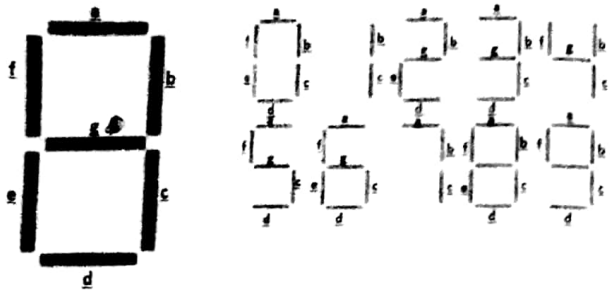


Fig: circuit diagram of BCD to decimal decoder

Now we can develop a decoder based on each logic function and implement the SOP logic circuits. This is shown in fig below.

4.5 Seven segment decoders

A seven segment display is a form of electronic display device for displaying decimal numbers. It accepts the BCD code on its input and provides output to energize seven segment display device in order to produce a decimal.



To design seven segment display consider

Number of input: A, B, C, D

Number of output: a, b, c, d, e, f, g

Display format:

Decimal number	Segment activated
0	a, b, c, d, e, f
1	b, c
2	a, b, g ,e ,d
3	a, b, c, d, g
4	b, c, f, g
5	a, c, d, f, g
6	a, c, d, e, f, g
7	a, b, c
8	a, b, c, d, e, f, g
9	a, b, c, d, f, g

Truth table:

Inputs				Outputs						
A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1

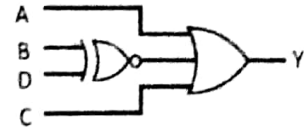
1) For segment a:

For segment "a" 0, 2, 3, 5, 6, 7, 8, 9 is high.

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

$$a = \overline{BD} + BD + C + A$$
$$= A + C + BD + \overline{BD}$$
$$= A + C + BOD$$

In circuit



For segment b:

For segment "b" 0, 1, 2, 3, 4, 7, 8, 9 is high

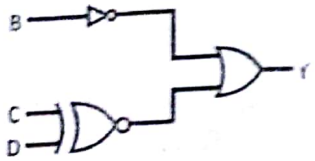
AB \ CD	00	01	11	10
00	1	1	1	1
01	1	0	1	0
11	x	x	x	x
10	1	1	x	x

In circuit

$b = \overline{CD} + CD + \overline{B}$

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

$= \overline{B} + COD$



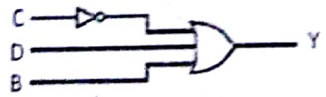
2) **For segment c:**

For segment "c" 0, 1, 3, 4, 5, 6, 7, 8, 9 is high

AB \ CD	00	01	11	10
00	1	1	1	0
01	1	1	1	1
11	x	x	x	X
10	1	1	x	X

In circuit

$c = \overline{C} + D + B$



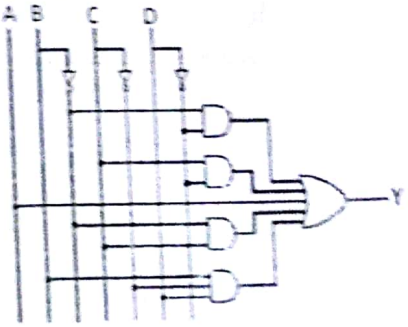
3) **For segment d:**

For segment "d" 0, 2, 3, 5, 6, 8, 9 is high

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

In circuit

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



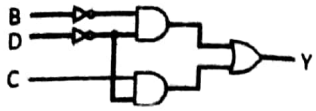
4) **For segment e:**

For segment "e" 0, 2, 6, 8 is high

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

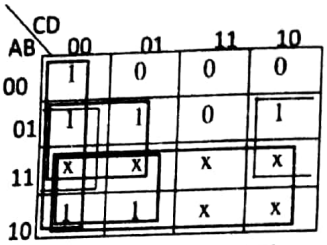
In circuit

$e = C\overline{D} + \overline{B}\overline{D}$



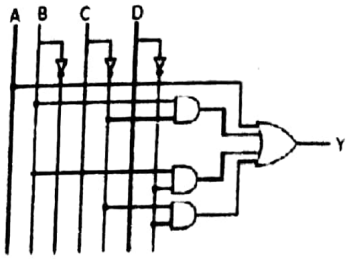
5) For segment f:

For segment "f"0, 4, 5, 6, 8, 9 is high



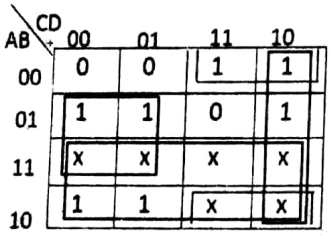
In circuit

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



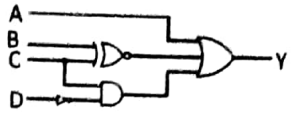
6) For segment g:

For segment "g"2, 3, 4, 5, 6, 8, 9 is high



In circuit

$g = A + B\overline{C} + C\overline{D} + \overline{B}C$
$= A + C\overline{D} + B\overline{C} + \overline{B}C$
$= A + C\overline{D} + BOC$



Q) Using a seven segment display decoder realise the logic circuit for segment 'b','c'and'd'. *[IOE 2071 shrawan]*

4.6 Encoder

A multiplexer selects one individual data input line and then sends that data to a single output line, digital encoder (binary encoder) takes all its data inputs one at a time and then converts them into a single encoded output. So we can say that a binary encoder, is a multi-input combinational logic circuit that converts the logic level "1" data at its inputs into an equivalent binary code at its output. It is a combinational logic function that has 2^n (or fewer) input lines and n output lines, which correspond to n selection lines in a multiplexer. The n output lines generate the binary code for the possible 2^n input lines.

Generally, digital encoders produce outputs of 2-bit, 3-bit or 4-bit codes depending upon the number of data input lines. An "n-bit" binary encoder has 2^n input lines and n-bit output lines with common types that include 4-to-2, 8-to-3 and 16-to-4 line configurations. The output lines of a digital encoder generate the binary equivalent of the input line whose value is equal to "1" and are available to encode either a decimal or hexadecimal input pattern to typically a binary or "B.C.D" (binary coded decimal) output code.

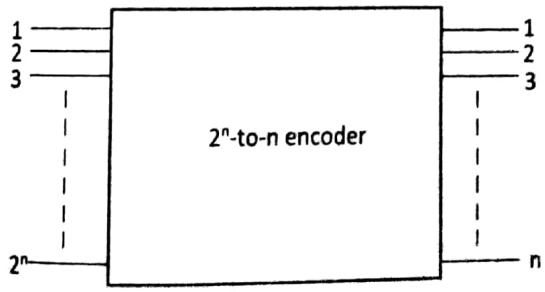


Fig: *Block diagram of encoder*

Disadvantage:

Consider in case of 8 to 3 line encoder in which, if D_2 and D_5 are 1 simultaneously, the encoder produces the output as 111 that means all three outputs are equal to logic 1 (as an example in case of a person operating keyboard might press a second key before releasing the first). This value does not correspond to either binary 2 or binary 5.

To overcome this problem, encoder circuit must establish a priority such that only one input is encoded at such cases. This means that whenever two inputs are equal to logic 1 simultaneously, then the encoder must priorities the level of each input such that it produce output corresponds to highest priority input. Such an encoder is called as priority encoder.

Octal to binary priority encoder:

An encoder having eight input lines, each representing an octal digit, and three output lines representing the three-bit binary equivalent is called octal to binary encoder. The truth table of such an encoder is given below:

INPUT								OUTPUT		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	X	Y	Z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

Here D_0 to D_7 represent octal digits 0 to 7 and X, Y, Z represents the binary digits. The eight input lines would have 28 = 256 possible combinations. However, in the case of an octal-to-binary encoder, only eight of these 256 combinations would have any meaning. The remaining combinations of input variables are

'Don't care' input combinations. Also, only one of the input lines at a time is in logic '1' state.

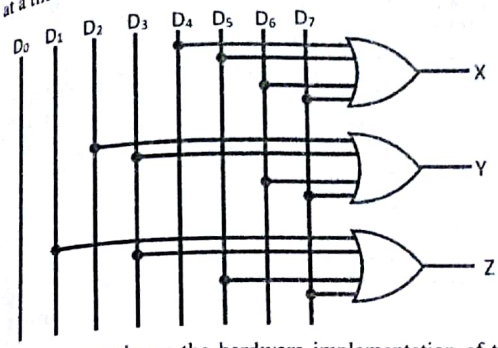


Figure above shows the hardware implementation of the octal-to-binary encoder described by the truth table in Table above. This circuit has the shortcoming that it produces an all 0's output sequence when all input lines are in logic '0' state. This can be overcome by having an additional line to indicate an all 0's input sequence.

Priority encoder:

A priority encoder is a practical form of an encoder. The encoders available in IC form are all priority encoders. In this type of encoder, a priority is assigned to each input so that, when more than one input is simultaneously active, the input with the highest priority is encoded.

Example.

Let us assume that the octal-to-binary encoder described above previous has an input priority for higher-order digits. Let us also assume that input lines D_2, D_4 and D_7 are all simultaneously in logic '1' state. In that case, only D_7 will be encoded and the output will be 111. The truth table of such a priority encoder is shown below:

Disadvantage:

Consider in case of 8 to 3 line encoder in which, if D_2 and D_5 are 1 simultaneously, the encoder produces the output as 111 that means all three outputs are equal to logic 1 (as an example in case of a person operating keyboard might press a second key before releasing the first). This value does not correspond to either binary 2 or binary 5.

To overcome this problem, encoder circuit must establish a priority such that only one input is encoded at such cases. This means that whenever two inputs are equal to logic 1 simultaneously, then the encoder must priorities the level of each input such that it produce output corresponds to highest priority input. Such an encoder is called as priority encoder.

Octal to binary priority encoder:

An encoder having eight input lines, each representing an octal digit, and three output lines representing the three-bit binary equivalent is called octal to binary encoder. The truth table of such an encoder is given below:

INPUT								OUTPUT		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	X	Y	Z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

Here D_0 to D_7 represent octal digits 0 to 7 and X, Y, Z represents the binary digits. The eight input lines would have 28 = 256 possible combinations. However, in the case of an octal-to-binary encoder, only eight of these 256 combinations would have any meaning. The remaining combinations of input variables are 'Don't care' input combinations. Also, only one of the input lines at a time is in logic '1' state.

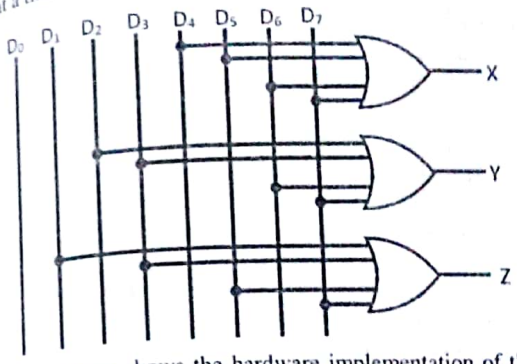


Figure above shows the hardware implementation of the octal-to-binary encoder described by the truth table in Table above. This circuit has the shortcoming that it produces an all 0's output sequence when all input lines are in logic '0' state. This can be overcome by having an additional line to indicate an all 0's input sequence.

Priority encoder:

A priority encoder is a practical form of an encoder. The encoders available in IC form are all priority encoders. In this type of encoder, a priority is assigned to each input so that, when more than one input is simultaneously active, the input with the highest priority is encoded.

Example.

Let us assume that the octal-to-binary encoder described above previous has an input priority for higher-order digits. Let us also assume that input lines D_2, D_4 and D_7 are all simultaneously in logic '1' state. In that case, only D_7 will be encoded and the output will be 111. The truth table of such a priority encoder is shown below:

INPUT								OUTPUT		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	X	Y	Z
1	0	0	0	0	0	0	0	0	0	0
x	1	0	0	0	0	0	0	0	0	1
x	x	1	0	0	0	0	0	0	1	0
x	x	x	1	0	0	0	0	0	1	1
x	x	x	x	1	0	0	0	1	0	0
x	x	x	x	x	1	0	0	1	0	1
x	x	x	x	x	x	1	0	1	1	0
x	x	x	x	x	x	x	1	1	1	1

Looking at the last row of the above table, it implies that, if D7 = 1, then, irrespective of the logic status of other inputs, the output is 111 as D7 will only be encoded

Q) ***What is priority encoder? Design Octal to binary priority encoder.*** *[IOE 2069]*

4.7 Parity generator and checker

The most common error detection code used is the parity bit. A parity bit is the extra bit included with a binary message to make the total number of 1's either even or odd. In case of even parity the parity bit is chosen so that total number of 1's in the coded message is even. Alternately, odd parity can be used in which the total number of 1's in the code message is made odd.

During the transfer of information the message at the sending end is applied to a parity generator where the parity bit is generated. At the receiving end a parity checker is used to detect single bit error in the transmitted data word by recognized the parity bit in the same fashion as the generator and then compare with parity bit transmitted.

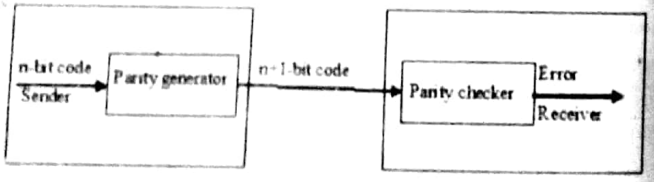


Fig: block diagram of parity generator and checker

Odd and even parity:

- An even parity bit will be set to "1" if the number of 1's + 1 is even
- An odd parity bit will be set to "1" if the number of 1's + 1 is odd

Suppose 110011, it has even parity because it contain four 1s and suppose 110001, it has odd parity because it contains three 1's.

Parity bit table:

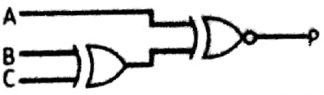
Input bits			Odd parity	Even parity
A	B	C	P	Q
0	0	0	1	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	0	1

In above table we get odd and even parity by counting additional 1 with 1's of ABC. Suppose 010, then number of 1 in it with additional 1 is 2 so it has even parity. Suppose 110, there are 3 number of 1 with additional 1 so it has odd parity.

K-map for odd parity:

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

$$P = \bar{A}\bar{B}\bar{C} + \bar{A}BC + A\bar{B}C + AB\bar{C}$$
$$= \bar{A}(\bar{B}\bar{C} + BC) + A(\bar{B}C + B\bar{C})$$
$$= \bar{A}(BOC) + A(B \oplus C)$$
$$= \bar{A}(\overline{B \oplus C}) + A(B \oplus C)$$
$$= A \oplus (BOC)$$



K-map for even parity:

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

$$P = \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C} + ABC$$
$$= \bar{A}(\bar{B}C + B\bar{C}) + A(BC + \bar{B}\bar{C})$$
$$= \bar{A}(B \oplus C) + A(B \odot C)$$
$$= \bar{A}(B \oplus C) + A(\overline{B \oplus C})$$
$$= A \oplus B \oplus C$$

Parity checker:

Exclusive OR gate is ideal for checking the parity of a binary number because they produce an output 1 when the input has an odd number of 1's. Therefore an even parity input to exclusive OR gate produce a low output, while an odd parity input produces a high output.

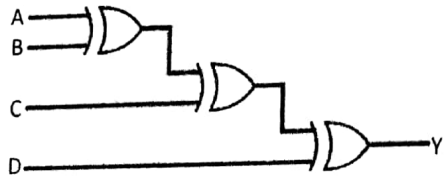


Fig: 4-bit parity checker

4.8 Magnitude comparator

It is the combinational circuit which compares the logic input and produces the respective result as its output. It compares two number say A and B and generate one of following output they are A = B, A < B, A > B.

One bit magnitude comparator:

It is the combinational logic circuit with two input A and B and three output A = B, A < B, A > B. It compares the two single bit number A and B and produces an output that result the result of the comparison.

Truth table:

Input		Output		
A	B	Y_1= A<B	Y_2= A=B	Y_3=A>B
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

Logic equation:

$$A < B : \bar{A}B$$
$$A = B : \bar{A}\bar{B} + AB = A \odot B$$
$$A > B : A\bar{B}$$

Logic diagram of single bit magnitude comparator:

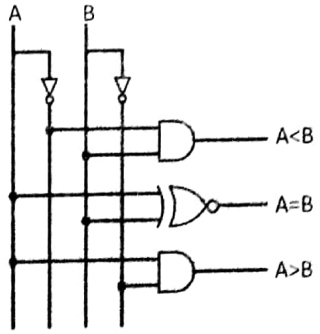


Fig: single bit magnitude comparator

Two bit magnitude comparator:

Two bit magnitude comparator compares two number each having two bits (A_1 and A_0, $B_1$and B_2). For this arrangement truth table has 4 input and 16 entire.

A_1	A_0	B_1	B_0	A<B	A=B	A>B
0	0	0	0	0	1	0
0	0	0	1	1	0	0
0	0	1	0	1	0	0
0	0	1	1	1	0	0
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	1	0	0
0	1	1	1	1	0	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	0	1	0
1	0	1	1	1	0	0
1	1	0	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	0	0	1
1	1	1	1	0	1	0

K-map for A<B

A_1A_0 \ B_1B	00	01	11	10
00	0	1	1	1
01	0	0	1	1
11	0	0	0	0
10	0	0	1	0

$$(A<B)=\overline{A_1}B_1+\overline{A_1}\,\overline{A_0}B_0+\overline{A_0}B_1B_0$$

K-map for A=B

A_1A_0 \ B_1B	00	01	11	10
00	1	0	0	0
01	0	1	0	0
11	0	0	1	0
10	0	0	0	1

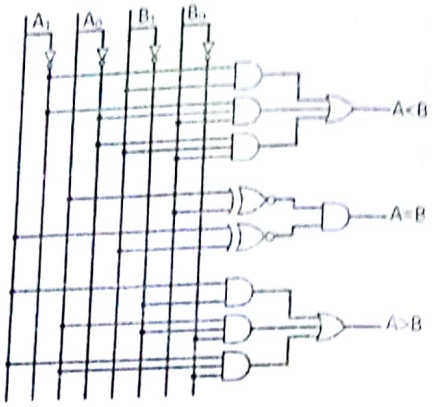
$$(A=B)=\overline{A_1}\,\overline{A_0}\,\overline{B_1}\,\overline{B_0}+\overline{A_1}A_0\overline{B_1}B_0+A_1A_0B_1B_0+A_1\overline{A_0}B_1\overline{B_0}$$
$$=\overline{A_1}\,\overline{B_1}(\overline{A_0}\,\overline{B_0}+A_0B_0)+A_1B_1(A_0B_0+\overline{A_0}\,\overline{B_0})$$
$$=(\overline{A_0}\,\overline{B_0}+A_0B_0)(\overline{A_1}\,\overline{B_1}+A_1B_1)$$
$$=(A_0OB_0)(A_1OB_1)$$

K-map for A>B

A_1A_0 \ B_1B	00	01	11	10
00	0	0	0	0
01	1	0	0	0
11	1	1	0	1
10	1	1	0	0

$$(A>B)=A_1\overline{B_1}+A_0\overline{B_1}\,\overline{B_0}+A_1A_0\overline{B_0}$$

Logic diagram:



4-bit magnitude comparator

Inputs: 8-bits (A ⇒ 4-bits, B ⇒ 4-bits)

A and B are two 4-bit numbers

- Let $A = A_3A_2A_1A_0$, and
  Let $B = B_3B_2B_1B_0$
- Inputs have 2^8 (256) possible combinations
- Not easy to design using conventional techniques

Inputs				Outputs		
Comparing						
A3, B3	A2, B2	A1, B1	A0, B0	A < B	A = B	A > B
A3 > B3	x	x	x	0	0	1
A3 = B3	A2 > B2	x	x	0	0	1
A3 = B3	A2 = B2	A1 > B1	x	0	0	1
A3 = B3	A2 = B2	A1 = B1	A0 > B0	0	0	1
A3 = B3	A2 = B2	A1 = B1	A0 = B0	0	0	1
A3 = B3	A2 = B2	A1 = B1	A0 = B0	0	1	0
A3 = B3	A2 = B2	A1 = B1	A0 = B0	1	0	0
A3 = B3	A2 = B2	A1 = B1	A0 = B0	1	1	0
A3 = B3	A2 = B2	A1 = B1	A0 < B0	1	0	0
A3 = B3	A2 = B2	A1 < B1	x	1	0	0
A3 = B3	A2 < B2	x	x	1	0	0
A3 < B3	x	x	x	1	0	0

Now,

$$Y(A=B) = \overline{(A_3 \oplus B_3)}.\overline{(A_2 \oplus B_2)}.\overline{(A_1 \oplus B_1)}.\overline{(A_0 \oplus B_0)}$$
$$= E_3E_2E_1E_0$$

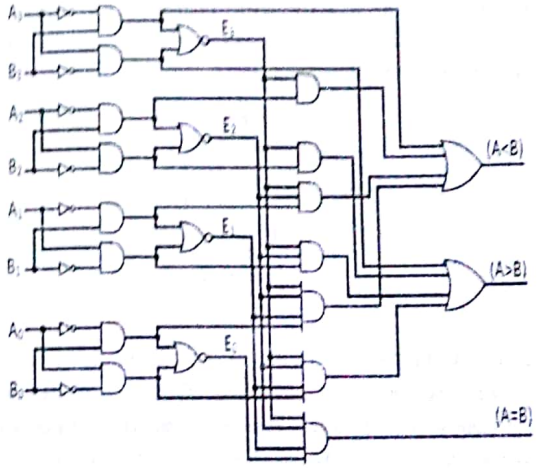
Where,

$$E_3 = \overline{(A_3 \oplus B_3)},\ E_2 = \overline{(A_2 \oplus B_2)},\ E_1 = \overline{(A_1 \oplus B_1)},\ E_0 = \overline{(A_0 \oplus B_0)}$$

$$Y(A>B) = A_3\overline{B_3} + E_3A_2\overline{B_2} + E_3E_2A_1\overline{B_1} + E_3E_2E_1A_0\overline{B_0}$$

Also,

$$Y(A<B) = \overline{A_3}B_3 + E_3\overline{A_2}B_2 + E_3E_2\overline{A_1}B_1 + E_3E_2E_1\overline{A_0}B_0$$



Note:

For n-bit magnitude comparator:

$$Y(A=B) = E_{n-1}E_{n-2}\ldots\ldots E_0$$
$$Y(A>B) = G_{n-1} + E_{n-1}G_{n-2} + \ldots\ldots + E_{n-1}E_{n-2}\ldots E_1G_0$$
$$Y(A<B) = L_{n-1} + E_{n-1}L_{n-2} + \ldots\ldots + E_{n-1}E_{n-2}\ldots E_1L_0$$

Where,

$$G_i = A_i\overline{B_i}, L_i = \overline{A_i}B_i, E_i = \overline{(A_iOB_i)}$$

Q) Design a 2 – bit magnitude comparator. [IOE 2068 Baisakh]

4.9 Read only memory

A ROM is essentially a memory or a storage device in which a fixed set of binary information is stored. A decoder generates the 2.

" min term of the n input variable By inserting OR gate to the sum of min terms of Boolean function, we were able to generate any desired combinational circuits.

A ROM is a device that includes both the decoder and the OR gates within the single IC package. The connection between the output of the decoder and the input of the OR gate can be specified from each particular configuration by "programming" the ROM. ROMs comes with the special internal links that can be fused or broken the desired interconnection for the application requires that certain links be fused to form that required circuit paths. Once a pattern s established far a ROM it remains fixed even when power is turned off and then on again.

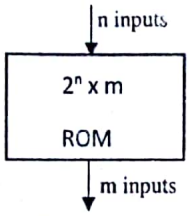


Fig: block diagram of ROM

A block diagram of ROM is shown above. It consists of the n inputs line and m output line. Each bit combination of input variable is called an **address.** Each bit combination that come out of the output line is called a **word.** The number of bit per word is equal to the number of output line m. An address is essentially a binary number that denotes the one of the min terms with n variable. The number of the distinct address possible with n input variable is 2^n. An output word can be selected by a unique address, and since there are 2^n distinct address in ROM, there are 2^n distinct word which are said to be stored in unit. The word available at on the output line at any given time depends on the address value applied to the input lines. A ROM is characteristics by a number of words 2^n and the number of bit per word m.

32 x 4 ROM:

The unit consists of 32 words of 4 bit each. This means there are four output lines and that there are 32 distinct word stored in the unit, each of which may be applied to the output lines. The

particular word selected that is presently available on the output line is determined from the input lines. There are only 5 input in 32 x 4 ROM because $2^5 = 32$ and with the five variable we can specify 32 address or min-terms. For each address input there is a unique select words. Thus if the input address is 0000 word number 0 is select and it appear in the output lines.

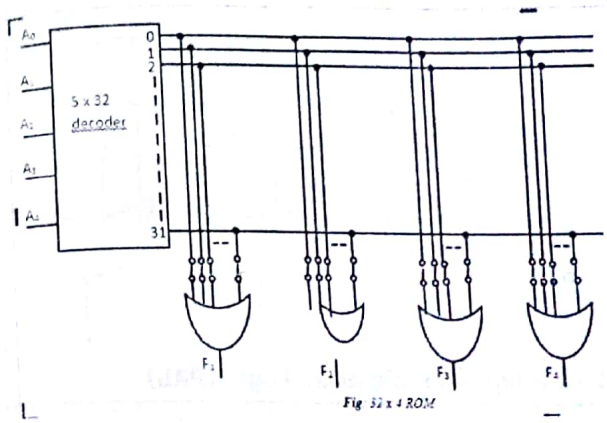


Fig: 32 x 4 ROM

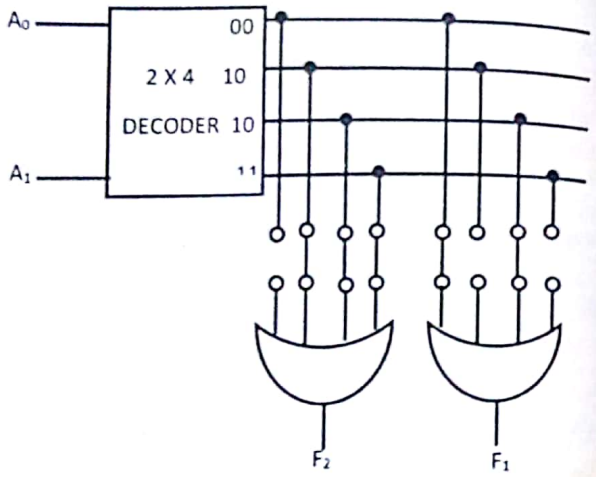
Q. Implement $F_1 = (A_1, A_0) = \sum(1,2,3)$

$F_2 = (A_1, A_0) = \sum(0,2)$

= when the combinational circuit is implemented by a ROM, the function must be expressed in sum of min terms or better yet, by a truth table.

Input		Output	
A_1	A_0	F_1	F_0
0	0	0	1
0	1	1	0
1	0	1	1
1	1	1	0

For four combination we use 2:4 decoder so input are A_1 and A_0 and output are 00, 01, 10, 11.



4.10 Programmable Array Logic (PAL)

Programmable array logic (PAL) architecture has a programmable AND array at the input and a fixed OR array at the output. The programmable AND array of a PAL device is similar to that of a PLA device. That is, the number of programmable AND gates is usually smaller than the number required to generate all possible minterms of the given number of input variables. The OR array is fixed and the AND outputs are equally divided between available OR gates. Figure below shows the internal architecture of a PAL device that has two input lines, an array of four AND gates at the input and two OR gates at the output.

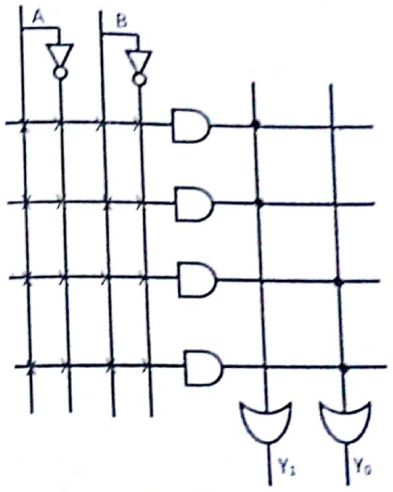


Fig: PAL

4.11 Programmable Logic Array (PLA)

The combinational circuit may occasionally have don't care condition. When implemented with ROM, a don't care condition become an address input that will never occur. The word at the don't care address need not be programmed and may be left in their original state (all 0's or all 1's). The result is that not all the bit pattern available in ROM are used which may be constructed a waste of available equipment.

For the case where don't care is excessive, it is more economical to use programmable logic array or PAL. A PAL is similar to ROM in concept; however the PAL donot provide a full decoding of the variable and donot generate all the min-terms as in the ROM. In the PAL the decoder is replaced by a group of AND gates each of which can be programmed to generate a product terms of the input variables. The AND and OR gates inside the PAL are initially fabricated with a links among them. The specific Boolean function are implemented in sum of product form by opening appropriate links and leaving the desired connection.

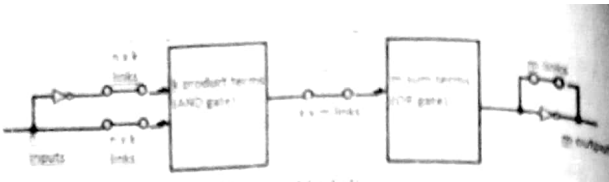


Fig: PLA block diagram

It consists of 'n' inputs, 'm' output, 'k' product terms and 'm' sum term. The product terms constitute a group of k n-gates and the sum term constitute a group of m OR gates. The size of PLA is specified by the number of inputs, the number of product terms and the number of outputs. A typical PLA has 16 inputs, 48 product terms and 8 outputs. The number of programmable links are: $2n \times k + k \times m + m$ where that of a ROM is $2^n \times M$.

PLA programmer table:

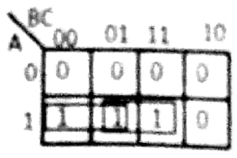
The use of PLA must be considered for combinational circuit that have a large number of inputs and outputs. It is superior to ROM for circuit that have a large number of don't care condition.

Let us take an example with truth table as below:

A	B	C	F_1	F_2
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	0
1	0	1	1	1
1	1	0	0	0
1	1	1	1	1

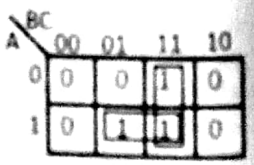
The simplified function in sum of products are obtained from the k-map as below:

K map for F_1



$F_1 = AB' + AC$

K map for F_2



$F_2 = AC + BC$

In this combinational circuits: AB', AC and BC. The circuits has three input and two outputs.

As we know that programming the PLA means we specify the paths in its AND-NOT-OR pattern. A PLA programmer table is as below:

Product term	Input			Output	
	A	B	C	F_1	F_2
1	1	0	-	1	-
2	1	-	1	1	1
3	-	1	1	-	1
				T	T T/C

It consists of three column. The first column lists the product terms numerically. The second column specify the required path between input and AND gates. The third column specify the path between AND gates and OR gates. For each product terms the input are marked with 1, 0 or dash (-). If the variable in the product terms appear in the normal form, the corresponding variable are marked with 1. If it appear with a complimented then marked with 0 and when it is absent then marked with a dash (-).

The output variable i.e. F_1 and F_2 are marked with 1's for all those product terms that formulate the function. Here $F_1 = AB' + AC'$ so F_1 is marked with 1's 1 and 2, with dash for product term 3. Each product term that has a 1 in a output column requires a path for a corresponding AND gates to the output OR gate. Finally the true (T) output dictates that the link across the output invert remains in place, and a complement(C) specifies that the corresponding links be broken.

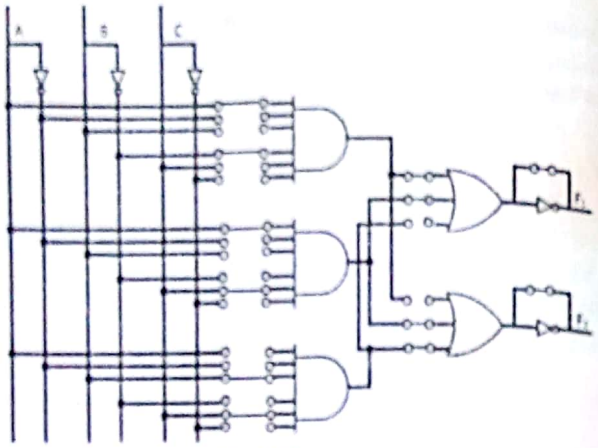


Fig: internal connection of PAL

Alternative method

Implement PAL from the combinational circuit which is defined by the functions

$F_1 = (A, B, C) = \sum(0,5,6)$

$F_2 = (A, B, C) = \sum(1,5,7)$

Truth table:

A	B	C	F_1	F_2
0	0	0	1	0
0	0	1	0	1
0	1	0	0	0
0	1	1	0	0
1	0	0	0	0
1	0	1	1	1
1	1	0	1	0
1	1	1	0	1

$F_2 = \overline{A}B\overline{C} + A\overline{B}C + ABC$

$F_1 = \overline{A}\overline{B}\overline{C} + A\overline{B}\overline{C} + AB\overline{C}$

Programmable OR array

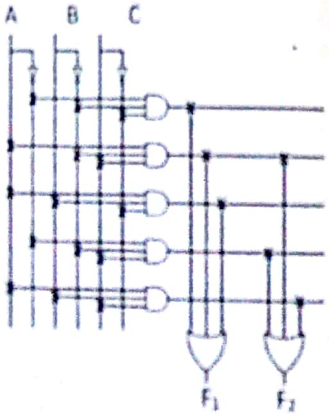
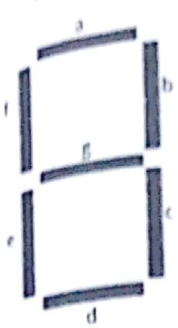
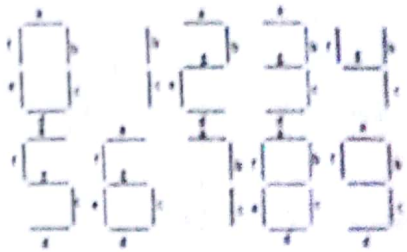


Fig: PAL

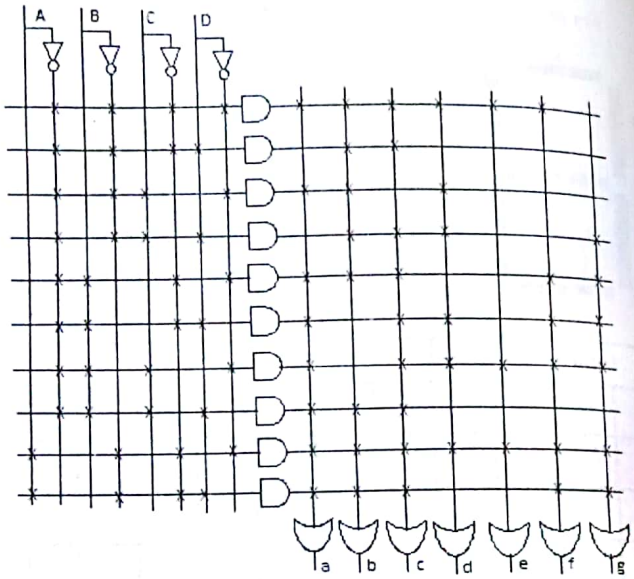
Q) Implement seven segment decoder using PAL.

For seven segment display:





Inputs				Outputs						
A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1



4.12 Trouble shooting with logical probe

The logical probe is the tools which helps us in diagnosing faulty circuits. Figure below shows a logic probe. When we touch the probe tip to the output node as shown the device lights up for the high state and goes dark for the low state. Suppose if either A or B, or both are low, then Y is high and probe lights up. On the other hand, if A and B are both high, Y is low and the probe is dark. The probe is useful for locating the short circuit that occurs in manufacturing.

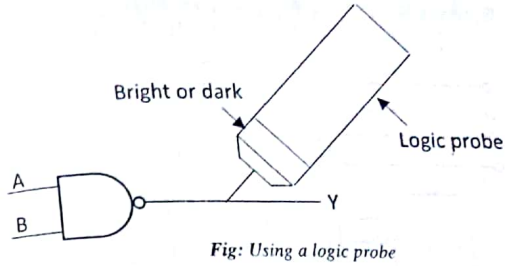


Fig: Using a logic probe

Problem Solutions

1) Construct a 5*32 decoder using 3 to 8 decoders and standard logic gates if necessary.

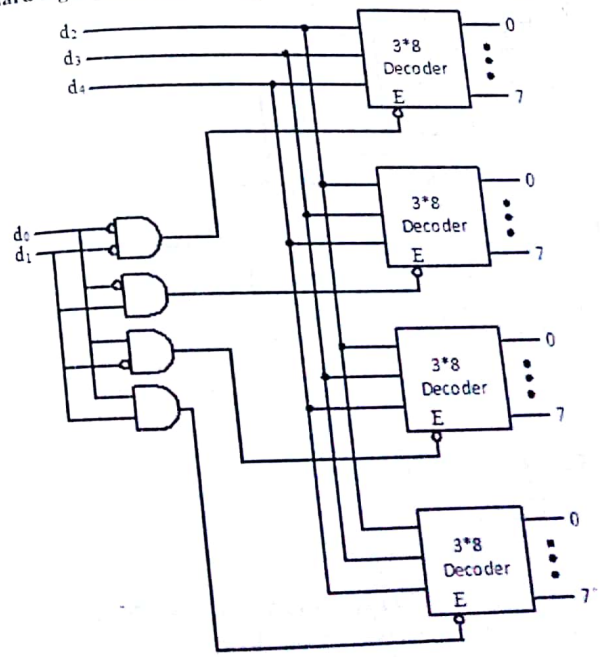


*Fig: Realization of 5*32 decoder using 3*8 decoders and 5*4 decoder*

2) Design a 8 * 1 MUX using two 4 * 1 MUX.

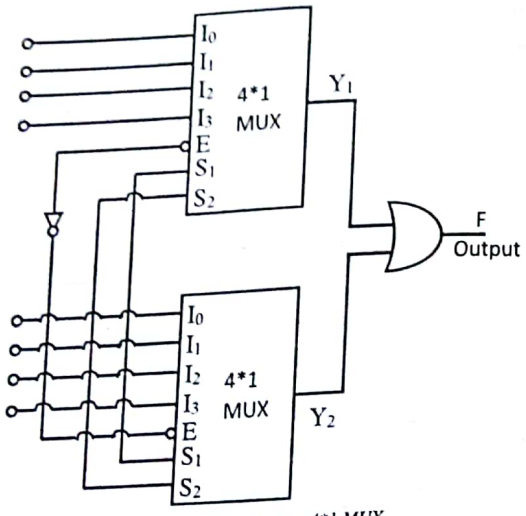


*Fig: 8*1 MUX using two 4*1 MUX*

3) Design a 32-to-1 multiplexer using 8-to-1 multiplexers having an active LOW ENABLE input and a 2-to-4 decoder.

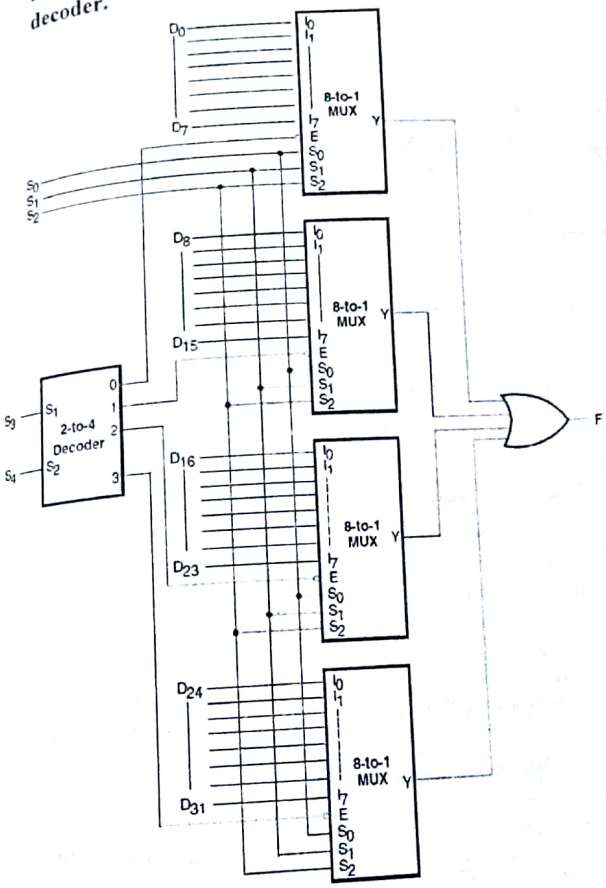


Fig: 32-to-1 multiplexer using 8-to-1 multiplexers

4) **Implement a full subtractor combinational circuit using a 3-to-8 decoder and external NOR gates.**

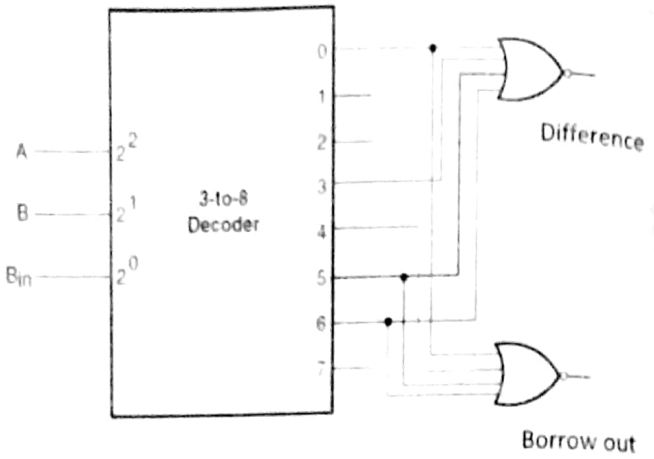


Fig Full-Subtractor using a 3-to-8 decoder

Comparison charts

1. RAM vs. ROM

	Random Access Memory (RAM)	Read Only Memory (ROM)
1	RAM is a form of data storage that can be accessed randomly at any time, in any order and from any physical location, allowing quick access and manipulation.	ROM is also a form of data storage that cannot be easily altered or reprogrammed. Stores instructions that are not necessary for re-booting up to make the computer operate when it is switched off.
2	RAM allows the computer to read data quickly to run applications. It allows reading and writing.	ROM stores the program required to initially boot the computer. It only allows reading.
3	RAM is volatile i.e. its contents are lost when the device is powered off.	It is non-volatile i.e. its contents are retained even when the device is powered off.
4	The two main types of RAM are static RAM and dynamic RAM.	The types of ROM include PROM, EPROM and EEPROM.

2. Static RAM vs. Dynamic RAM

	Static RAM	Dynamic RAM
1	Uses transistor to store a single bit of data.	Uses a separate capacitor to store each bit of data.
2	Does not need periodic refreshment to maintain data.	Needs periodic refreshment to maintain the charge in the capacitors for data.
3	Structure is complex than DRAM.	Structure is simplex than SRAM.
4	Expensive as compared to DRAM.	Less expensive as compared to SRAM.
5	Used in Cache memory.	Used in Main memory.

Important Questions

1) Explain the operation of BCD to decimal decoder with truth table and circuit diagram. Implement 1:4 DEMUX using VHDL. [IOE 2069 Ashad]
2) Design 32 to 1 multiplexer using 16 to 1 and 2 to 1 multiplexers. [IOE 2068 Chaitra]
3) Design a 3 bit even parity generator and 4 bit even parity checker circuit. [IOE 2068 Chaitra]
4) Design a 3 bit even parity generator and parity checker circuit. What do mean by BCD code. [IOE 2068 Shrawan]
5) Construct a 5 x 32 decoder using 3 to 8 decoders and standard logic if necessary. Define the term 'decoder'.
6) Design a 8 x 1 MUX using two 4 x 1 MUX. [IOE 2066 Bhadra]
7) Write down the truth table for an odd parity generator for any two bit word. Design your circuit using any universal gates of your choice. [IOE 2065 Shrawan]
8) State De'morgans theorem. Find out the simple logic circuit for'd' segment of the seven segment display decoder and apply only NAND gates in its logic circuit.
9) Write short notes on de multiplexer. [IOE 2064 Jestha]

10) Design a two bit magnitude comparator. [IOE 2068 Baisakh]
11) With the aid of circuit diagram describe how 2 to 4 decoder can be used to construct 4 to 16 decoder.
12) Why the priority bit are important in digital system? [IOE 2067 Magh]
13) What are priority encoder? Design octal to binary priority encoder. [IOE 2069 Chaitra]
14) What is multiplexer? Describe 4:1 multiplexer with internal diagram. [PU Spring 2014]

Chapter 5

Arithmetic Circuits

Introduction

Arithmetic circuits are nothing but the logic circuits, which is used to compare some arithmetic functions like addition, subtraction, multiplication, division, parity calculation etc. Arithmetic logic circuits are always a combinational logic circuits because the output is totally depend on the current input not depends on the previous inputs and outputs.

5.1 Introduction to Number System

When we type some letters or words, the computer translates them in numbers as computers can understand only numbers. A computer can understand positional number system where there are only a few symbols called digits and these symbols represent different values depending on the position they occupy in the number. A value of each digit in a number can be determined using:

- The digit
- The position of the digit in the number
- The radix (or base) of the number system (where base is defined as the total number of digits available in the number system).

The decimal number system with which we are all so familiar can be said to have a radix of 10 as it has 10 independent digits, i.e. 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9. Similarly, the binary number system with only two independent digits, 0 and 1, is a radix-2 number system.

The octal and hexadecimal number systems have a radix (or base) of 8 and 16 respectively. The place values of different digits in the integer part of the number are given by r^0, r^1, r^2, r^3 and so on, starting with the digit adjacent to the radix point. For the fractional part, these are r^{-1}, r^{-2}, r^{-3} and so on, again starting with the digit next to the radix point. Here, r is the radix of the number system. Also, maximum numbers that can be written with n digits in a given number system are equal to r^n.

5.1.1 Decimal Number System

The number system that we use in our day-to-day life is the decimal number system. Decimal number system has base 10 as it uses 10 digits from 0 to 9. In decimal number system, the successive positions to the left of the decimal point represent units, tens, hundreds, thousands and so on.

The place values of different digits in a mixed decimal number, starting from the decimal point, are 10^0, 10^1, 10^2 and so on (for the integer part) and 10^{-1}, 10^{-2}, 10^{-3} and so on (for the fractional part).

The value or magnitude of a given decimal number can be expressed as the sum of the various digits multiplied by their place values or weights. As an illustration, in the case of the decimal number 3586.265, the integer part (i.e. 3586) can be expressed as

$3586 = 6 \times 10^0 + 8 \times 10^1 + 5 \times 10^2 + 3 \times 10^3 = 6 + 80 + 500 + 3000 = 3586$

and the fractional part can be expressed as

$265 = 2 \times 10^{-1} + 6 \times 10^{-2} + 5 \times 10^{-3} = 0.2 + 0.06 + 0.005 = 0\ 265$

5.1.2 Binary Number System

The binary number system is a radix-2 number system with '0' and '1' as the two independent digits. All larger binary numbers are represented in terms of '0' and '1'. The procedure for writing higher-order binary numbers after '1' is similar to the one explained in the case of the decimal number system. For example, the first 16 numbers in the binary number system would be 0, 1, 10, 11, 100, 101, 110, 111, 1000, 1001, 1010, 1011, 1100, 1101, 1110 and 1111. The next number after 1111 is 10000, which is the lowest binary number with five digits. This also proves the point made earlier that a maximum of only 16 (= 2^4 numbers could be written with four digits. Starting from the binary point, the place values of different digits in a mixed binary number are 2^0, 2^1, 2^2 and so on (for the integer part) and 2^{-1}, 2^{-2}, 2^{-3} and so on (for the fractional part).

5.1.3 Octal Number System

The octal number system has a radix of 8 and therefore has eight distinct digits. All higher-order numbers are expressed as a combination of these on the same pattern as the one followed in the case of the binary and decimal number systems described in Sections 1.3 and 1.4. The independent digits are 0, 1, 2, 3, 4, 5, 6 and 7. The next 10 numbers that follow '7', for example, would be 10, 11, 12, 13, 14, 15, 16, 17, 20 and 21. In fact, if we omit all the numbers containing the digits 8 or 9, or both, from the decimal number system, we end up with an octal number system. The place values for the different digits in the octal number system are 8^0, 8^1, 8^2 and so on (for the integer part) and 8^{-1}, 8^{-2}, 8^{-3} and so on (for the fractional part).

5.1.4 Hexadecimal Number System

The hexadecimal number system is a radix-16 number system and its 16 basic digits are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F. The place values or weights of different digits in a mixed hexadecimal number are 16^0, 16^1, 16^2 and so on (for the integer part) and 16^{-1}, 16^{-2}, 16^{-3} and so on (for the fractional part). The decimal equivalent of A, B, C, D, E and F are 10, 11, 12, 13, 14 and 15 respectively, for obvious reasons.

The hexadecimal number system provides a condensed way of representing large binary numbers stored and processed inside the computer. One such example is in representing addresses of different memory locations. Let us assume that a machine has 64K of memory. Such a memory has 64K (= 2^{16} = 65,536) memory locations and needs 65,536 different addresses. These addresses can be designated as 0 to 65,535 in the decimal number system and 00000000 00000000 to 11111111 11111111 in the binary number system. The decimal number system is not used in computers and the binary notation here appears too cumbersome and inconvenient to handle. In the hexadecimal number system, 65,536 different

addresses can be expressed with four digits from 0000 to FFFF. Similarly, the contents of the memory when represented in hexadecimal form are very convenient to handle.

Examples: Conversion of Number System

Q) Convert the binary number $(1001.0101)_2$ to its decimal equivalent.

Solution:

The integer part = 1001

The decimal equivalent = $1 \times 2^0 + 0 \times 2^1 + 0 \times 2^2 + 1 \times 2^3 = 1 + 0 + 0 + 8 = 9$

The fractional part = .0101

Therefore, the decimal equivalent = $0 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 2^{-4} = 0 + 0.25 + 0$

$0.0625 = 0.3125$

Therefore, the decimal equivalent of $(1001.0101)_2 = 9.3125$

Q) Convert the octal number $(137.21)_8$ to its decimal equivalent.

Solution:

The integer part = 137

The decimal equivalent = $7 \times 8^0 + 3 \times 8^1 + 1 \times 8^2 = 7 + 24 + 64 = 95$

The fractional part = .21

The decimal equivalent = $2 \times 8^{-1} + 1 \times 8^{-2} = 0.265$

Therefore, the decimal equivalent of $(137.21)_8 = (95.265)_{10}$

Q) Convert the hexadecimal number $(1E0.2A)_{16}$ to its decimal equivalent.

Solution:

The integer part = 1E0

The decimal equivalent = $0 \times 16^0 + 14 \times 16^1 + 1 \times 16^2 = 0 + 224 + 256 = 480$

The fractional part = 2A

The decimal equivalent = $2 \times 16^{-1} + 10 \times 16^{-2} = 0.164$

Therefore, the decimal equivalent of $(1E0.2A)_{16} = (480.164)_{10}$

Q) Convert the decimal number $(13.375)_{10}$ to its binary equivalent.

Solution:

The integer part = 13

Divisor	Dividend	Remainder
2	13	—
2	6	1
2	3	0
2	1	1
—	0	1

The binary equivalent of $(13)_{10}$ is therefore $(1101)_2$

The fractional part = .375

$0.375 \times 2 = 0.75$ with a carry of 0

$0.75 \times 2 = 0.5$ with a carry of 1

$0.5 \times 2 = 0$ with a carry of 1

The binary equivalent of $(0.375)_{10} = (.011)_2$

Therefore, the binary equivalent of $(13.375)_{10} = (1101.011)_2$

Q) Convert the decimal number of $(73.75)_{10}$ to its octal equivalent.

Solution:

The integer part = 73

Divisor	Dividend	Remainder
8	73	—
8	9	1
8	1	1
—	0	1

The octal equivalent of $(73)_{10} = (111)_8$

The fractional part = 0.75

$0.75 \times 8 = 0$ with a carry of 6

The octal equivalent of $(0.75)_{10} = (.6)_8$

Therefore, the octal equivalent of $(73.75)_{10} = (111.6)_8$

Q) Convert the decimal number $(82.25)_{10}$ to its hexadecimal equivalent.

Solution:

The integer part = 82

Divisor	Dividend	Remainder
16	82	—
16	5	2
—	0	5

The hexadecimal equivalent of $(82)_{10} = (52)_{16}$

The fractional part = 0.25

$0.25 \times 16 = 0$ with a carry of 4

Therefore, the hexadecimal equivalent of $(82.25)_{10} = (52.4)_{16}$

Q) Convert the octal number $(374.26)_8$ to its binary equivalent.

Solution:

The given octal number = $(374.26)_8$

The binary equivalent = $(011\ 111\ 100.010\ 110)_2$= $(011111100.010110)_2$

Note: Any 0s on the extreme left of the integer part and extreme right of the fractional part of the equivalent binary number should be omitted.

Therefore, $(011111100.010110)_2$= $(11111100.01011)_2$

Q) Convert the binary number $(1110100.0100111)_2$ to its octal equivalent.

Solution:

The given binary number = $(1110100.0100111)_2$

$= (1\ 110\ 100.010\ 011\ 1)_2$

$= (001\ 110\ 100.010\ 011\ 100)_2$

Hence, $(001\ 110\ 100.010\ 011\ 100)_2 = (164.234)_8$

Q) Convert the hexadecimal number $(2F.C4)_{16}$ to its octal equivalent.

Solution:

The given hex number = $(2F.C4)_{16}$.

The binary equivalent = $(0010\ 1111.1100\ 0100)_2$

$= (00101111.11000100)_2$

$= (101111.110001)_2$

And finally, $(101\ 111.110\ 001)_2 = (57.61)_8$

Q) Convert the octal number $(762.013)_8$ to its hexadecimal equivalent.

Solution:

The given octal number = $(762.013)_8$.

The octal number = $(762.013)_8 = (111\ 110\ 010.000\ 001\ 011)_2$

$= (111110010.000001011)_2$

Hence, $(0001\ 1111\ 0010.0000\ 0101\ 1000)_2 = (1F2.058)_{16}$

Q) Convert the hexadecimal number $(17E.F6)_{16}$ to its binary equivalent.

Solution:

The given hex number = $(17E.F6)_{16}$

The binary equivalent = $(0001\ 0111\ 1110.1111\ 0110)_2$

$= (000101111110.11110110)_2$

$= (101111110.1111011)_2$

Here, the 0s on the extreme left of the integer part and on the extreme right of the fractional part have been omitted.

Hence, $(17E.F6)_{16} = (101111110.1111011)_2$

Q) Convert the binary number $(1011001110.011011101)_2$ to its hexadecimal equivalent.

Solution:

The given binary number = $(1011001110.011011101)_2$

$= (10\ 1100\ 1110.0110\ 1110\ 1)_2$

$= (0010\ 1100\ 1110.0110\ 1110\ 1000)_2$

Hence, $(0010\ 1100\ 1110.0110\ 1110\ 1000)_2 = (2CE.6E8)_{16}$

5.2 Binary Addition

Computer circuits don't process decimal numbers; they process binary numbers. Binary addition is the key to binary subtraction, multiplication and division.

Rules for Binary Addition:

0+0=0

0+1=1

1+0=1

1+1=10 (Here 1+1=10 but we write 1+1=0 and 1 as carry)

Eg.

```
  1010          0101   0111
+ 1101        + 0011   0101
-------       -------------
 10111          1000   1100
```

5.3 Binary Subtraction

Rules for Binary Subtraction:

$0 - 0 = 0$

$1 - 1 = 0$

$1 - 0 = 1$

$0 - 1 = 1$ (here, 1 is taken as borrow)

In case of the larger binary numbers, subtract column by column, same as we do with decimal numbers and sometimes we have to borrow from the next higher column.

Eg.

```
  1110          1100   0000
- 0101        - 0111   1101
-------       -------------
  1001          0100   0011
```

5.4 Unsigned Binary Number

The range of 8-bit number is from 0000 0000 to 1111 1111 . The decimal equivalent of this range are from 0 to 255. Similarly, the decimal equivalent of 16-bit numbers are from 0 to 65,535.

Data of the foregoing type is called unsigned binary. Because all of the bits in the binary number are used to represent the magnitude of corresponding decimal number. When the sum exceeds the range of the number system then overflow occurs.

Eg.

```
              1010    1111
            + 0111    0110
            -------------
overflow→  1 0010    0101
```

Here, 8-bit answer is wrong. If we take the overflow into account, the answer is valid but violate 8-bit arithmetic.

5.5 Sign-Magnitude Numbers

When the data has positive and negative values, some complication arises. One way to code these as binary numbers is to convert the magnitude to its binary equivalent and prefix the sign. *The MSB is 0 for the +ve number and 1 for –ve number.* Therefore -001, -010 and -011 are coded as 1001, 1010 and 1011.

Some examples of converting sign-magnitude numbers:

+7 → 0000 0111

-16 → 1001 0000

+25 → 0000 0000 0001 1001

-128 → 1000 0000 1000 0000

5.6 2's Complement Representation

a) **1's complement**

The 1's complement of the binary number is the number that results when we complement each bit.

For instance, if the input is

$X_3\ X_2\ X_1\ X_0 = 1\ 0\ 0\ 0$

Then 1's complement is,

$X'_3\ X'_2\ X'_1\ X'_0 = 0\ 1\ 1\ 1$

Eg.

Number		1's complement
1010	→	0101
1110 1100	→	0001 0011

b) **2's complement**

The 2's complement is the binary number that results when we add 1 to 1's complement.

i.e. 2's complement = 1's complement + 1

For instance,

1011 → 0100 (1's complement)

0100 + 1 = 0101 (2's complement)

Eg.

Number	1's complement	2's complement
1101	→ 0010	→ 0011
1110 1100	→ 0001 0011	→ 0001 0100

5.7 2's Complement Arithmetic

Here, the negative numbers are denoted by 2's complement.
Eg.

-6 → 1010 (6 → 0110 ; 1's complement → 1001 ; -6 → 2's complement → 1010)

Similarly,

-7 → 1001

-1 → 1111 etc.

a) Addition

i) Both Positive

→ Addition of +83 and +16

+83 → 0101 0011

+16 → 0001 0000

So,

```
+83     0101 0011
+16    +0001 0000
 99     0110 0011
```

ii) Positive and smaller negative

→ Addition of +125 and -68

+125 → 0111 1101

-68 → 1011 1100 (2's complement of 68)

So,

```
+125      0111 1101
+(-68)   +1011 1100
 57   10011 1001 → 0011 1001
```

iii) Positive and larger negative

→Addition of +37 and -115

+37 → 0010 0101

-115 → 1000 1101

So,

```
 +37      0010 0101
+(-115)  +1000 1101
 -78  →   1011 0010
```


iv) Both negative

→Addition of -43 and -78

-43 → 1101 0101

-78 → 1011 0010

So,

```
 -43      1101 0101
+(-78)   +1011 0010
-121 → 1 1000 0111 → 1000 0111
```

Note:
We ignore the final carry because it is meaningless in 8-bit arithmetic.

b) Subtraction

The format for the subtraction is:

```
  Minuend
- Subtrahend
  Difference
```

i) Both Positive

→ Subtraction of +16 from +83

+83 → 0101 0011

+16 → 0001 0000

& -16 → 1111 0000 (2's complement)

So,

```
 +83     0101 0011
+(-16)  +1111 0000
 67   10100 0011 → 0100 0011
```

ii) Positive and smaller negative

→ Subtraction of -27 from +68

+68 → 0100 0100

-27 → 1110 0101

& +27 → 0001 1011

So,

```
 +68     0100 0100
 +27    +0001 1011
 95      0101 1111
```

iii) Positive and larger negative

→Subtraction of -108 from +14

+14 → 0000 1110
-108 → 1001 0100
& +108 → 0110 1100

So,

```
  +14      0000 1110
 +108    + 0110 1100
 ------------------
  122  → 0111 1010
```

iv) Both negative

→Subtraction of -78 from -43

-43 → 1101 0101
-78 → 1011 0010
+78 → 0100 1110

So,

```
  -43      1101 0101
 +78    + 0100 1110
 ------------------
  35 → 10010 0011 → 0010 0011
```

5.8 Arithmetic Building Blocks

The building blocks are the half adder, the full-adder, and the controller inverter. Once you understand how these work; it is only a short step to see how it all comes together, that is, how a computer is able to add and subtract binary numbers of any length.

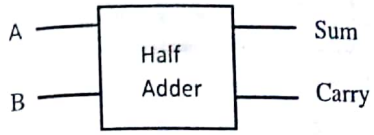
5.8.1 Half Adder

A combinational circuit that performs the addition of two bits is called a half adder.

A half adder has two bit inputs and two bit output as one bit sum and one bit carry.

Let the input variables be designed as A & B and output functions are sum and carry.

- *Block Diagram:*



- *Truth Table:*

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

From the truth table, it can be seen that the output *Sum* and *Carry* functions are similar to *Ex-OR* and *AND* functions respectively.

- *Boolean Expression:*

  Sum = A B' + A' B = A ⊕ B

  Carry = A.B

- *Logic diagram:*

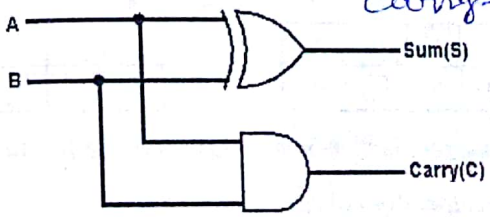


Fig: Half adder circuit

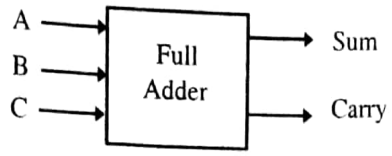
5.8.2 Full Adder

A full adder circuit adds two bits and a previous carry & produces the outputs sum and carry.

So a full adder contains three input circuit and produces a sum result as well as carry. In circuit development two half adders can be employed to form a full-adder.

Let the input variable be designed as B and C & previous carry as A, and outputs are *sum* and *carry*.

- *Block Diagram:*



- *Truth Table:*

A	B	C	Sum	Carry
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

To derive the simplified Boolean expression from the truth table, the k-map method is adopted.

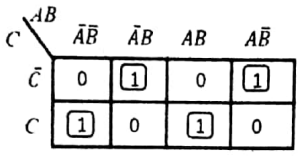


***Fig:** k-map for function Sum*

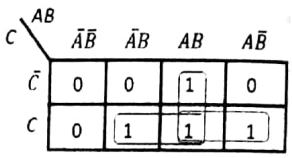


***Fig:** k-map for function Carry*

- *Boolean Expression:*

Sum $= \bar{C}\bar{A}B + \bar{C}A\bar{B} + C\bar{A}\bar{B} + CAB$

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

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

$= C \oplus A \oplus B$

$= A \oplus B \oplus C$

Carry $= AB + BC + AC$

Sum $= \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C} + ABC$

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

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

$= 1$

- *Logic diagram:*

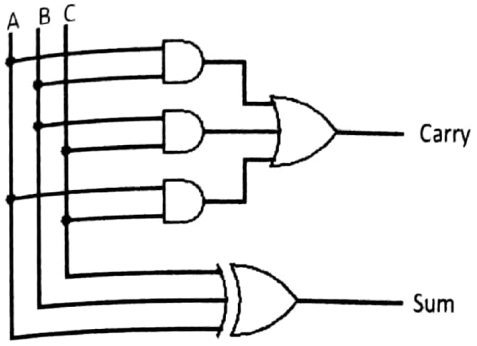


***Fig:** Circuit diagram*

Full adder using two half adder

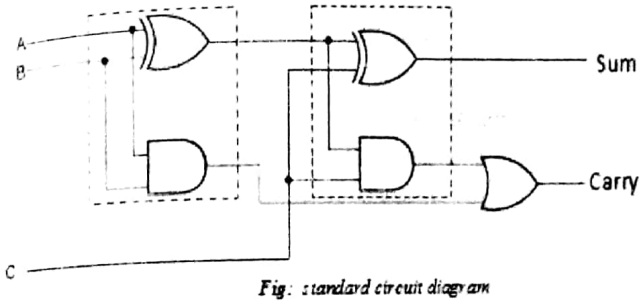


***Fig:** standard circuit diagram*

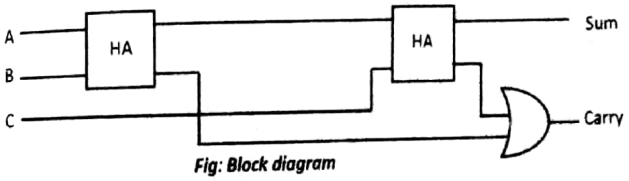
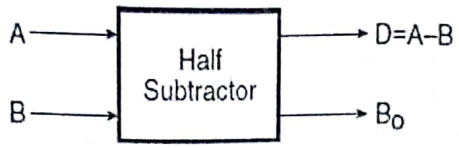


Fig: Block diagram

5.8.3 Half Subtractor

A *half subtractor* is a combinational circuit that can be used to subtract one binary digit from another to produce a DIFFERENCE output and a BORROW output. The BORROW output here specifies whether a '1' has been borrowed to perform the subtraction.

- *Block Diagram:*



- *Truth Table:*

A	B	D	B_o
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

From the truth table it can be seen that the output of difference(D) and borrow(B_0) functions are similar to Ex-or gate and AND gate with input A complemented before it is fed to the gate.

- *Boolean Expression:*

  Difference (D) = A B' + A' B = A ⊕ B

  Borrow (B_0) = A'B

- *Logic Diagram:*

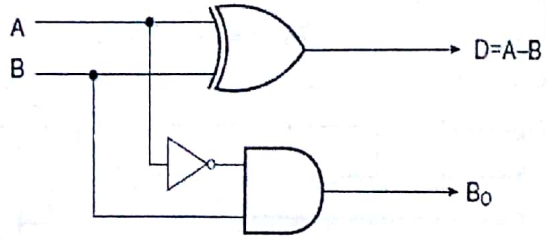
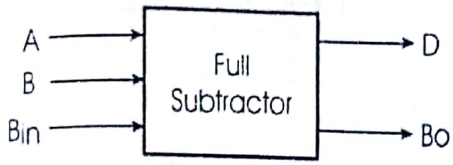


Fig: Half Subtractor circuit

5.8.4 Full Subtractor

A full subtractor performs subtraction operation on two bits, a minuend and a subtrahend, and also takes into consideration whether a '1' has already been borrowed by the previous adjacent lower minuend bit or not. As a result, there are three bits to be handled at the input of a full subtractor, namely the two bits to be subtracted and a borrow bit designated as B_{in}. There are two outputs, namely the DIFFERENCE output D and the BORROW output B_o. The BORROW output bit tells whether the minuend bit needs to borrow a '1' from the next possible higher minuend bit.

- *Block Diagram:*



- *Truth Table:*

Minuend (A)	Subtrahend (B)	Borrow In (B_{in})	Difference (D)	Borrow Out (B_0)
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

To derive the simplified Boolean expression from the truth table, the k-map method is adopted.

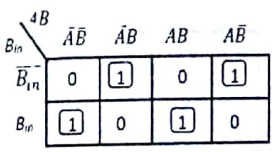


Fig: k-map for function Difference

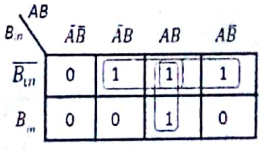


Fig: k-map for function Borrow

- *Boolean Expression:*

$$\text{Difference(D)} = \overline{B_{in}}\,\bar{A}\,\bar{B} + \overline{B_{in}}A\bar{B} + B_{in}\bar{A}\,\bar{B} + B_{in}AB$$
$$= \overline{B_{in}}(\bar{A}\,\bar{B} + A\bar{B}) + B_{in}(\bar{A}\,\bar{B} + AB)$$
$$= \overline{B_{in}}(A \oplus B) + B_{in}\overline{(A \oplus B)}$$
$$= B_{in} \oplus A \oplus B$$
$$= A \oplus B \oplus B_{in}$$
$$\text{Borrow}(B_0) = AB + B\overline{B_{in}} + A\overline{B_{in}}$$

- *Logic Diagram:*

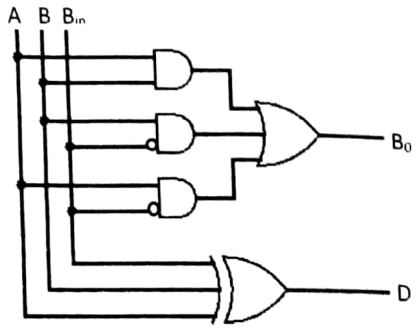


Fig: Full subtractor circuit

Full subtractor using two half subtractor

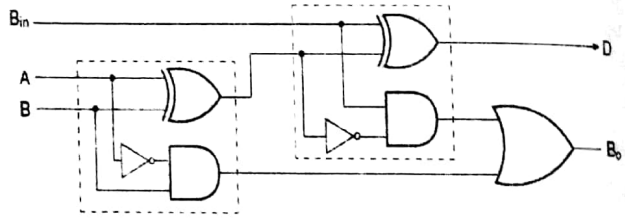


Fig: Standard circuit diagram

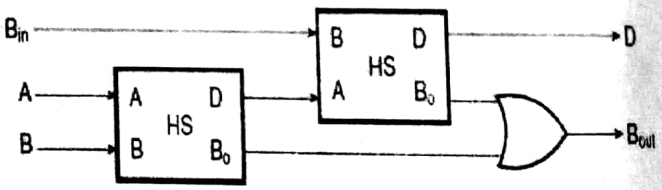


Fig: Block diagram

Q) Implement outputs of a full adder circuit by using 8:1 multiplexer. [IOE 2064 Jestha]

For full adder circuit, let we take A, B & C are inputs with sum and carry.

A	B	C	Sum	Carry
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

$Sum(x,y,z) = \Sigma(1,2,4,7)$ $Carry(x,y,z) = \Sigma(3,5,6,7)$

Implementing it using 8:1 multiplexer.

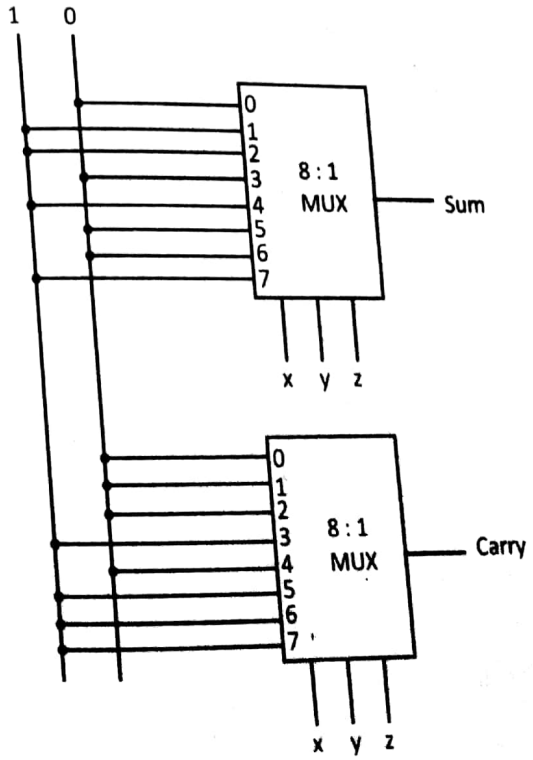


Fig: Implementation of output of full adder using 8:1 MUX

5.8.5 Controlled Inverter

A controlled inverter is needed when an adder is to be used as a subtractor. We know, the subtraction is nothing but addition of the 2's complement of the subtrahend to the minuend. Thus, the first step towards practical implementation of a subtractor is to determine the 2's complement of the subtrahend. And for this, one needs firstly to find 1's complement. A controlled inverter is used to find 1's complement. A one-bit controlled inverter is nothing but a two-input EX-OR gate with one of its inputs treated as a control input, as shown in Figure (a). When the control input is LOW, the input bit is passed as such to the output. (Recall the truth table of an EX-OR gate.) When the control input is HIGH, the input bit gets complemented at the output. Figure (b) shows an eight-bit controlled inverter of this type. When the control input is LOW, the output ($Y_7\ Y_6\ Y_5\ Y_4\ Y_3\ Y_2\ Y_1\ Y_0$) is the same as the input ($A_6\ A_5\ A_4\ A_3\ A_2\ A_1\ A_0$). When the control input is HIGH, the output is 1's complement of the input. As an example, 11010010 at the input would produce 00101101 at the output when the control input is in a logic '1' state.

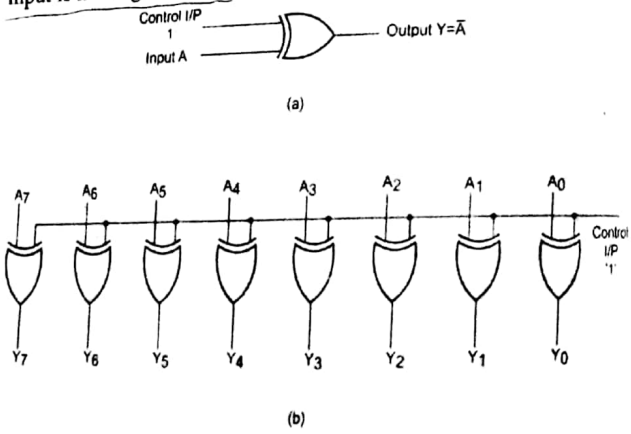


Fig: (a) One-bit controlled inverter and (b) Eight-bit controlled inverter

5.9 Binary Adder-Subtractor

A binary adder-subtractor is a combinational circuit that performs the arithmetic operations of addition and subtraction with binary numbers. We will develop this circuit by means of a hierarchical design. The half adder design is carried out first, from which we develop the adder. Connecting n full adders in cascade produces a binary adder for two n-bit numbers. The subtraction circuit is included by providing a complementing circuit.

5.9.1 Binary Adder

A binary adder is a digital circuit that produces the arithmetic sum of two binary numbers. It can be constructed with full adder connected in cascade, the output carry from each full adder connected to the input carry of the next full adder in the chain. Fig below shows the interconnection of four full adder (FA) circuits to provide a 4-bit binary ripple carry adder. The 4-bit adder is also called a ***nibble adder*** (4-bit=1 nibble). The augend bits of A and addend bits of B are designated by subscript numbers from right to left, with subscript 0 denoting the least significant bit. The carries are connected in the chain through the full adders. The input carry to the adder is C_0 and it ripples through the full adder to the output carry C_4.The S output generate the required sum bits. An n-bit adder requires n full adders with each output connected to the input carry of the next higher order full adder.

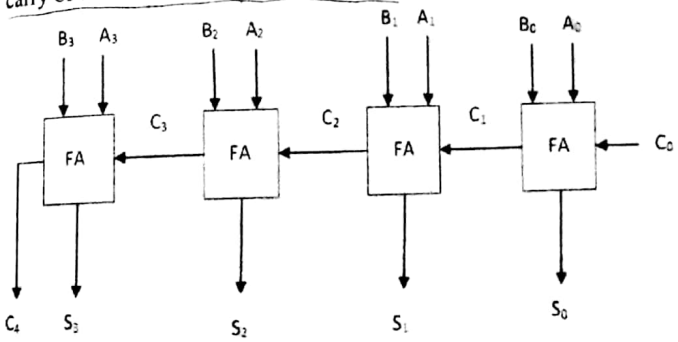


Fig: 4-bit Adder

The bits are added with full adders, starting from the position to form the sum bit and carry. The input carry C_0 in the least significant position must be 0. The value of C_{i+1} in a given significant position is the output carry of the full adder. This value is transferred into the input carry of the full adder that adds the bits one higher significant position to the left. The sum bits are thus generated starting from the rightmost position and are available for the correct sum bits to appear at the outputs.

The 4-bit adder is a typical example of a standard component. It can be used in many applications involving arithmetic operations. Observe that the design of this circuit by the classical method would require a truth table with $2^9 = 512$ entries, since there are nine inputs to the circuit. By using an iterative method of cascading a standard function, it is possible to obtain a simple and straightforward implementation.

5.9.2 Binary Subtractor

The subtraction of unsigned binary numbers can be done most conveniently by means of complement. Subtraction A–B can be done by tacking the 2's complement of B and adding it to A. The 2's complement can be obtained by taking the 1's complement and adding one to the least significant pair of bits. The 1's complement can be implemented with the inverters and a one can be added to the sum through the input carry.

The circuit for subtracting A–B consists of an adder with inverter placed between each data input B and the corresponding input of the full adder. The input carry C_0 must be equal to 1 when performing subtraction. The operation thus performed becomes A, plus the 1's complement of B, plus 1. This is equal to A plus 2's complement of B. For unsigned numbers this gives A–B if $A \geq B$ or the 2's complement of (B–A) if $A < B$. for signed numbers, the result is A – B, provided that there is no overflow.

The addition and subtraction operations can be combined into one circuit with one common binary adder. This is done by including an EX-OR gate with each full adder. A 4-bit adder-subtractor circuit is shown in fig below. The mode input M controls the operation. When M = 0, the circuit is an adder, and when M = 1, the circuit becomes a subtractor. Each EX-OR gate receives input M and one of the inputs of B. when M = 0, we have B (EX-OR) 0 = B. the full adder receive the value of B, the input carry is 0, and the circuit performs A plus B. when M = 1, we have B (EX-OR) 1= B' and $C_0 = 1$. The B inputs are complemented and a 1 is added through the input carry. The circuit performs the operation A plus the 2's complement of B. (The Ex-OR with output is for detecting an overflow.)

It is worth noting that binary numbers in the signed-complemented system are added and subtracted by the same basic addition and subtraction rules as unsigned numbers. Therefore, computers need only one common hardware circuit to handle both type of arithmetic. The user or programmer must interpret the results of such addition or subtraction differently. Depending on whether it is assumed that the numbers are signed or unsigned.

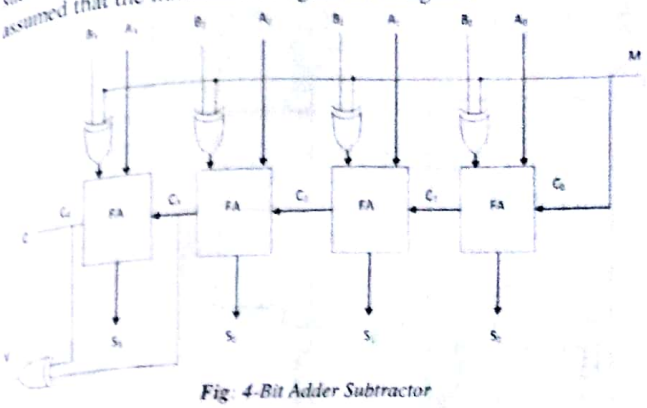


Fig. 4-Bit Adder Subtractor

5.10 Fast Adder

Fast adder is also called *parallel adder* or *carry look ahead adder* because that is how it attains high speed in addition operation.

A fast adder brings parallelism in addition process, more specifically by generating the carry using extra hardware through a look ahead logic. Cascading different arithmetic and logic unit *(devices)* is usually possible for working with larger sized numbers. Let us take 2-bit fast adder (say A and B), the carry generation equation from previous section i.e.

$$C_i = A_i B_i + (A_i + B_i) C_{i-1}$$

This can be written as,

$$C_i = G_i + P_i C_{i-1}$$

Where $G_i = A_i B_i$ →generation of carry

& $P_i = (A_i + B_i)$ →progress of carry

Now, starting from LSB,

$$C_0 = G_0 + P_0 C_{-1} \quad [C_{-1} \text{ is normally } 0]$$

$$C_1 = G_1 + P_1 C_0 = G_1 + P_1 G_0 + P_1 P_0 C_{-1}$$

After carry is available at any stage there are two more gate delays from Ex-OR gate to generate the sum bit as we can write

$S_i = G_i \oplus P_i \oplus C_i$

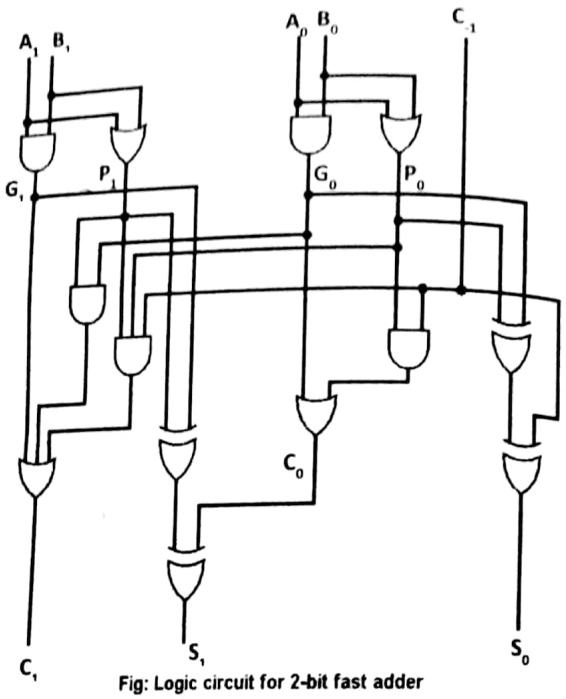


Fig: Logic circuit for 2-bit fast adder

5.11 Arithmetic Logic Unit

The arithmetic logic unit is that part of the CPU that handles all the calculations the CPU may need. Most of these operations are logical in nature. Depending on how the ALU is designed, it can make the CPU more powerful, but it also consumes more energy and creates more heat. Therefore, there must be a balance between how powerful and complex the ALU is and how expensive the whole unit becomes. This is why faster CPUs are more expensive, consume more power and dissipate more heat.

The main functions of the ALU are to do arithmetic and logic operations, including bit shifting operations. These are essential processes that need to be done on almost any data that is being processed by the CPU.

ALUs routinely perform the following operations:

- Logical Operations: These include AND, OR, NOT, XOR, NOR, NAND, etc.
- Bit-Shifting Operations: This pertains to shifting the positions of the bits by a certain number of places to the right or left, which is considered a multiplication operation.
- Arithmetic Operations: This refers to bit addition and subtraction. Although multiplication and division are sometimes used, these operations are more expensive to make. Addition can be used to substitute for multiplication and subtraction for division.

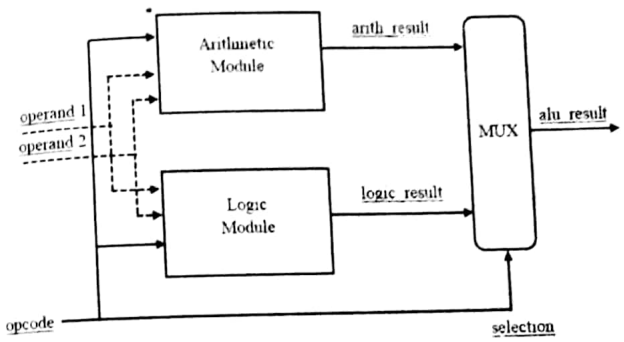


Fig: ALU Structure

5.11 Binary Multiplication and Division

5.12.1 Binary multiplication

Binary multiplication is actually much simpler than decimal multiplication. In the case of decimal multiplication, we need to remember 3 x 9 = 27, 7 x 8 = 56, and so on. In binary multiplication, we only need to remember the following,

$0 \times 0 = 0$

$0 \times 1 = 0$

$1 \times 0 = 0$

$1 \times 1 = 1$

Note that since binary operates in base 2, the multiplication rules we need to remember are those that involve 0 and 1 only. As an example of binary multiplication we have 101 times 11,

```
 101
×11
```

First we multiply 101 by 1, which produces 101. Then we put a 0 as a placeholder as we would in decimal multiplication, and multiply 101 by 1, which produces 101.

```
 101
×11
 101
1010 <-- the 0 here is the placeholder
```

The next step, as with decimal multiplication, is to add. The results from our previous step indicates that we must add 101 and 1010, the sum of which is 1111.

```
 101
×11
 101
1010
1111
```

5.12.2 Binary division

Binary division is almost as easy, and involves our knowledge of binary multiplication. Take for example the division of 1011 into 11.

```
  11  R=10
11 )1011
   -11
    101
   -11
    10  <-- remainder, R
```


To check our answer, we first multiply our divisor 11 by our quotient 11. Then we add its' product to the remainder 10, and compare it to our dividend of 1011.

```
  11
× 11
  11
 11
1001 ← product of 11 and 11

1001
+ 10
1011 <-- sum of product and remainder
```

The sum is equal to our initial dividend, therefore our solution is correct.

5.13 Complements

In digital computers actually subtraction and many other logical operations are done in terms of complements. There are two types of complements for each base/radix(r) system number.

a) r's complement
b) (r-1)'s complement

a) r's complement:

if 'N' be a given positive number of base system 'r' having 'n' number of digits(i.e. in integer part) in it then r's complement is defined as

$r^n - N$ for $N \neq 0$

0 for $N = 0$

E.g.

10's complement of $(52520)_{10}$ $= 10^5 - 54529$
$= 45471$

10's complement of $(23.324)_{10}$ $= 10^2 - 23.324$
$= 76.676$

Now if we consider a binary system, then $r = 2$.

2's complement of $(101101)_2 = (2^6)_{10} - (101101)_2$
$= (64)_{10} - (101101)_2$
$= (1000000)_2 - (101101)_2$
$= (10011)_2$

2's complement of $(0.0110)_2$ $= (2^0)_{10} - (0.0110)_2$
$= (1)_{10} - (0.0110)_2$
$= (1)_2 - (0.0110)_2$
$= (0.1010)_2$

Similarly,

Note: Complement can be found only if $r \geq 1$

b) (r-1)'s complement

For number 'N' of base 'b' having integer part of n digits and a fraction part of 'm' digits then (b-1)'s complement of 'N' is defined as

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

E.g.

9's complement of $(23450)_{10}$ $= 10^5 - 10^0 - 23450$
$= 76549$

9's complement of $(23.324)_{10}$ $= 10^2 - 10^{-3} - 23.324$
$= 76.675$

Now if we consider a binary system, then $r = 2$, *i.e.*, $(r-1)$ = 1

1's complement of $(0.1011)_2 = (1-2^{-4})_{10} - (0.1011)_2 = 0.0100$

Solved Problems

1) Perform the subtraction with the following decimal and binary number using 9's complement and 1's complement respectively.

a) 3570 – 2100 (using 9's complement)

b) 10010 – 10011 (using 1's complement)

Solution:

a) Firstly we have to know the conversion of decimal number in to 9's complement. 9's complement of decimal number can be obtained by $((10^n - 1) -$ number) where number of digits in given number.

Here

9's complement of 2100 = $((10^4 - 1) - 2100)$
$= 9999 - 2100$
$= 7899$

Adding 9's complement of 2100 to 3570

3570
+ 7899
End Around Carry → +1
1 1469
1470

So, the required value is 1470.

b) 1's complement of 10011 = 01100

10010
+ 01100
11110

So, the required value is 00001

2) Convert decimal number 73 to binary and hexadecimal and use 2's complement method to subtract 5-16. [IOE 2064]

Solution:

2	73	→ 1
2	36	→ 0
2	18	→ 0
2	9	→ 1
2	4	→ 0
2	2	→ 0
	1	

Therefore $(73)_{10} = (1001001)_2$

For hexadecimal

$(73)_{10} = (1001001)_2$
$= (0100\ 1001)_2$
$= (49)_{16}$

Now subtracting 16 from 5

Let X=5 and Y=16

$(16)_{10} = (10000)_2$

$(5)_{10} = (00101)_2$

1's complement of 16 = (01111)

Now 2's complement can be calculated as = 1's complement of 16 + 1

$= (01111)_2 + (1)_2$
$= (10000)_2$

Adding 5 and 2's complement of 16

$$\begin{array}{r} 00101 \\ +10000 \\ \hline 10101 \end{array}$$

Since there is no end carry take 2's complement of above result and introduce a ' - ' sign which gives final result :

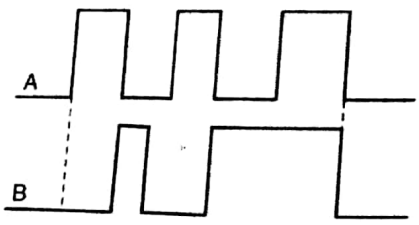
2's complement of $(10101)_2$ = -[1's complement of $(10101)_2$ + $(1)_2$]

$= -[\ 01010+1]$
$= -\ (01011)_2$

Hence $(5)_{10} - (16)_{10} = -(01011)_2$
$= -(11)_{10}$

3) For the half-adder circuit of Fig.(a), the inputs applied at A and B are as shown in Fig. (b). Plot the corresponding SUM and CARRY outputs on the same scale.

(a)



(b)

Fig (a), (b): Example

Solution

The SUM and CARRY waveforms can be plotted from our knowledge of the truth table of the half-adder. All that we need to remember to solve this problem is that 0 + 0 yields a '0' as the SUM output and a '0' as the CARRY. 0 + 1 or 1 + 0 yield '1' as the SUM output and '0' as the CARRY. 1 + 1 produces a '0' as the SUM output and a '1' as the CARRY. The output waveforms are as shown in figure below.

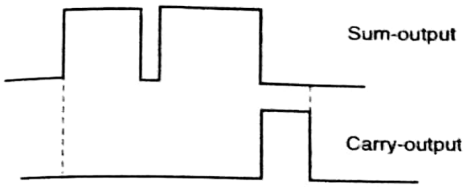
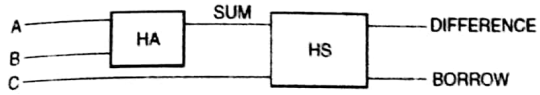


Fig: Solution to example

4) Refer to figure below, write the simplified Boolean expressions for DIFFERENCE and BORROW outputs.



Solution

Let us assume that the two inputs to the half-subtractor circuit are X and Y , with X equal to the SUM output of the half-adder and Y equal to C. DIFFERENCE and BORROW outputs can then be expressed as follows:

DIFFERENCE output= $X \oplus Y = \bar{X}.Y + X.\bar{Y}$ and BORROW output = $\bar{X}.Y$

Also, $X = \bar{A}.B + A.\bar{B}$ and Y = C.

Substituting the values of X and Y, we obtain

DIFFERENCE output =(

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

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

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

BORROW output

$= \bar{X}.Y = (\overline{\bar{A}.B + A.\bar{B}}).C = (A.B + \bar{A}.\bar{B}).C =$

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

5) Given the relevant Boolean expressions for half-adder and half-subtractor circuits, design a half-adder–subtractor circuit that can be used to perform either addition or subtraction on two one-bit numbers. The desired arithmetic operation should be selectable from a control input.

Solution

Boolean expressions for the half-adder and half-subtractor are given as follows:

Half-adder

SUM output $= \bar{A}B + A\bar{B}$ and CARRY output $= AB$

Half-subtractor

DIFFERENCE output

$= \bar{A}B + A\bar{B}$ and BORROW output $= \bar{A}B$

If we use a controlled inverter for complementing A in the case of the half-subtractor circuit, then the same hardware can also be used to add two one-bit numbers. Figure below shows the logic circuit diagram. When the control input is '0', input variable A is passed uncomplemented to the input of the NAND gate. In this case, the AND gate generates the CARRY output of the addition operation. The EX-OR gate generates the SUM output. On the other hand, when the control input is '1', the AND gate generates the BORROW output and the EX-OR gate generates the DIFFERENCE output. Thus, '0' at the control input makes it a half-adder, while '1' at the control input makes it a half-subtractor.

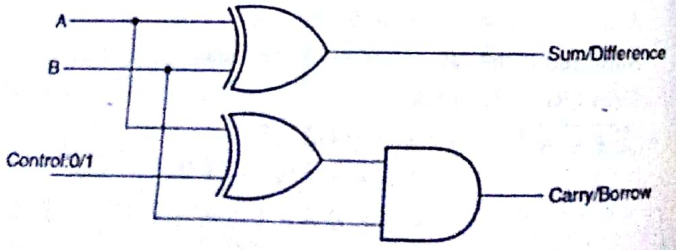


Fig: logic circuit diagram

6) Design a 3-bit substractor that operate on 2's complement numbers using full adders, and explain its operation when 2 is subtracted from 5. [IOE 2065 Shrawan]

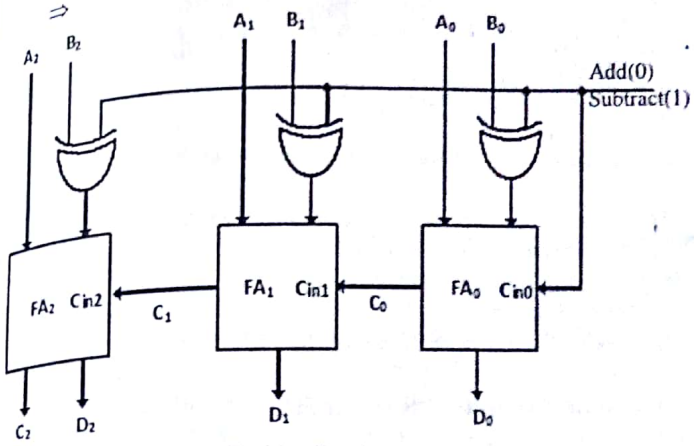


Fig: 3-bit adder subtractor

Illustration with operation 5 - 2

$5 \rightarrow 101 \Rightarrow A_2A_1A_0$

$2 \rightarrow 010 \Rightarrow B_2B_1B_0$

When 1 is fed to C_{in0} XORing with $B_2B_1B_0$ generates B_2' B_1' B_0' i.e. 1's complement of 2 is obtained. FA_0 adds $C_{in0} = 1$ to the sum of A_0 and B_0 to generate 2's complement of 2, by adding 1 to the LSB of 010. C_0 is fed to C_{in1}, to FA_1 which adds it to the sum of A_1 and B_1' and similar process occurs in FA_2. The sum is obtained as $D_2D_1D_0$, which is the real difference obtained by subtracting 2 from 5 in binary form.

Important Questions

1) What is a fast adder? Explain with examples. [IOE 2069 Ashad]
2) With the neat and clean diagram explain the operation of adder-subtractor circuit. [IOE 2071 Shrawan]
3) Convert the decimal number 168 into hexadecimal and gray code by first converting it into binary and perform the following addition using 2's complement 11+15.

[IOE 2072 Kartik]

4) Show with design that a full-adder can be implemented using two half-adders. Subtract $(16)_{10}$ from $(14)_{10}$ using 2's complement method. [IOE 2071 Chaitra]

5) Design a combinational logic that performs multiplication between two 4 bit numbers using binary parallel adder and other gates. [IOE 2069 Chaitra]

6) Implement the full adder using two half adders. [IOE 2070 Ashad]

7) Explain the working principle of binary multiplication. [IOE 2070 Ashad]

8) Describe the operation of addition process by fast adder and the importance of fast adder. [IOE 2068 Shrawan]

9) Draw the block diagram of n bit full adder and explain its operation. [IOE 2068 Chaitra]

10) Write short note on design of BCD adder. [PU 2011 Fall]

11) Write short note on Nibble adder. [PU 2013 Fall]

12) Design an adder/substractor circuit with one selection variable S and two inputs A and B. When S=0 the circuit performs A+B when S=1 the circuit performs A-B by taking the 2's complement of B. [PU 2013 Fall]

13) Compute any four of the followings as indicated.

$(11101)_{gray} = (\ )_2$

$(706)_8 = (\ )_{16}$

$(512)_{10} = (\ )_8$

$(110110.101)_2 = (\ )_{10}$

$(3FAFE.1BA)_{16} = (\ )_{10}$

[PU 2012 Spring]

14) Convert the following octal numbers to hexadecimal.

a) 1760.46

b) 6055.263

[BSC-CSIT 2066]

Chapter 6

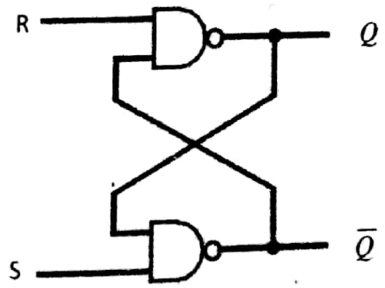
Flip Flops

The basic one bit digital memory circuit is called flip flop. It is the basic building blocks of all other types of sequential circuit whose output will remain unchanged once set, even if there is change in input level is called flip flop. It is a bistable electronic circuit that has two stable state and has a memory. It is a memory device. When the flip flop has output set as 0v dc it can be regarded as a storing a logic 0 and when its output is set as a +5v dc it is regarded as storing a logic 1. It is often called latch. Latch is a bistable device it can reside in either of two state by virtue of feedback arrangement.

6.1 R S flip flop

It is a basic flip flop. We can design other flip flop by using R S flip flop. It can be constructed using NOR gate and NAND gate. It has two input set(S) and reset(R) and two output Q and $\overline{Q}$

NAND based RS flip flop:



For NAND gate we have truth table:

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

From the above table it is clear that, single or double 0 gives output 1,

Step for NAND based RS flip flop:

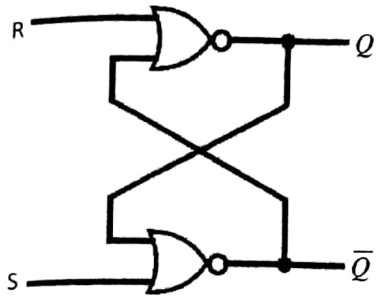
- When R=0 and S=1 then $Q=1$, $Q=1$ feeds to S so that we can obtain $\overline{Q}=0$ this is the set condition.
- When R=1 and S=0 then $\overline{Q}=1$ since from above table all 0 gives output 1, $\overline{Q}$ feeds to R so that we can obtain $Q=0$ this is reset condition.
- When both R=1 and S=1 then no change condition occurs.
- When both R=0 and S=0 then illegal condition occurs.

Now in truth table:

R	S	Q	$\overline{Q}$	comment
0	1	1	0	Set
1	1	1	0	No change
1	0	0	1	Reset
1	1	0	1	No change
0	0	-	-	illegal

NOR gated RS flip flop:

Similarly as above NAND based RS flip flop we can get NOR gated RS flip flop.



Truth table for NOR gate

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

Truth table for NOR gated RS flip flop

R	S	Q	$\overline{Q}$	comment
1	0	1	0	Set
0	0	1	0	No change
0	1	0	1	Reset
0	0	0	1	No change
1	1	-	-	illegal

6.2 Gated flip flop/clocked flip flop

In the flip flop it is possible to provide clock to store information i.e. set it or reset it at any time and then hold the stored information for any desired period of time then it is called gated flip flop or clocked flip flop.

Clocked RS flip flop

Here we can control the operation of RS flip flop by enabled clock signal. Here SR flip flop contain the two AND gate at the input of S and R terminal.

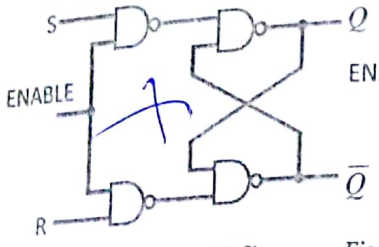


Fig: clocked SR flip flop using NAND gate

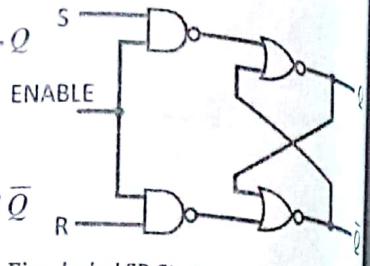


Fig: clocked SR flip flop using NOR gate

Characteristics table:

CLK	S	R	Q_{n+1}
0	×	×	Q_n (no change)
1	0	0	Q_n (no change)
1	0	1	0 (reset)
1	1	0	1 (reset)
1	1	1	Illegal

Here Q_n is previous output of Q

Q_{n+1} is present output of Q

This is activated only when clock pulse is 1 or high. Most of the time purpose of flip flop is to store 1 bit of memory and hence either Q or $\overline{Q}$ can be taken in account.

Disadvantage of clocked RS flip flop

- It requires two input signal to derive the flip flop
- When both input signal is high, we get illegal condition

Q) Draw the circuit diagram and explain the operation of edge triggered RS flip flop. *[IOE 2069 Ashad]*

Clocked D flip flop:

Here the flip flop has only one input called as delay input and two output Q and $\overline{Q}$. We can obtain the change in output only when clock pulse is high. In clocked D flip flop forbidden condition is removed. When we insert inverter in SR flip flop then we can obtained clocked D flip flop.

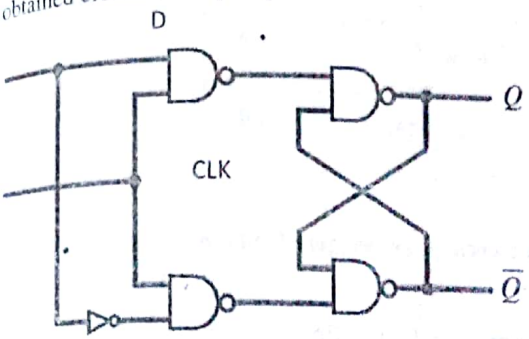


Fig: D- flip flop using NAND gate

Truth table:

CLK	Input	Output
0	×	No change
1	0	0
1	1	1

Edge triggered flip flop

Clocked flip flop are triggered by pulse. A pulse start from an initial value of 0, go instantaneously to 1, and after a short time returns to its initial 0 value. In triggering by pulse method we have to face feedback timing problem so that the concept of edge triggering or pulse transition occurs rather than the pulse duration. Hence edge triggering is those having capacity to change state either at the positive edge or negative edge of clock pulse and is sensitive to its input only at the transition of clock. It is divided in to two types:

Positive edge trigger:
It occurs when the pulse goes through signal transition from 0 to 1.

Negative edge trigger:
It occurs when the pulse goes through signal transition from 1 to 0.

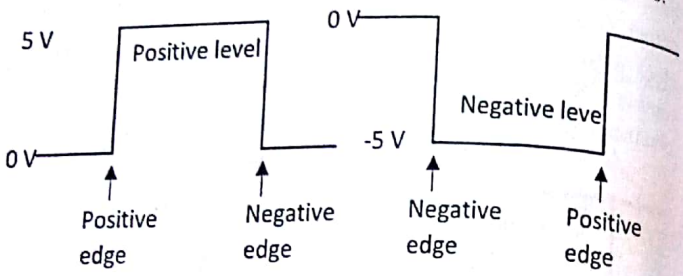


Fig: positive pulse

Fig: negative pulse

Q) Different between edge triggered and pulse triggered flip flop. *[IOE 2064 falgun]*

6.3 Edge triggered R S flip flop

Here the flip flop can't change state except on triggering edge of clock pulse. The input are transferred to the flip flop output only triggered edge of clock pulse. So R and S input of the edge triggered RS flip flop are called synchronous input.

Positive edge triggered RS flip flop

In this S and R input only effect Q when the positive trigger is high and they need to be static only during this very short time.

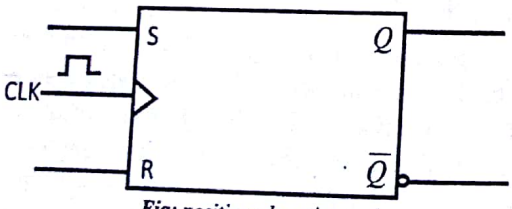


Fig: positive edge triggered

C	S	R	Q_{n+1}	Action
↑	0	0	Q_n	No change
↑	0	1	0	Reset
↑	1	0	1	Set
↑	1	1	×	Illegal

Negative edge triggered RS flip flop:
The change in output of Q is due to the input of S & R with negative trigger of a clock pulse.

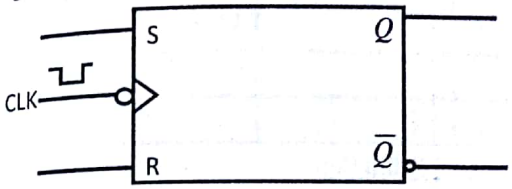


Fig: Negative edge triggered

C	S	R	Q_{n+1}	Action
↓	0	0	Q_n	No change
↓	0	1	0	Reset
↓	1	0	1	Set
↓	1	1	×	Illegal

6.4 Edge triggered D- flip flop

When an inverter is connected to S R flip flop then we can obtained D flip flops. As in the R S flip flop the flip flop can't change the state except on the triggering edge of the clock pulse.

Positive edge triggered D flip flop:

Input only effect output when positive trigger is high.

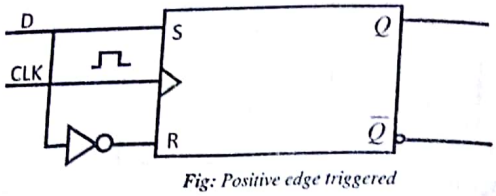


Fig: Positive edge triggered

C	D	Q_{n+1}
0	×	Q_n (last state)
↑	0	0
↑	1	1

Negative edge triggered D flip flop:

Input only effect output when negative trigger is high.

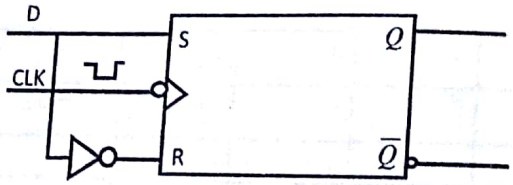


Fig: negative edge triggered

C	D	Q_{n+1}
0	×	Q_n (last state)
↓	0	0
↓	1	1

6.5 Edge triggered J K flip flop

This is the more versatile circuit since illegal condition occurs in R S flip flop is removed and we can get output. Here also output is obtained only when the positive or negative trigger is provided. In this flip flop the Q output is connected back to the input of lower AND gate and the $\overline{Q}$ output is connected back to the input of upper AND gate.

Positive edge triggered JK flip flop:

Input only effect output when positive trigger is high.

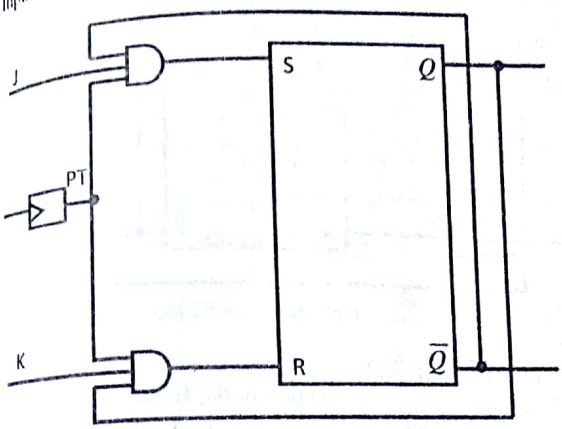


Fig: positive JK flip flop

C	J	K	Q_{n+1}	Action
↑	0	0	Q_n	No change
↑	0	1	0	Reset
↑	1	0	1	Set
↑	1	1	$\overline{Q_n}$	Toggle(complement)

2. Hold Time

Hold time is the minimum amount of time the data input should be held steady after the clock event, so that the data is reliably sampled by the clock

3. Propagation Time

Another important timing value for a flip-flop is the clock-to-output delay (t_{CO}) or propagation delay (t_P), which is the time a flip-flop takes to change its output after the clock edge. The time for a high-to-low transition (t_{PHL}) is sometimes different from the time for a low-to-high transition (t_{PLH}).

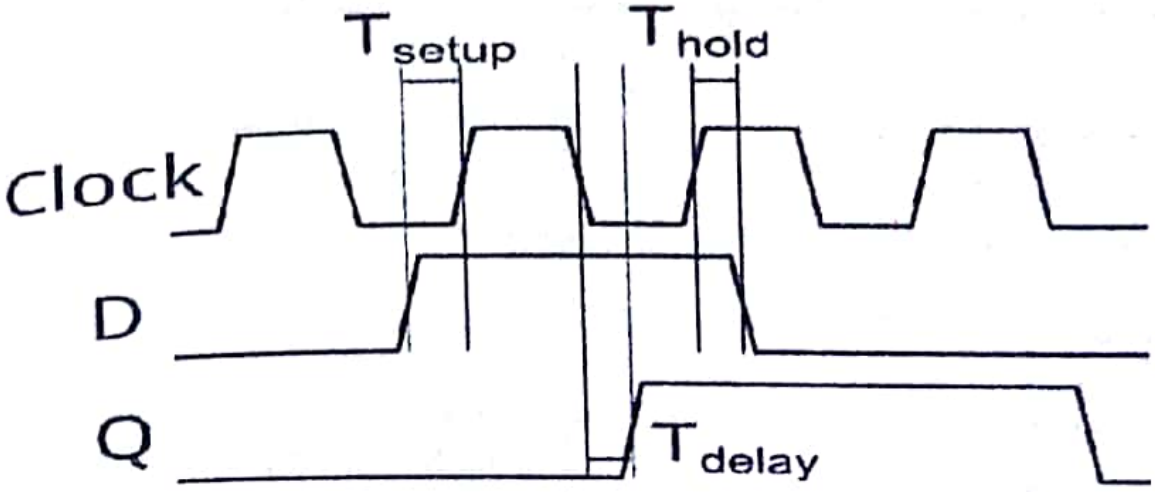


Fig: Showing different time parameters in a flip-flop

Some other parameters are: clock pulse HIGH and LOW times, asynchronous input active pulse width, clock transition time and maximum clock frequency.

6.7 JK master slave flip flop

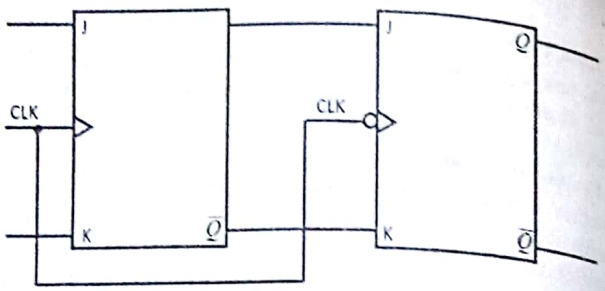


Fig: JK master slave flip flop

C	J	K	Q_{n+1}	Action
↑	0	0	Q_n	No change
↑	0	1	0	Reset
↑	1	0	1	Set
↑	1	1	$\overline{Q_n}$	Toggle (complement)

The first flip flop called master is driven by the positive edge of clock pulse and second called as slave is driven by the negative edge of clock pulse. When the clock input has a positive edge the master acts according to its JK input but the slave does not respond. When the clock end has the negative edge the slave flip flop copies the master output but the master does not respond to the feedback Q and $\overline{Q}$ since it requires the positive edge of clock pulse.

Q) Differentiate between edge triggered and pulse triggered flip flop. Explain about the Master slave flip flop.[BE IOE 2064]

6.8 Switch contact bounce circuit

Any mechanical switching device consists of a moving contact arms with a spring system. As a result when the arm is moved from one stable position to the other, arm bounce as a hard ball bounces when dropped in the hard surface. The number of bounce that occurs and period of bounce differ from each switching device.

When the switch is closed the circuit responds as if multiple signal is applied and the transition occurs in the voltage level across the switch contact. We have to eliminate the contact bounce problem using some sort of electronic circuit. Keyboard is an example of switching device.

The figure below is a single-pole-single-throw (SPST) switch. When the switch is open the voltage at point A is +5v dc. When the switch is closed, the voltage at point A is 0v dc.

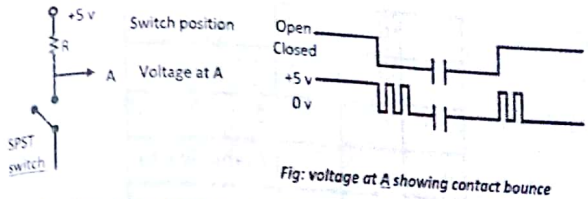


Fig: voltage at A showing contact bounce

Elimination of switch contact bounce

To eliminate it a simple RS latch de-bounce circuit is used.

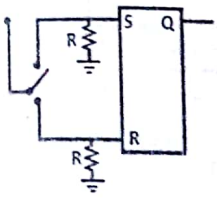


Fig: Switch contact bounce eliminator

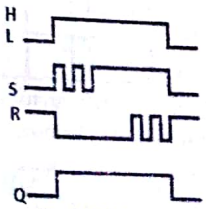


Fig: Switch bounce

Here when the switch is moved to the position H, R=0 and S=0 then the flip flop will set with Q = 1 at the first high level on S. Again when the switch bounces losing the contact to the input signal then R = S = 0, therefore the flip flop remain set (Q = 1). When the switch regain contact R = 0 and S = 1; this causes an attempt to again set the flip flop. But the flip flop is already set no change occurs at Q. The result is a "clean" low-to-high signal at its output.

When the switch is moved to position L, S = 0 & R = 1. Bouncing occurs at the R input due to switch. Here it simply respond to the first high level, and ignores all following transition. The result is a "clean" low-to-high signal at its output.

Q) *What is the contact switch bouncing? Describe how it can be eliminated.* [BE IOE 2068]

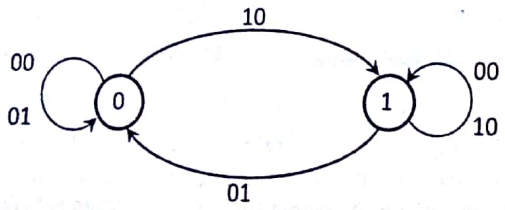
6.9 Various representation of flip flops

S-R flip flop:

The **state transition table** for S-R flip flop is as follow:

Q_n	S	R	Q_{n+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	Forbidden(x)
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	Forbidden(x)

From the above state transition table we can draw a **state transition diagram** as below:



Excitation table are very useful in design or synthesis problem excitation table of flip flop is looking at its truth table in reverse way. Here the flip fop is presented as a dependent function of transition $Q_n \rightarrow Q_{n+1}$ and come later on the table. This is derived more directly from state transition diagram.

From above state transition diagram:

If present state is 0 application of S R = 0 x does not alter its value where 'x' denotes don't care condition in R input.

State 0 to 1 transition occurs when S R = 1 0 is present at the input side while state 1 to 0 transition occurs if S R = 0 1.

If present state is 1 application of S R = x 0 does not alter its value.

Q_n	Q_{n+1}	S	R
0	0	0	x
0	1	1	0
1	0	0	1
1	1	x	0

The **characteristics equation** of S-R flip flop are useful in analyzing circuits made of them which is as follow.

Q \ SR	00	01	11	10
0	0	0	x	1
1	1	0	x	1

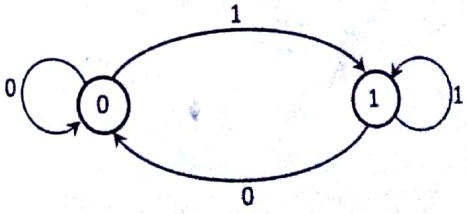
$$Q_{n+1} = S + \bar{R}Q_n$$

D – flip flop:

State transition table:

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

State transition diagram:



Excitation table:

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

Characteristics equation:

Q_n \ D	0	1
0	0	1
1	0	1

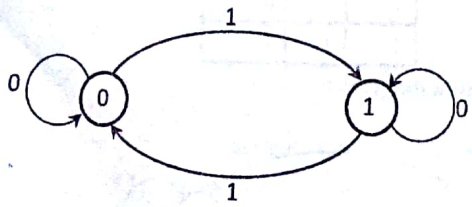
$$Q_{n+1} = D$$

T – flip flop:

State transition table:

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

State transition diagram:



Excitation table:

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

Characteristics equation:

Q_n \ T	0	1
0	0	1
1	1	0

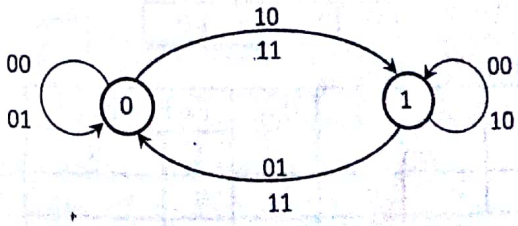
$$Q_{n+1} = T\overline{Q_n} + \overline{T}Q_n$$

J-K flip flop:

State transition table:

Q_n	J	K	Q_{n+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1 (toggle)
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0 (toggle)

State transition diagram:



Excitation table:

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

Characteristics equation:

Q_n \ JK	00	01	11	10
0	0	0	1	1
1	1	0	0	1

$$Q_{n+1} = J\overline{Q_n} + \overline{K}Q_n$$

Realization of one flip flop using another flip flop:

We can convert the available type of flip flop to another. It is important to know the conversion since we have design the circuit of one types and for implementation we get the different type from the market. When we have knowledge ibn conversion then we can change one type of flip flop in to another easily.

1) Realization of J-K flip flop using S-R flip flop:

Excitation table for S-R flip flop:

Q_n	Q_{n+1}	S	R
0	0	0	x
0	1	1	0
1	0	0	1
1	1	x	0

State synthesis table:

Q_n	J	K	Q_{n+1}	Excitation input	
				S	R
0	0	0	0	0	x
0	0	1	0	0	x
0	1	0	1	1	0
0	1	1	1	1	0
1	0	0	1	x	0
1	0	1	0	0	1
1	1	0	1	x	0
1	1	1	0	0	1

Karnaugh map for S R input are:

For S:

Q_n \ JK	00	01	11	10
0	0	0	1	1
1	x	0	0	x

$$S = J\overline{Q_n}$$

For R:

Q_n \ JK	00	01	11	10
0	x	x	0	0
1	0	1	1	0

$$R = KQ_n$$

From those two equation of S and R we can draw the below circuit

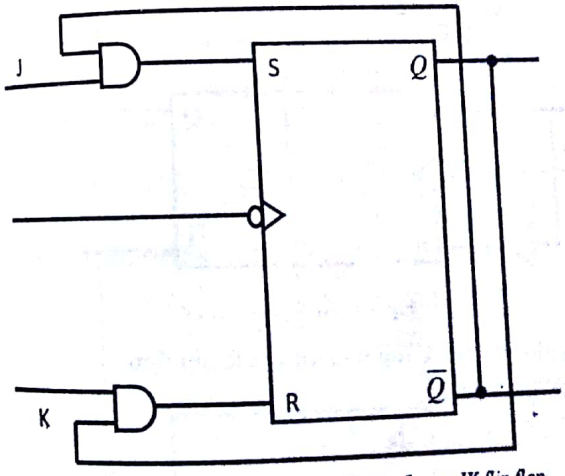


Fig: Conversion to SR flip flop to JK flip flop

2) Realization of D-flip flop using SR flip flop:

Excitation table for SR flip flop

Q_n	Q_{n+1}	S	R
0	0	0	x
0	1	1	0
1	0	0	1
1	1	x	0

State synthesis table

Q_n	D	Q_{n+1}	Excitation input	
			S	R
0	0	0	0	x
0	1	1	1	0
1	0	0	0	1
1	1	1	x	0

Karnaugh map for S R input are:

For S

Q_n \ D	0	1
0	0	1
1	0	x

S=D

For R

Q_n \ D	0	1
0	x	0
1	1	0

R=D'

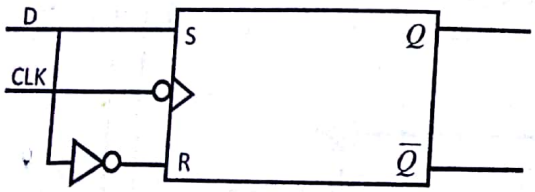


Fig: positive edge triggered

3) Realization of T-flip flop using SR flip flop:

Excitation table:

Q_n	Q_{n+1}	S	R
0	0	0	x
0	1	1	0
1	0	0	1
1	1	x	0

State synthesis table:

Q_n	T	Q_{n+1}	Excitation input	
			S	R
0	0	0	0	x
0	1	1	1	0
1	0	1	x	0
1	1	0	0	1

Karnaugh map for S and R input are:

For S

Q_n \ T	0	1
0	0	1
1	x	0

$S = \overline{Q_n}T$

For R

Q_n \ T	0	1
0	x	0
1	1	0

$R = Q_nT$

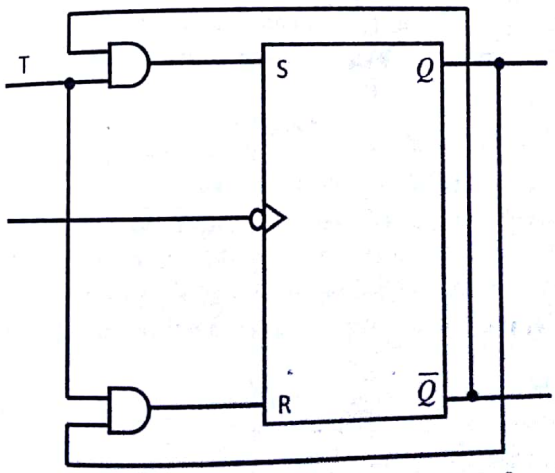


Fig: Conversion to SR flip flop to JK flip flop

Note: Similarly we can realize one flip flop from another

Q) Convert RS flip flop to JK flip flop. *[IOE 2068 Ashad]*

Important Questions

1) Draw the circuit diagram and explain the operation of edge triggered RS flip flop. Convert RS flip flop to JK flip flop. [IOE 2069 Ashad]
2) Write down the drawback of RS flip flop. Explain the operation of D flip flop using timing diagram and truth table. [IOE 2068 Chaitra]
3) Draw the circuit diagram of edge triggered JK flip flop and explain it. [IOE 2068]
4) Explain the operation of positive edge triggered with circuit diagram, truth table and excitation table.
5) Explain the operation of RS flip flop showing its logic diagram, characteristics table and then derive its characteristics equation and excitation table. [IOE 2070 Ashad]
6) What do you mean by a clocked flip-flop? With the logic diagram describe the function of RS, D, JK, T flip-flop. (Negative edge trigger) [Purbanchal 2005]
7) Give the truth table logic circuit and characteristic equation of J-K flip flop. How it can be converted into T flip flop. [PU Fall 2013]

Chapter 7

Registers

A register is a group of flip-flops that can be used to store a binary number. An n-bit register, has n flip-flops and is capable of holding n-bits of information. In addition to flip-flops a register can have a combinational part that performs data-processing tasks.

7.1 Types of Registers

The flip-flops must be connected such that the binary number can be entered and shifted into the register and possibly shifted out. *A register capable of shifting its binary contents either to the left or to the right is called a shift register.* The shift register permits the stored data to move from a particular location to some other location within the register. Registers can be designed using discrete flip-flops(S-R, J-K, and D-type).

The data in a shift register can be shifted in two possible ways: (*a*) serial shifting and (*b*) parallel shifting. The serial shifting method shifts one bit at a time for each clock pulse in a serial manner, beginning with either LSB or MSB. On the other hand, in parallel shifting operation, all the data (input or output) gets shifted simultaneously during a single clock pulse. Hence, we may say that parallel shifting operation is much faster than serial shifting operation.

There are two ways to shift data into a register (serial or parallel) and similarly two ways to shift the data out of the register. All of the four configurations are commercially available as TTL MSI/LSI circuits. They are:

1. Serial in/Serial out (SISO)

2. Serial in/Parallel out (SIPO)
3. Parallel in/Serial out (PISO)
4. Parallel in/Parallel out (PIPO)

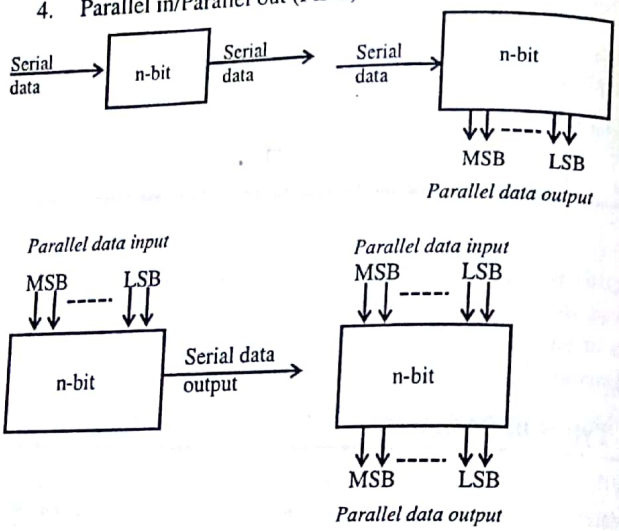


Fig: Four types of shift register

There are two ways for shifting. a) Shift-right & b) Shift-left. Here we only talk about *Shift-right* register.

7.2 Serial in/Serial out (SISO)

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.

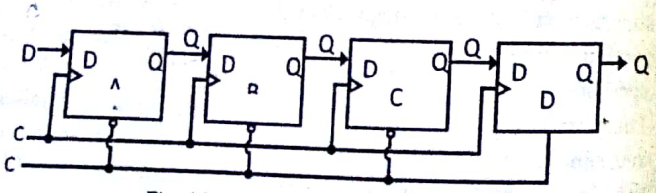


Fig: 4-bit serial in serial out circuit

The clock pulse is applied to all the flip-flops simultaneously. When the clock pulse is applied, each flip-flop is either set or reset according to the data available at that point of time at the respective inputs of the individual flip-flops. Hence the input data bit at the serial input line is entered into flip-flop A by the first clock pulse. At the same time, the data of stage A is shifted into stage B and so on to the following stages. For each clock pulse, data stored in the register is shifted to the right by one stage. New data is entered into stage A, whereas the data present in stage D are shifted out (to the right).

For example: consider that all the stages are reset and a logical input 1011 is applied at the serial input(LSB to MSB) line connected to stage A. The data after four clock pulses is shown in table.

Timing pulse	Q_A	Q_B	Q_C	Q_D	*Serial output at Q_D*
Initial value	0	0	0	0	0
After 1st clock pulse	1	0	0	0	0
After 2nd clock pulse	1	1	0	0	0
After 3rd clock pulse	0	1	1	0	0
After 4th clock pulse	1	0	1	1	1

Let us now illustrate the entry of the 4-bit number 1011 into the register, beginning with the right-most bit. A 1 is applied at the serial input line, making D = 1. As the first clock pulse is applied, flip-flop A is SET, thus storing the 1. Next, a 1 is applied to the serial input, making D = 1 for flip-flop A and D = 1 for flip-flop B also, because the input of flip-flop B is connected to the Q_A output. When the second clock pulse occurs, the 1 on the data input is "shifted" to the flip-flop A and the 1 in the flip-flop A is "shifted" to flip-flop B. The 0 in the binary number is now applied at the serial input line, and the third clock pulse is now applied. This 0 is entered in flip-flop A and the 1 stored in flip-flop A is now

"shifted" to flip-flop B and the 1 stored in flip-flop B is now "shifted" to flip-flop C. The last bit in the binary number that is the 1 is now applied at the serial input line and the fourth clock pulse is now applied. This 1 now enters the flip-flop A and the 0 stored in flip-flop A is now "shifted" to flip-flop B and the 1 stored in flip-flop B is now "shifted" to flip-flop C and the 1 stored in flip-flop C is now "shifted" to flip-flop D. Thus the entry of the 4-bit binary number in the shift-right register is now completed.

From the third column of the above table we can get the serial output of the data that is being entered in the register. We find that after the first, second, and the third clock pulses the output at the serial output line *i.e.*, Q_D is 0. After the fourth clock pulse the output at the serial output line is 1. If we want to get the total data that we have entered in the register in a serial manner from Q_D, then we have to apply another three clock pulses. After the fifth clock pulse we will gate another 1 at Q_D. After the sixth clock pulse the output at Q_D will be 0 and after the seventh clock pulse the output at Q_D will be 1. In this process of the fifth, sixth, and the seventh clock pulses if no data is being supplied at the serial input line then the A, B, and C flip-flops will again be RESET with output 0.

Timing Diagram

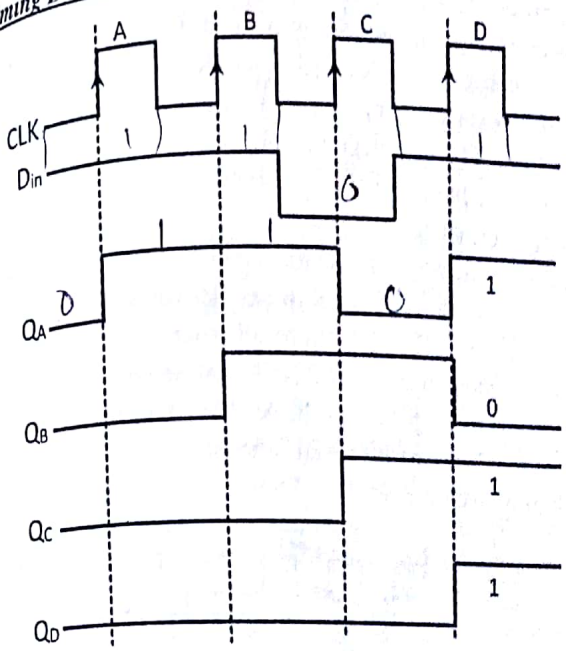


Fig: waveforms of 4-bit serial input shift register

7.3 Serial in/Parallel out (SIPO)

In this type of register, the data is shifted in serially, but shifted out in parallel. To obtain the output data in parallel, it is required that all the output bits are available at the same time. This can be accomplished by connecting the output of each flip-flop to an output pin. Once the data is stored in the flip-flop the bits are available simultaneously.

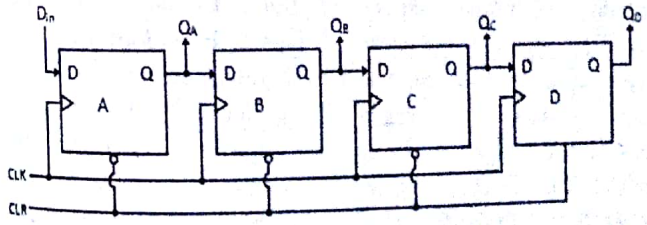


Fig: 4-bit serial in parallel out circuit

The above details of the serial-in/parallel-out shift register are fairly simple. It looks like a serial-in/serial-out shift register with taps added to each stage output. Serial data shifts in at D_{in} (Serial Input). After a number of clocks equal to the number of stages, the first data bit in appears at Q_D in the above figure. In general, there is no serial out pin. The last stage (Q_D above) serves as serial out and is cascaded to the next package if it exists.

If a serial-in/parallel-out shift register is so similar to a serial-in/serial-out shift register, why do manufacturers bother to offer both types? Why not just offer the serial-in/parallel-out shift register? They actually only offer the serial-in/parallel-out shift register, as long as it has no more than 8-bits. Note that serial-in/serial-out shift registers come in bigger than 8-bit lengths of 18 to 64-bits. It is not practical to offer a 64-bit serial-in/parallel-out shift register requiring that many output pins.

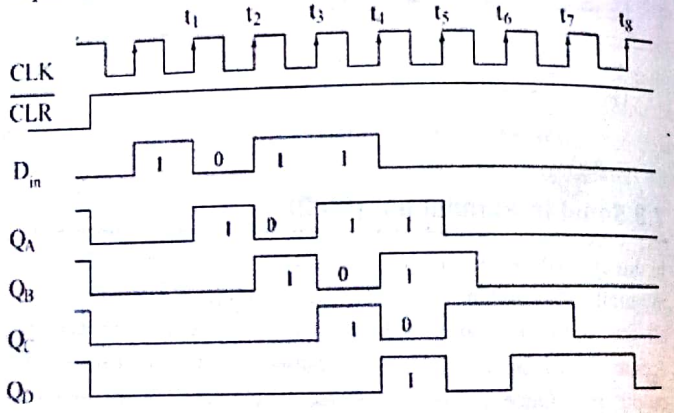


Fig: waveforms of 4-bit Serial in/Parallel out shift register

The shift register has been cleared prior to any data by *CLR'*, an active low signal, which clears all type D Flip-Flops within the shift register. Note the serial data *1011* pattern presented at the D_{in} input. This data is synchronized with the clock *CLK*. This would be the case if it is being shifted in from something like another shift register, for example, a parallel-in/serial-out shift register (not shown here). On the first clock at t_1, the data *1* at D_{in} is shifted from *D* to *Q* of the first shift register stage. After t_2 this first data bit is at Q_B. After t_3 it is at Q_C. After t_4 it is at Q_D. Four clock pulses

have shifted the first data bit all the way to the last stage Q_D. The second data bit a 0 is at Q_C after the 4th clock. The third data bit a 1 is at Q_B. The fourth data bit another 1 is at Q_A. Thus, the serial data input pattern 1011 is contained in ($Q_D Q_C Q_B Q_A$).

It is now available on the four outputs. It will available on the four outputs from just after clock t_4 to just before t_5. This parallel data must be used or stored between these two times, or it will be lost due to shifting out the Q_D stage on clocks t_5 to t_8 as shown above.

7.4 Parallel in/Serial out (PISO)

Parallel-in/ serial-out shift registers do everything that the previous serial-in/serial-out shift registers do plus input data to all stages simultaneously. The parallel-in/ serial-out shift register stores data, shifts it on a clock by clock basis, and delays it by the number of stages times the clock period. In addition, parallel-in/ serial-out really means that we can load data in parallel into all stages before any shifting ever begins. This is a way to convert data from a *parallel* format to a *serial* format. By parallel format we mean that the data bits are present simultaneously on individual wires, one for each data bit as shown below. By serial format we mean that the data bits are presented sequentially in time on a single wire or circuit as in the case of the "data out" on the diagram below.

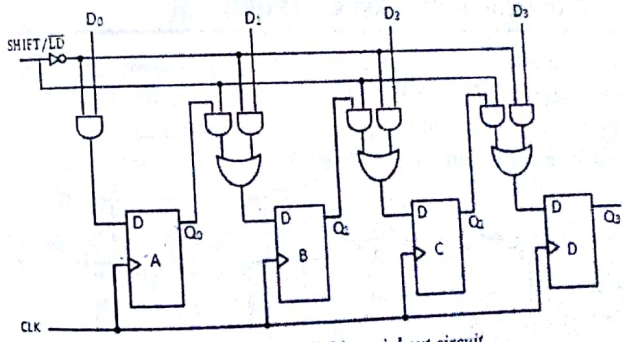


Fig: 4-bit parallel in serial out circuit

The circuit shows 4-bits PISO shift register D_0, D_1, D_2 and D_3 are parallel inputs where, D_0 is MSB and D_3 is LSB. 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.

Timing waveform for 4-bit PISO shift register when data bits $D_0D_1D_2D_3 = 0101$ is entered. Assume D input remains a 1.

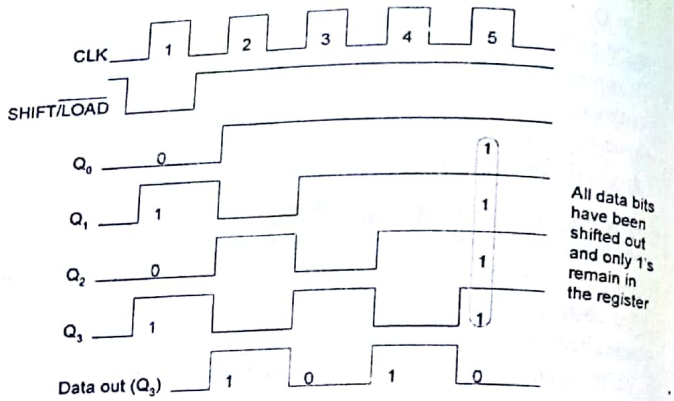


Fig: waveforms of PISO

Why provide serial in and serial out pins on a shift register? These connections allow us to cascade shift register stages to provide large shifters than available in a single IC (Integrated Circuit) package. They also allow serial connections to and from other ICs like microprocessors.

7.5 Parallel in/Parallel out (PIPO)

In parallel-in/parallel-out registers the data will appear immediately at the output after a single clock pulse and there is no clock delay. This PIPO register are also known as general register because of no shifting operation.

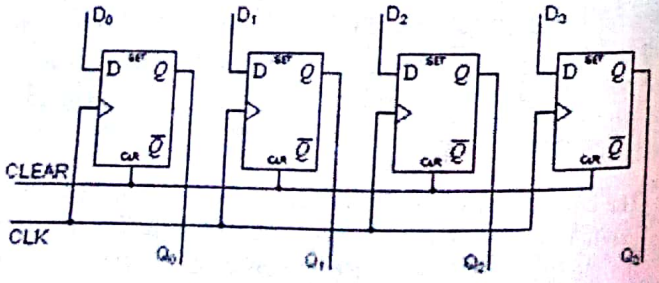


Fig: parallel in parallel out circuit

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. The above circuit is a four-bit parallel-in/parallel-out shift register constructed by D flip-flop switch D_0=1, D_1=0, D_2=1 and D_3=0.

The clock waveform can be drawn as follows where the inserted data bits are shifted out in parallel.

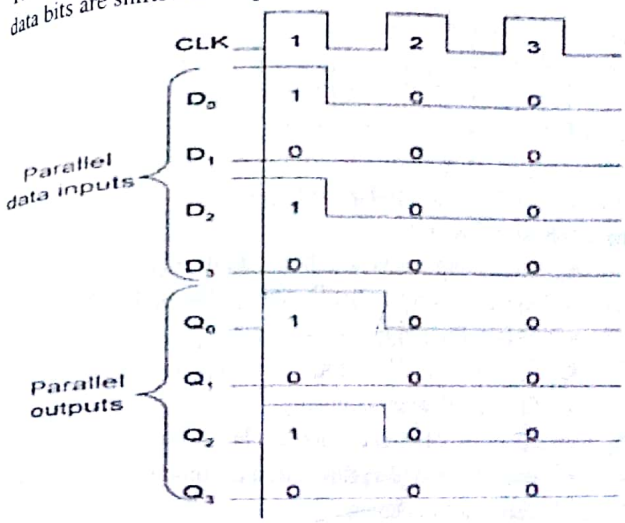


Fig: clock waveforms of PIPO

7.6 Applications of shift registers

- 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

The most common types of shift register counters are introduced here. They are basically shift registers with the serial outputs connected back to the serial inputs in order to produce particular sequences.

- Ring counter
- Johnson counter or twisted counter
- Pseudo random pattern generator

a) Ring Counter

- Ring counter is a shift register in which the output of the last flip flop is fed to the input again.
- For every clock pulse the data shifted into the next flip flop thus it can count only N states.

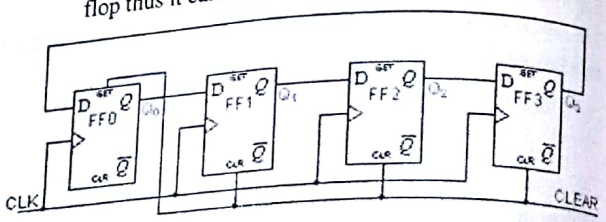


Fig: Ring counter

b) Johnson Counter

- The complement of the output of the last stage is connected back to the D input of the first stage.
- Also called the *twisted-ring counter*.
- Require fewer flip-flops than ring counters but more flip-flops than binary counters.
- An *n*-bit Johnson counter cycles through 2*n* states.
- Require more decoding circuitry than ring counter but less than binary counters.

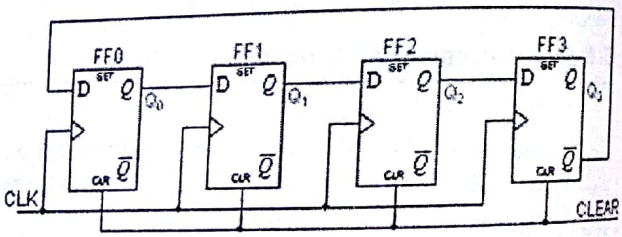


Fig: Johnson counter

c) Pseudo Random Pattern Generator

- It is a simple shift register where the vacated bit of is filled with the exclusive-or of last two bits in the shift register.
- By using this for a n bit shift register we can generate 2^n patterns.
- This is very useful in test pattern generation.

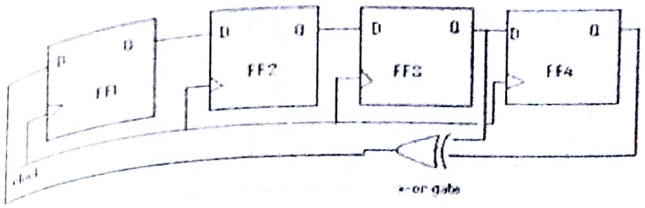


Fig: Pseudo random pattern generator

Important Questions

1) With the clear circuit and timing diagram, Explain the operation of serial in serial out shift register. [IOE 2068 Chaitra]
2) What is shift register? With clear timing diagram, describe the operation of a 4-bit parallel in serial out (PISO) shift register. [IOE 2068 Baisakh]
3) Describe briefly the operation of a 4-bit serial in parallel out register with a clear circuit diagram. [IOE 2067 Ashad]
4) Define shift registers. Explain operation of parallel in parallel out shift register. [PU 2011 Fall]
5) What are the shift register operations? [BSC-CSIT 2065]
6) What are the various type of shift registers? [BSC-CSIT 2066]

Chapter 8

Counters

A counter is essentially a register that goes through a predetermined sequence of states upon the application of input pulse. The gate in a counter are connected in such a way as to produce a prescribed sequence of binary state in a register. A counter driven by clock can be used to count the number of clock cycles. It can be used as an important for measuring time and therefore period and frequency. There are two different types of counter they are:

1) Synchronous counter
2) Asynchronous counter

8.1 Asynchronous counter

In asynchronous counter each flip flop is triggered by previous flip flop. Ripple counter is one of the example. It is also called as the serial counter. It require minimum hardware

8.1.1 Ripple counter

In ripple counter the flip flop output transition serves as a source of triggering other flip flop. In other words, the clock pulse input of all flip flop except the first are triggered not by the incoming pulse, but rather by the transition that occurs in other flip flop. A binary ripple counter can be constructed by using clocked JK flip flop.

Below figure is a three negative edge triggered JK flip flop connected in cascade.

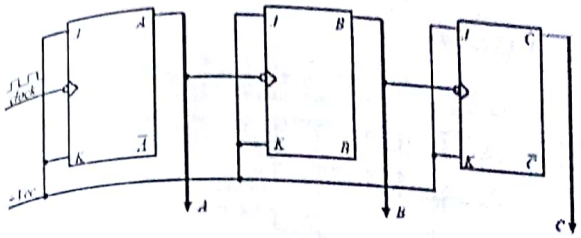


Fig: Three bit binary ripple

Here in figure, the A flip flop must change state before it trigger the B flip flop and B flip flop has to change the state before it can trigger the C flip flop. The trigger moves through the flip flop like ripple of the water. Because of this the overall propagation delay time is the sum off all individual delays.

Let us assume all flip flop are at reset to produce 0 output. If we consider A to be LSB and C to be MSB. The content of the counter initially be 000. When there is negative edge triggered then the flip flop A will change state. This is indicated by downward arrow. At the point a in the time line, A goes high, at the point b it goes back low, at the point c it again goes back high, at the point d it goes low and so on. The wave form at the output of flip flop A is one half the clock frequency.

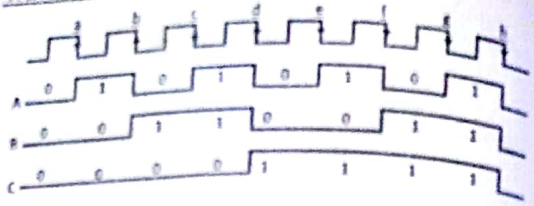
Since A acts as the clock for B, each time the wave form at A goes low, flip flop B will toggle thus at the point b in time line B goes high; it then goes low at point d and toggle back high at again at point f. The wave form at the output of flip flop B is one fourth of the clock frequency.

Since B acts as the clock for C, each time the wave form at B goes low, flip flop C will toggle thus C goes high at point d and goes low at point h. The waveform at the output of flip flop C is one eighth of the clock frequency.

Truth table:

Count	C	B	A
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

Wave form



8.1.2 Ripple down counter:

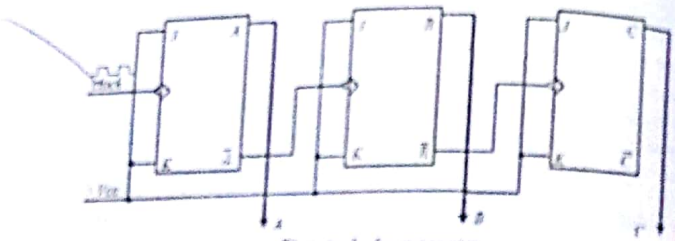


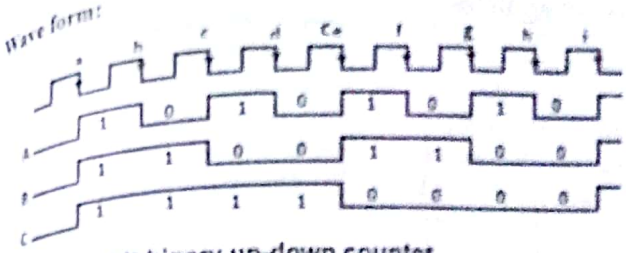
Fig: ripple down counter

In the ripple down counter the system clock is used at the clock input of flip flop A but the complement i.e. $\overline{A}$ is used to drive flip flop B likewise $\overline{B}$ is used to drive flip flop C and so on.

Truth table:

Count	C	B	A
7	1	1	1
6	1	1	0
5	1	0	1
4	1	0	0
3	0	1	1
2	0	1	0
1	0	0	1
0	0	0	0

Wave form:



8.1.3 Three bit binary up-down counter

It is the combination of two counter i.e. up counter and down counter. Here if the count-down control line is low and count-up control line is high then the counter will have count-up wave form. On the other hand, if the count-down is high and count-up is low, each flip flop will be triggered from the complement side of previous flip flop. The counter will then be in count down mode.

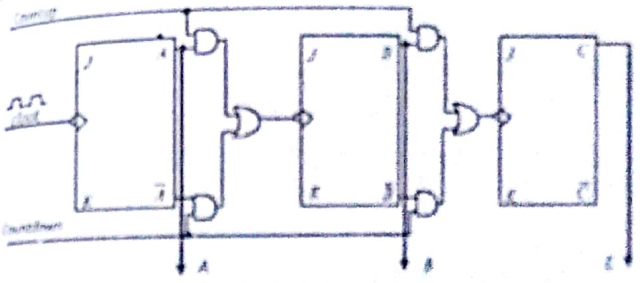


Fig: ripple up-down counter

Disadvantage of Ripple counter:

a) It has a limit to its highest operating frequency

b) Each flip flop has a delay time, in a ripple counter these delay time are additive and the total "settling" time for a counter is approximately the delay time times the total number of flip flop.

c) There is a possibility of glitches i.e. an undesired positive or negative pulse appearing at the outside of logic gate with a ripple counter.

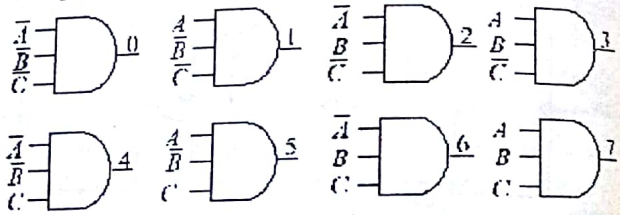
Q) ***Define ripple counter? Explain the operation of MOD-10 ripple counter with timing diagram.*** *[BE IOE2069 Chaitra]*

8.2 Decoding Gates

A decoding gates can be connected to the output of the counter in such a way that the output of the gate will be high or low only when the counter content are equal to a given state.

Suppose the decoding gates connected to the three bit ripple counter will decode 2 (CBA = 010). Thus the gate output will be high only when A=0, B=1 and C=0. The Boolean expression for this gate can be written 2=CBA. A comparison with the truth table for this counter will reveal that the condition CBA=010. Only true for the state 2.

Other seven state of the counter can be decoded in a similar manner. Suppose decode state 7, the truth table reveal that CBA=111 is a unique state. Similarly suppose decode state 5, the truth table reveal that CBA=101 is the unique state. All the decoding gates of three bit counter are below.



8.3 Synchronous Counter

In a synchronous counter pulse are applied to the input of all flip flop. The common pulse trigger all the flip flop simultaneously, rather than one at a time in a succession as in a ripple counter. The decision whether a flip flop is to be complemented or not is determined from the values of J and K inputs at the time of the pulse. If J = K = 0, the flip flop remains unchanged. If J = K = 1 the flip flop complements.

8.3.1 Three bit synchronous up counter

Up counter are those which counts upward or in the forward direction by one LSB every time it is clocked. Figure below figure is three bit synchronous up counter. Here clock pulse is passed to all of the flip flop. But the A

output of the first stage is used to derive the J and K input of second stage. B output of second stage is used to derive the J and K of third stage.

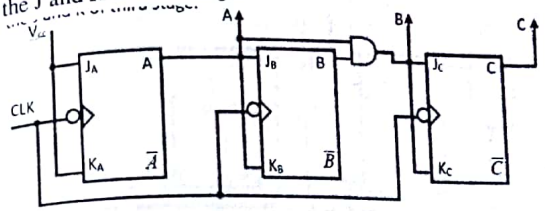
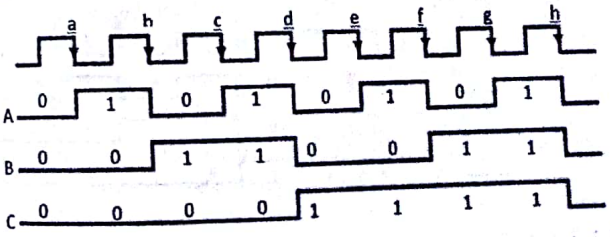


Fig: three bit synchronous up counter

Truth table:

Count	C	B	A
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

Wave form:



8.3.2 Three bit synchronous down counter

A down counter counts in a reverse direction or downward by the LSB every time it is clocked. Figure below is a three bit synchronous down counter. Here the clock pulse is passed to all of the flip flop but the complement $\bar{A}$ is used to derive the

J and K input of second stage. $\overline{B}$ output of the second stage is used to derive the J and K of the third stage.

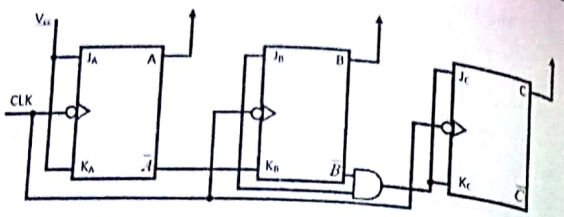
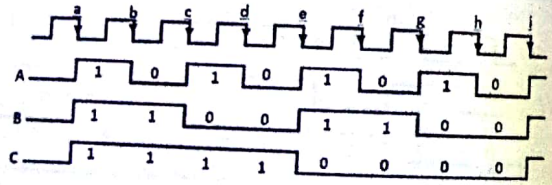


Fig: three bit synchronous down counter

Truth table:

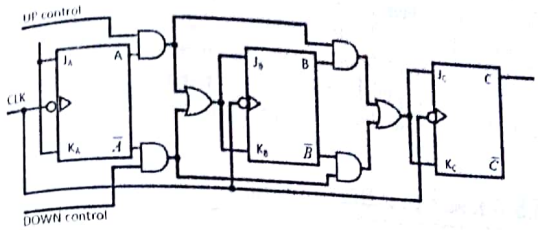
Count	C	B	A
7	1	1	1
6	1	1	0
5	1	0	1
4	1	0	0
3	0	1	1
2	0	1	0
1	0	0	1
0	0	0	0

Wave form:



8.3.3 Three bit up - down counter

Counter are also available in integrated circuit form as up/down counter which can be made to operate as either up or down counter. The counter counts up when UP control is logic 1 and DOWN control is logic 0. Similarly the counter counts downward when UP control input is logic 0 and DOWN control input is logic 1.



Advantages of synchronous counter over a ripple counter:

a) It has no limit to its highest operating frequency. Faster in operation if high frequency is applied.

b) There is not possibility of glitches i.e. an undesired positive or negative pulse appearing at the outside of logic gate with a ripple counter.

c) Here settling time is equal to the delay time of a single flip flop.

8.4 Changing the counter modulus

Modulus is defined as the number of states through which a counter can progress.The counters which progress 1 count at a time in a strict binary progression, and they all have a modulus given by 2^n, where n=number of flip-flops. Such counters are said to have a "naturalcount" of 2^n. For example,

→ mod2 counter consists of 1 flip-flop, and it counts two discrete states (0->1)

→ mod4 counter consists of 2 flip-flops, and it counts through 4 discrete states (00->01->10-> 11)

→ mod8 counter consists of 3 flip-flops, and it counts through 8 discrete states (000->001 ->010->011->100->101->110->111)

It is often desirable to construct counters having a modulus other than 2, 4, 8 and so on. Forexample, a counter having a modulus of 3 or 7 would be useful. A small modulus counter can always be

constructed from a larger modulus counter by skipping states. Such counters are said to have a *modified-count.*

Firstly, it is necessary to determine the number of flip-flops required. The correct number of flip flops is determined choosing the lowest natural count that is greater than the desired modified count. For example, a mod-7 counter requires 3 flip-flops, since 8 is the lowest natural count greater than the desired modified count of 7.

8.5 Decade and BCD counter

A **decading counter** is one that goes through 10 unique output combinations and then reset as the clock proceed further. Since it is a MOD-10 counter, it can be constructed with a minimum of four flip flop. A four flip flop counter would have a 16 states. By skipping any of the six states, by using some kind of feedback or some kind of additional logic, we can convert a normal four bit counter in to decade counter. A decade counter does not necessary count from 0000 to 1001. It could even count as 0000, 0001, 0010, 0101, 0110, 1001, 1010, 1100, 1101, 1111, 0000,..... , in this count sequence we have skipped 0011, 0100, 0111, 1000,1011 and 1110.

A **BCD counter** is a special case of decade counter in which the counter counts from 0000 to 1001 and then reset. Different counter state in this counter are binary equivalent of the decimal number 0 to 9.

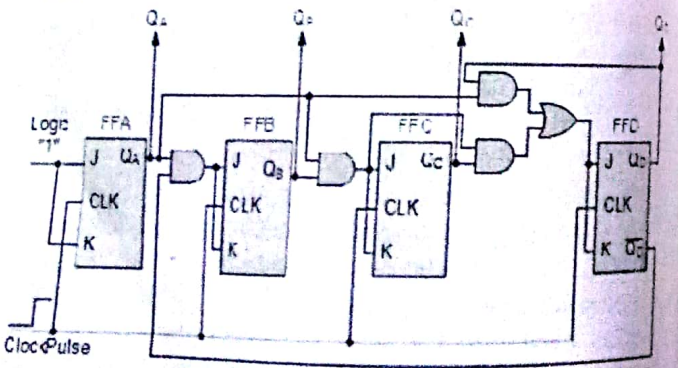


Fig: decade four bit synchronous counter

8.6 Presettable counter

Presettable counter are those that can be preset to any starting count either asynchronous (independent of clock signal) or synchronous (with the active transition of the clock signal). The presettable operation is achieved with the help of PRESET and CLEAR input available in the flip flop. The presettable operation is known as the 'preloading' or simply 'loading' operation. known. The diagram below shows a 3-bit asynchronously presettable synchronous up counter.

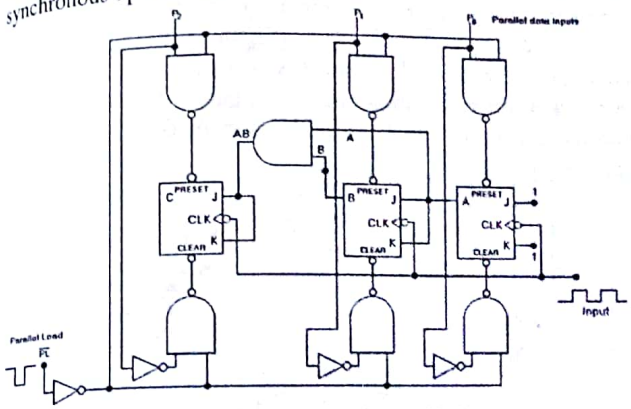


Fig: 3-bit Synchronous Binary Presettable Counter

In the diagram above, the J, K and CLK inputs are wired the same way as a synchronous up counter. The asynchronous PRESET and CLEAR inputs are used to perform the asynchronous presetting. The counter is loaded by applying the desired binary number to the inputs P_2, P_1 and P_0 and a LOW pulse is applied to the PARALLEL LOAD input, not(PL). This will asynchronously transfer P_2, P_1 and P_0 into the flip-flops. This transfer occurs independently of the J, K, and CLK inputs. As long as not(PL) remains in the LOW state, the CLK input has no effect on the flip-flop. After not(PL) returns to high, the counter resumes counting, starting from the number that was loaded into the counter.

For the example above, say that $P_2 = 1$, $P_1 = 0$, and $P_0 = 1$. When not(PL) is high, these inputs have no effect. The counter will perform normal count-up operations if there are clock pulses. Now

let's say that not(PL) goes low at $Q_2 = 0$, $Q_1 = 1$ and $Q_0 = 0$. This will produce LOW states at the CLEAR input of Q_1, and the PRESET inputs of Q_2 and Q_0. This will make the counter go to state 101 regardless of what is occurring at the CLK input. The counter will remain at state 101 until not(PL) goes back to HIGH. The counter will then continue counting from 101.

8.7 Counter design as a synthesis problem

8.7.1 MOD – 5 counter

Suppose we are going to design a MOD – 5 counter. While designing counter we can use either JK - flip flop or T - flip flop. It is easier to design a counter using T – flip flop then JK flip flop. To design a modulo – 5 counter firstly we have to draw a state transition diagram. It is shown below in figure 1. We need four flip flop for this and we use T-flip flop named A, B, C as a memory element of design.

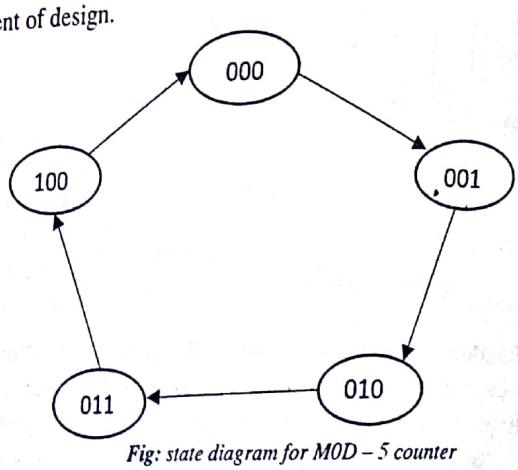


Fig: state diagram for MOD – 5 counter

With three flip flop 8 different state are possible but in our design 101, 110, 111 are not used in a counting sequence.

Excitation table for JK flip flop:

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

State table:

Present state			Next state			J_C	K_C	J_B	K_B	J_A	K_A
C	B	A	C	B	A						
0	0	0	0	0	1	0	x	0	x	1	x
0	0	1	0	1	0	0	x	1	x	x	1
0	1	0	0	1	1	0	x	x	0	1	x
0	1	1	1	0	0	1	x	x	1	x	1
1	0	0	0	0	0	x	1	0	x	0	x

In k maps:

For J_C

C \ BA	00	01	11	10
0	0	0	1	0
1	x	x	x	x

$J_C = BA$

For K_C

C \ BA	00	01	11	10
0	x	x	x	x
1	1	x	x	x

$K_C = 1$

For J_B

C \ BA	00	01	11	10
0	0	1	x	x
1	0	x	x	x

$J_B = B'A$

For K_B

C \ BA	00	01	11	10
0	x	x	1	0
1	x	x	x	x

$K_B = BA$

For J_A

C \ BA	00	01	11	10
0	1	x	x	1
1	0	x	x	x

$J_A = C'$

For K_A

C \ BA	00	01	11	10
0	x	1	1	x
1	x	x	x	x

$K_A = 1$

Design:

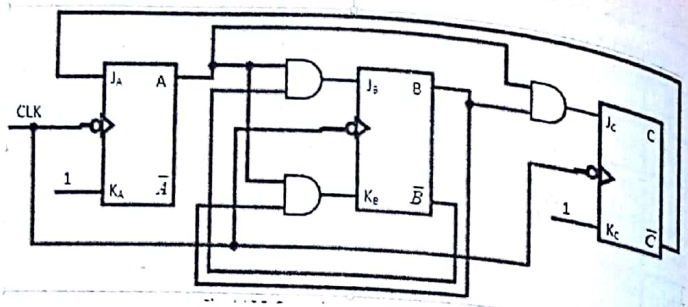
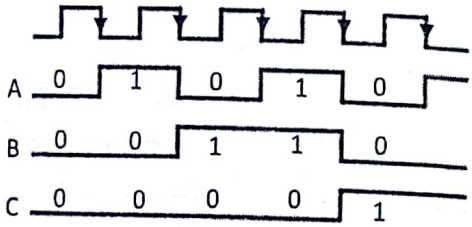


Fig: MOD-5 synchronous counter

Wave form:

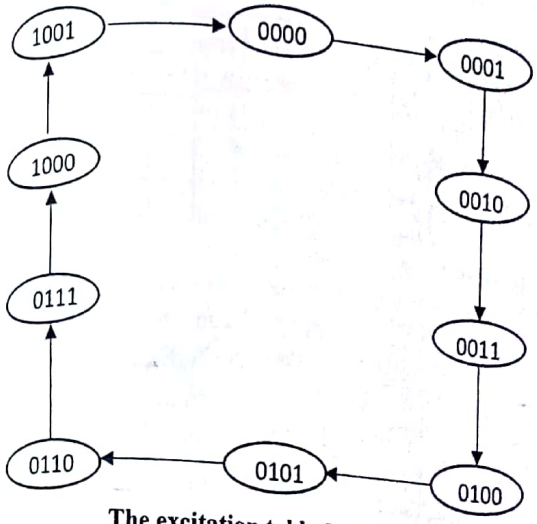


Q) What is counter? Design a MOD – 6 synchronous counter. Draw its timing diagram. *[IOE 2068 Baisakh]*

8.7.2 MOD-10 counter

We design MOD-10 counter using T flip flop. We need four T flip flop named A, B, C, and D

State transition diagram:



The excitation table for T flip flop:

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

State table:

Present State				Next State				T_D	T_C	T_B	T_A
D	C	B	A	D	C	B	A				
0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	1	0	0	1	0	0	0	1	1
0	0	1	0	0	0	1	1	0	0	0	1
0	0	1	1	0	1	0	0	0	1	1	1
0	1	0	0	0	1	0	1	0	0	0	1
0	1	0	1	0	1	1	0	0	0	1	1
0	1	1	0	0	1	1	1	0	0	0	1
0	1	1	1	1	0	0	0	1	1	1	1
1	0	0	0	1	0	0	1	0	0	0	1
1	0	0	1	0	0	0	0	1	0	0	1

For T_D

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

$T_D = AD+BCD$

For T_C

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

$T_C=CD$

For T_B

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

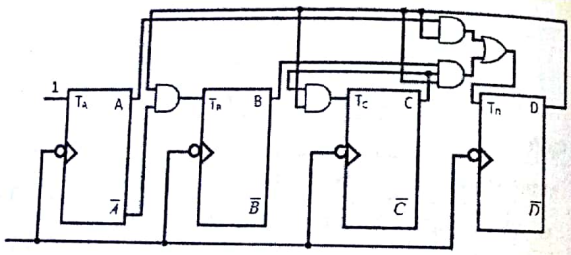
$T_B = A'D$

For T_A

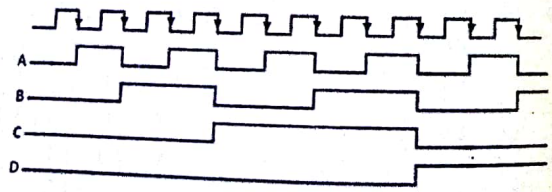
AB \ CD	00	01	11	10
00	1	1	1	1
01	1	1	1	1
11	x	x	x	x
10	1	1	x	x

$T_C = 1$

Design:



Wave form:



8.8 A Digital Clock

A digital clock, which displays the time of day in hours, minutes, and seconds, is one of the most common applications of counters. To construct an accurate digital clock, a very highly controlled basic clock frequency is required. For battery-operated digital clocks (or watches) the basic frequency can be obtained from a quartz-crystal oscillator. Digital clocks operated from the AC power line can use the 50 Hz power frequency as the basic clock frequency. In either case, the basic frequency has to be divided down to a frequency of 1 Hz or pulse of 1 second (pps). The basic block diagram for a digital clock operating from 50 Hz is shown in figure below.

The 50 Hz signal is sent through a Schmitt trigger circuit to produce square pulses at the rate of 50 pps. The 50 pps waveform is fed into a MOD-50 counter, which is used to divide the 50 pps down to 1 pps. The 1-pps signal is then fed into the SECONDS section. This section is used to count and display seconds from 0 through 59. The BCD counter advances one count per second. After 9 seconds the BCD counter recycles to 0. This triggers the MOD-6 counter and causes it to advance one count. This continues for 59 seconds. At this point, the BCD counter is at 1001 (9) count and the MOD-6 counter is at 101 (5). Hence, the display reads 59 seconds. The next pulse recycles the BCD counter to 0. This, in turn, recycles the MOD-6 counter to 0.

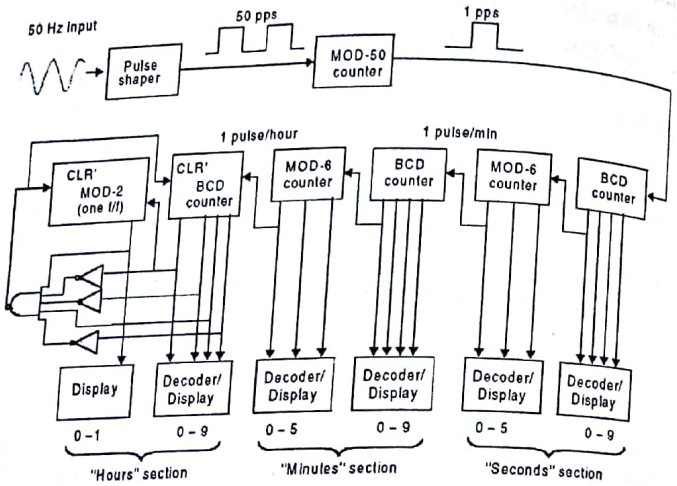


Fig: Block diagram for a digital clock

The output of the MOD-6 counter in the SECONDS section has a frequency of 1 pulse per minute. This signal is fed to the MINUTES section, which counts and displays minutes from 0 through 59. The MINUTES section is identical to the SECONDS section and operates in exactly the same manner. The output of the MOD-6 counter in the MINUTES section has a frequency of 1 pulse per hour. This signal is fed to the HOURS section, which counts and displays hours from 1 through 12. The HOURS section is different from the MINUTES and SECONDS section in that it never goes to the zero state. The circuitry in this section is different. When the hours counter reaches 12, it will be reset to zero by the NAND gate.

1) **Differentiate between synchronous and asynchronous counter.**

Asynchronous counter	Synchronous counter
– In this type of counter flip flop are connected in such a way that output of first flip flop derive the clock for next flip flop	– In this type there is no connection between output of first flip flop and clock input of next flip flop.
– All the flip flop are not clocked simultaneously.	– All the flip flop are clocked simultaneously
– Logic circuit are simpler even for more number of states.	– Design involves complex logic circuit as number of state increases
– Main drawback of this counter is their low speed as the clock is propagated through number of flip flop before it reaches last state	– As the clock is simultaneously given to all flip flop there is no problem of propagation delay. Hence they are high speed counter and are preferred when number of flip flop increases in given design.

Important Questions

1) Deign a MOD – 10 synchronous counter showing its state circuit diagram and output wave form. [IOE 2067 ashad]
2) Design a three bit down counter and show the timing diagram for 3 clock cycle, assuming that the initial reading of the counter is 0. [IOE 2067 magh]
3) Design a MOD – 5 binary ripple up counter. What are the advantage of synchronous counter over a ripple counter? [IOE 2066 bhadra]
4) Design a three bit binary synchronous up counter using the excitation table and dray its timing diagram. [IOE 2066 bhadra]

5) Design a synchronous decade counter and also show its timing diagram. [IOE 2069 chaitra]

6) Define the ripple counter. Explain the operation of MOD – 10 ripple counter with timing diagram. [IOE 2068 chaitra]

7) Differentiate between synchronous and asynchronous counter. Draw and explain the operation of asynchronous decade counter with clear timing diagram.

8) Design a counter with the following binary sequence 0, 1, 3, 2, 6, 4, 5, 7 and repeat. Use T flip flop. [PU Fall 2013]

9) What do you mean by counters? What does it counts? Describe decade counter with logic and timing diagram. [Purbanchal 2005]

❁❁❁

12

Chapter 9

Sequential Machine

Sequential logic circuit

The combinational circuit does not use any memory. Hence the previous state of input does not have any effect on the present state of the circuit. But sequential circuit has memory so output can vary based on input. *Hence the logic circuit whose output depends not only on the present value of its input signals but on the sequence of past inputs history is called sequential logic circuit.* It is a form of binary circuit design that employs one or more inputs and one or more outputs, whose states are related by defined rules that depend, in part, on previous states. Each of the inputs and output can attain either of two states: logic 0 (low) or logic 1 (high). So, in addition to combinational logic, it remember values and also base decisions on both input values and stored values.

Block diagram

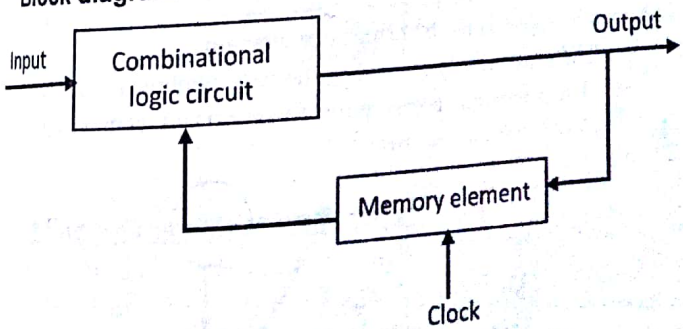


Fig: Sequential logic circuit

It is basically divided in to two types:

a) Synchronous sequential circuit
b) Asynchronous sequential circuit

Sequential logic circuit and FSM (finite state machine):

Building block of FSM is sequential logic circuit like flip flop. FSM, is a mathematical model of computation used to design sequential logic circuits. It is conceived as an abstract machine that can be in one of a finite number of states. The machine is in only one state at a time; the state it is in at any given time is called the **current state**. It can change from one state to another when initiated by a triggering event or condition; this is called a **transition.** A particular FSM is defined by

- An initial state or record
- A set of possible input events
- A set of new states that may result from the input
- A set of possible actions or output events that result from a new state

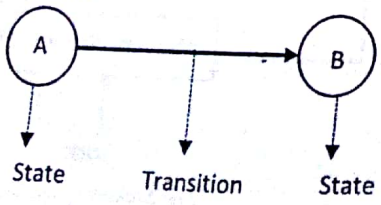
Example:

a) Street traffic light controller device is FSM setting traffic lights to RGB values, depending on the combination of current light setting, and input from optical sensors on the road.
b) Elevators, whose sequence of stops is determined by the floors requested by riders.

State diagram:

FSM can be represented by using state diagram.

- The state are represented by the state symbols (circles).
- The transition are represented by arrows connecting the state symbols.



Synchronous machine:

A synchronous circuit is a digital circuit in which the changes in the state of memory elements are synchronized by a clock signal. Here data is stored in memory devices called flip-flops or latches. The input are pulses (or levels and pulses) with certain restrictions on pulse width and circuit propagation delay. Therefore synchronous circuits can be divided into clocked sequential circuits and unclocked or pulsed sequential circuits.

There are two different ways of state machine design Moore model and Mealy model.

a) **Moore model :**

Moore machine is an FSM whose outputs depend on only the present state. A Moore machine can be described by a 6 tuple $(Q, \Sigma, O, \delta, X, q_0)$ where

- Q is a finite set of states.
- Σ is a finite set of symbols called the input alphabet.
- O is a finite set of symbols called the output alphabet.
- δ is the input transition function where $\delta: Q \times \Sigma \rightarrow Q$
- X is the output transition function where $X: Q \times \Sigma \rightarrow O$
- q_0 is the initial state from where any input is processed ($q_0 \in Q$).

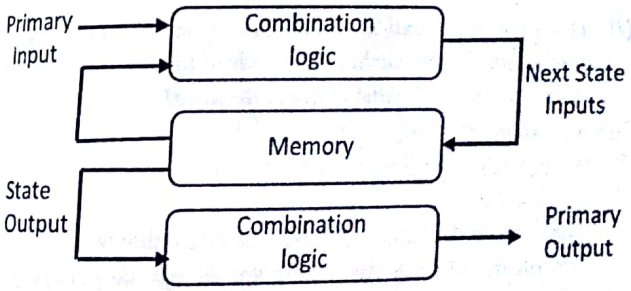


Fig: Moore model

Taking the example of traffic light:

When switch is pressed, traffic light is on and "yellow" light glows, again when it is pressed "green" light glows, again when switch is pressed "red" light glows in last when switch is pressed it

goes to "off" state which is initial state. This can be shown in state diagram by using Moore model as:

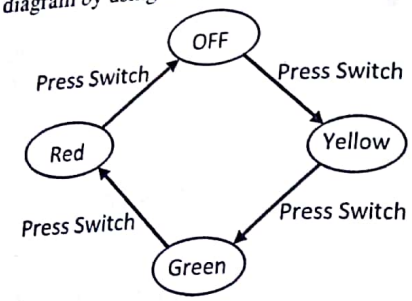


Fig: State diagram of traffic signal using Moore model

In Moore model outputs are defined inside the circles. It seems a bit more intuitive in their rotation. Their transition conditions are listed in their rotation. The difference between Moore and Mealy model is only in the way the output is generated.

Output change rule: Only on state change.

Note: Compare the above traffic light state diagram to Mealy diagram to be more clear in parallel way.

Q) Design a sequential machine having one serial input and one output Z. the machine is required to give an output Z=1 when the input X contains the pattern 001.

Suppose input pattern:

X = 0 1 1 0 0 1 0 0 1 0 0 0 0 1

Z = 0 0 0 0 0 1 0 0 1 0 0 0 0 0

Here when the pattern 001 completes as shown above output becomes 1. We use Moore Machine design the state graph:

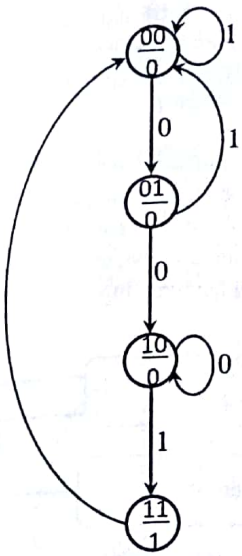


Fig: state diagram

Steps:

1) List the state number and output condition inside the circle. Here we have four state number state-0 **"00"**, state-1 **"01"**, state-2 **"10"**, state-3 **"11"** and their respective output condition are **0, 0, 0, 1.**
2) When input is '0' at state-0 we go to state-1 since it is a part of sequence '001' but when input is '1' we remain in state-0.
3) When input is '0' at state-1 we go to state-2 since it is a part of sequence '001' but, when input is '1' we move to state-0 because 1 already occurs in it which reduce the hardware.
4) When input is '1' in state-2 we go to state-3 since it is the part of sequence '001' but when input is '0' it remains in same state-2.
5) After last state-3 the transition automatically moves to state-0.

b) Mealy model:

A Mealy Machine is an FSM whose output depends on the present state as well as the present input. It can be described by a 6 tuple $(Q, \sum, O, \delta, X, q_0)$ where

- Q is a finite set of states.
- $\sum$ is a finite set of symbols called the input alphabet.
- O is a finite set of symbols called the output alphabet.
- δ is the input transition function where δ: $Q \times \sum \rightarrow Q$
- X is the output transition function where X: $Q \rightarrow O$
- q_0 is the initial state from where any input is processed ($q_0 \in Q$).

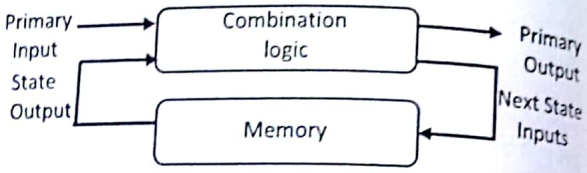


Fig: Mealy model

Taking the example of traffic light as similar to Moore model:

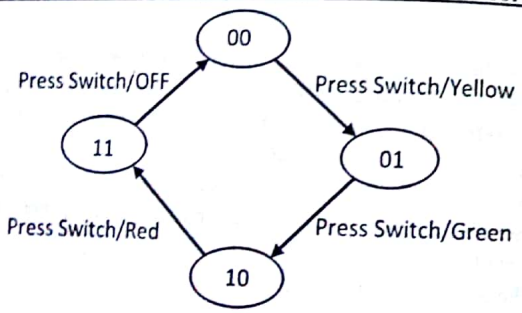


Fig: State diagram of traffic signal using Moore model

Here in Mealy model input and output variables are listed in transition arrow. It seems bit more abstraction in their notation. It is more flexible and effect then Moore model. It also requires less hardware to implement.

Output change rule: May change with input change.

1) **Design a sequential machine having one serial input and one output Z. the machine is required to give an output Z=1 when the input X contains the pattern 001.**

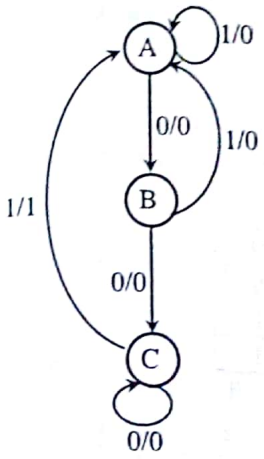
Solution:

Suppose input pattern:

X = 0 1 1 0 0 1 0 0 1 0 0 0 1

Z = 0 0 0 0 0 1 0 0 1 0 0 0 0

Here when the pattern 001 completes as shown above output becomes 1. We use Mealy Machine design the state graph is:



A= Reset

B= 0 (Here 0 already occurs)

C= 00 (Here 00 already occurs)

Fig: state diagram

In Mealy machine design technique transition and output both are listed in transition arrow.

Explanation of state diagram:

- At A, we have two input either '0' or '1' we must careful which is the part of pattern '001'. Here '0' is part of pattern so it proceeds to next state and '1' has no use.
- At B, when '0' occurs it proceeds to next state since it is part of '001' but 1 has no meaning so it moves to state A.
- At C, we have either '0' or '1'. To complete pattern we have to take '1'. Here we have to move on state A because it stores first 1. Since it do not contain overlapping so we moved to reset condition.

None overlapping of sequence

X = 0 1 1 0 0 1 0 0 1 0 0 0 0 1

Overlapping of sequence of pattern is 010

X = 0 1 1 0 1 0 1 0 0 1 1 0 1 1

Note: When none overlapping occurs it is simply starting from new i.e. A.

When overlapping occurs it is like proceeding from old one. At C, '0' do not make pattern '001' complete it remains in C.

Present State Y_1Y_2	*Next State* Y_1Y_2		*Output* Z	
	at X=0	at X=1	at X=0	at X=1
A	B	A	0	0
B	C	A	0	0
C	C	A	0	1

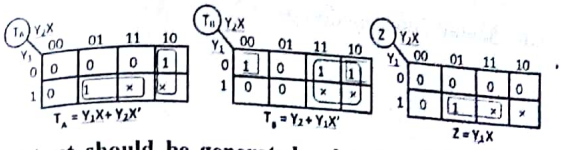
We know the transition table for T flip-flop

Q_t	Q_{t+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

Assigning A= 00, B= 01 and C= 10.

Present State Y_1	Y_2	*Input* X	*Next State* Y_1	Y_2	*Output* Z	T_A	T_B
0	0	0	0	1	0	0	1
0	0	1	0	0	0	0	0
0	1	0	1	0	0	1	1
0	1	1	0	0	0	0	1
1	0	0	1	0	0	0	0
1	0	1	0	0	1	1	0

Fig: state table



Output should be generated only when it is required and all the times an output should be zero in place of Don't care condition.

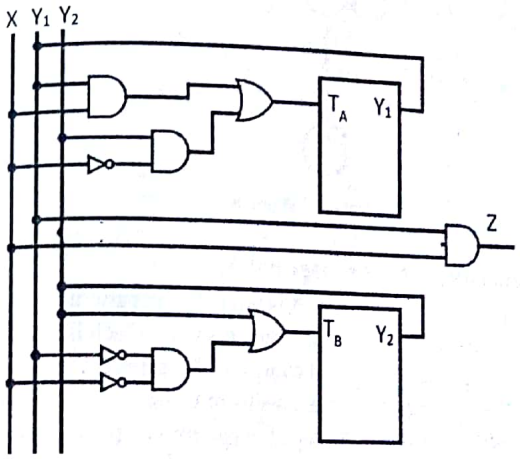


Fig: circuit diagram

2) Design a sequential machine that has one serial input and one output Z. The machine is required to have an output Z= 1, when the input X contains the message 010.

Solution:

Suppose the input pattern and its corresponding output be:

X = 0 1 1 0 0 1 0 1 0 0 0 0 0 1

Z = 0 0 0 0 0 0 1 0 1 0 0 0 0 0

Here the pattern '010' contains first bit '0' and last bit is also '0' so it contains overlapping patterns.

Using Malay machine design

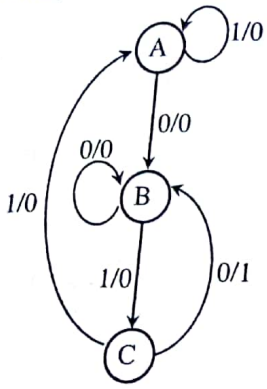


Fig: state diagram

Explanation of state diagram:

- If the input is '0' at A it belongs to the pattern '010' so it goes to the next state B. But 1 remains in A which is reset state.
- If input is '1' at B it belongs to the pattern '010' so it goes to the next state C. But 0 remains in B until 1 is not achieved.
- When third bit is '0' then the pattern completes and we have to go on B which stores the first '0'. This may be the starting of next pattern as pattern is overlapping pattern.

Present State Y_1Y_2	*Next State* Y_1Y_2		*Output Z*	
	at X=0	at X=1	at X=0	at X=1
A	B	A	0	0
B	B	C	0	0
C	B	A	1	0

Assigning A = 00, B= 01, and C= 10

Present state Y_1Y_2	*Input X*	*Next State* Y_1Y_2	*Output Z*	T_A	T_B
00	0	01	0	0	1
00	1	00	0	0	0
01	0	01	0	0	0
01	1	10	0	1	1
10	0	01	1	1	1
10	1	00	0	1	0

3) **The output of the machine is to be set high when the data in the input 1001 in sequence starting from MSB.**

Solution:

Suppose a pattern:

X = 0 1 1 1 0 1 0 0 1 0 0 1 0 1

Z = 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0

Here the pattern contains overlapping (last '1' of first sequence is first '1' of other). When 1001 pattern completes then output becomes 1.

Using Malay machine design

State graph:

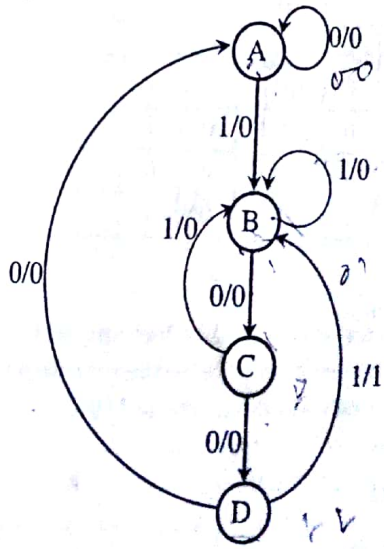


Fig: state diagram

Explanation of state diagram:

- If input is '1' at A then it moves to next state B as it belongs to pattern '1001', but '0' remains in A.
- When input is '0' at A it belongs to pattern '1001' so move to next state C but '1' remains in B
- When input is '0' at C it belongs to pattern '1001' so move to next state D, but '1' moves to B because it has no meaning as it do not belongs to pattern.
- When input is '1' at D it belongs to pattern '1001', we have to go in state B because it stores first '1'. This '1' could be the starting of next pattern.

Assigning A= 00, B= 01, C= 10, D= 11

Present State		*Input X*	*Next State*		*Output Z*	T_A	T_B
Y_1	Y_2		Y_1	Y_2			
0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	1
0	1	0	1	0	0	1	1
0	1	1	0	1	0	0	0
1	0	0	1	1	0	0	0
1	0	1	0	1	0	1	1
1	1	0	0	0	0	1	1
1	1	1	0	1	1	1	0

Fig: state table

4) Design a sequential machine that has one serial input and one output Z. The machine required to give an output Z= 1, when the input X contain the message 1100.

Solution:

Suppose a pattern

X = 0 0 1 1 1 0 1 1 0 0 1 1 0 0 1 0

Z = 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0

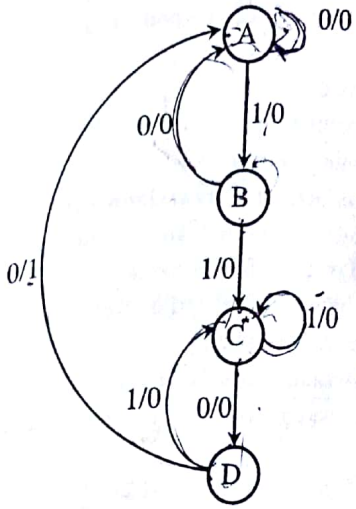


Fig: state diagram

Mealy Machine vs. Moore Machine

Mealy Machine	Moore Machine
Output depends both upon present state and present input.	Output depends only upon the present state.
Generally, it has fewer states than Moore Machine.	Generally, it has more states than Mealy Machine.
Output changes at the clock edges.	Input change can cause change in output change as soon as logic is done.
Mealy machines react faster to inputs.	In Moore machines, more logic is needed to decode the outputs since it has more circuit delays.

State reduction method:

State reduction is the elimination of states which are equivalent within the state machine and state assignment is the method of assigning a binary value to a state name that will create a reduced logic equation. The purpose of state reduction method is to reduce the number if states in synchronous machine without altering the

input and output relationship. It is performed in state table rather than the state diagram. The three methods examined for state reduction are

(1) Row matching method,
(2) Implication chart method, and
(3) Successive partitioning method.

The four methods examined for state assignment are

(1) Binary, (2) Gray code, (3) One hot, and (4) Enumerated.

Row matching method is quite simple method. The typical recipe for reducing in the row reduction method is (Katz, 1993):

1. Start with the state transition table
2. Identify states with same output behavior
3. If such states transition to the same next state, they are equivalent
4. Combine into a single new renamed state
5. Repeat until no new states are combined

Example of state reduction method:

Equivalent state table:

Present state	Next state		Output	
	X=0	X=1	X=0	X=1
A	A	B	1	1
B	D	C	0	0
C	A	F	0	1
D	F	C	0	0
E	A	F	0	1
F	D	C	0	0

Going through the state table we look for two present state that goes to the same next state and have the same output for both input combination. State **"C" and "E"** are two such state: they both go to state **"A" and "F"** and have the output of 0 and 1. So the state C and E are equivalent and one can be removed. The procedure of removing the state and replacing it by its equivalent is demonstrated in fig below:

Present state	Next state		Output	
	X=0	X=1	X=0	X=1
A	A	B	1	1
B	D	C	0	0
C	A	F	0	1
D	F	C	0	0
F	D	C	0	0

Again state B and F are same so one can be removed:

Present state	Next state		Output	
	X=0	X=1	X=0	X=1
A	A	B	1	1
B	D	C	0	0
C	A	~~F~~ B	0	1
D	~~F~~ B	C	0	0

F is removed and replaced by B in above table.

Reduced state table is below

Present state	Next state		Output	
	X=0	X=1	X=0	X=1
A	A	B	1	1
B	D	C	0	0
C	A	B	0	1
D	B	C	0	0

Here no further simplification is possible so this is known as row matching.

State assignment:

The example of state assignment is shown in figure below:

State	Assignment
A	00
B	01
C	10
D	11

The reduced state table with binary assignment in below in table:

Present state	Next state		Output	
	X=0	X=1	X=0	X=1
00	00	01	1	1
01	11	10	0	0
10	00	01	0	1
11	01	10	0	0

Excitation table for above reduced state table is given below:

Present state		Input	Next state		Flip flop input		Output
A	B	X	A	B	T_A	T_B	Z
0	0	0	0	0	0	0	1
0	0	1	0	1	0	1	1
0	1	0	1	1	1	0	0
0	1	1	1	0	1	1	0
1	0	0	0	0	1	0	0
1	0	1	0	1	1	1	1
1	1	0	0	1	1	0	0
1	1	1	1	0	0	1	0

Follow the same process as above for K-map and circuit diagram.

Asynchronous machines:

Asynchronous sequential circuit has no clock internal state changes when there is a change in a input variable. Memory is efficiently provided by the finite time taken by the signal propagation through gates. They are used when the fast response in tan input changes, without having to wait for the clock transition, is necessary. They are also used where the introduction of extra component related to a clock must be avoided.

Fundamental mode:

Asynchronous machine operate under the restriction that:

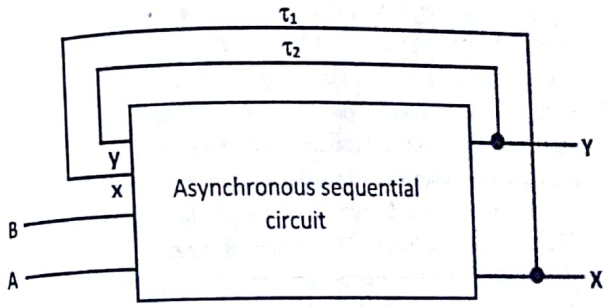
a) Only one input variable may changes with time.

b) The time between input changes is greater than the time required to reach steady state.

Problem related to asynchronous machine:

Asynchronous circuit responds to all the transient values and problems like oscillation, critical race, hazards can cause major problem unless they addressed at design stage. To explain these problems we take help of truth table shown below, where the circuit has two external inputs A, B and two outputs X, Y. Both the output are fed back to the input side in the form of x, y cannot change simultaneously but with time delays τ_1 and τ_2 respectively and we can write.

$x = X(t-\tau_1)$ and $y = Y(t-\tau_2)$



Fie: Block diaeram

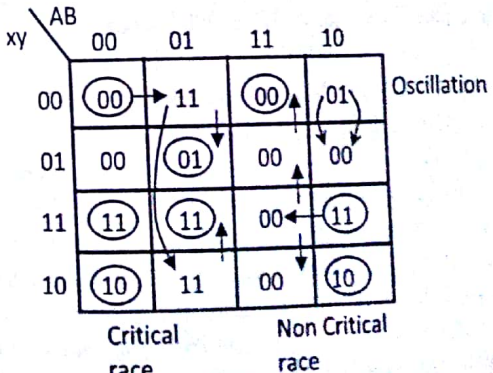


Fig: Truth table of two input, two output circuit

The major problems with truth table are discussed below:

a) Oscillation

Consider the stable state xyAB = 0000, where x = X and y =Y. If input AB changes from 00 to 10, the circuit goes to xyAB = 0010 state and then output xy = 0. This is a transient state because xy = XY. After time τ_1, y takes the value of Y = 1 and the circuit goes to xyAB = 0110 where output XY = 00. This again is a transient state and after another propagation τ_2 the circuit goes to xyAB = 0010. Thus the circuit oscillates between state 0010 and 0110 and the output τ oscillates between 0 and 1 with a time gap τ_2.

b) Hazards

Static and dynamic hazards causes malfunctioning of asynchronous sequential circuit. Situations like Y = A + A' or Y = A A' are to be avoided for any input output combination with the help of hazard covers in truth table. In circuit with feedback even when those hazards and adequately covered there can be another problem called *essential hazard*. This occurs when change in input does not react one part of the circuit while from other part one output feedback to the input side become available. The essential hazard is avoided by adding delay, may be in the form of additional gates that does not change the logic level, in the feedback path.

c) Critical race

Consider the stable state xyAB = 0000, now if AB changes to 01 the circuit moves to xyAB = 0001 where XY = 11, now depending which τ_1 and τ_2 is lower, xy moves from 00 to either 01 or 10. If τ_1 is lower, x changes earlier and the circuit goes to xyAB = 1001 which is an unusual state with output XY = 11 and xy ≠ XY. The circuit next moves to state xyAB = 1101 which is a stable state and final output XY = 11. If τ_2 is lower, y changes earlier and the circuit goes to xyAB = 0101, a stable state and final output is 01. *Thus depending on propagation delays in feedback path, the circuit settles at two different states generating two different set of outputs such a situation is called critical race condition.*

Problem Solved

1) **Design synchronous sequential circuit for the given state diagram using JK flip flop.** [IOE 2069 Ashad]

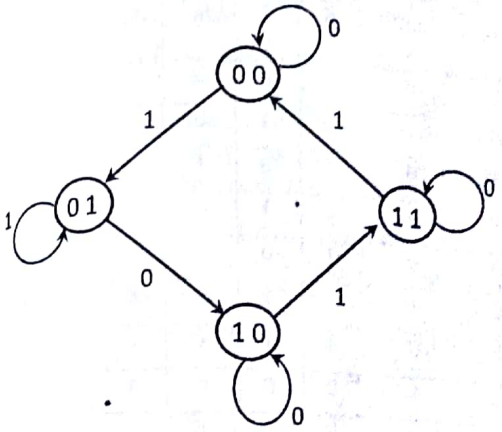


Fig: state diagram

⇒The excitation table for JK flip-flop is,

Q_t	Q_{t+1}	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

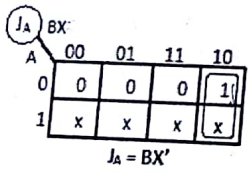
Now the state table is,

(Note that during forming state table check A of present state with A of next state and fill for flip flops with checking the respective state of excutation table. The process is also similar for B.)

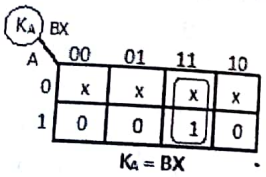
Present state A	Present state B	Input Control X	Next State A	Next State B	J_A	K_A	J_B	K_B
0	0	0	0	0	0	x	0	x
0	0	1	0	1	0	x	1	x
0	1	0	1	0	1	x	x	1
0	1	1	0	1	0	x	x	0
1	0	0	1	0	x	0	0	x
1	0	1	1	1	x	0	1	x
1	1	0	1	1	x	0	x	0
1	1	1	0	0	x	1	x	1

Fig: state table

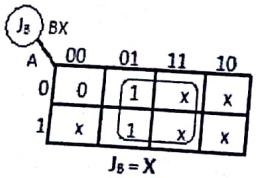
Now, using K-map,



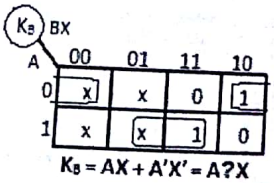
$J_A = BX'$



$K_A = BX$



$J_B = X$



$K_B = AX + A'X' = A?X$

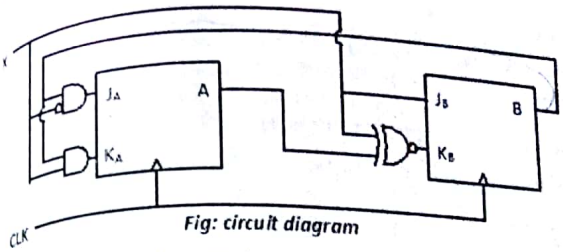


Fig: circuit diagram

Note: The circuit diagram is not necessary to draw as it is presented here in this book. The main concept is : you have to manage and connect with less path crossing.

2) **Design a sequential circuit with two D FFs, A and B, and one input X. When X = 0, the state of the circuit remains the same. When X = 1, the circuit goes through the state transitions from 00 to 01 to 11 to 10 back to 00 and repeats.**

Solution:

State A	State B	Input X	Next state A	Next state B	D_A	D_B
0	0	0	0	0	0	0
0	0	**1**	**0**	**1**	0	1
0	1	0	0	1	0	1
0	1	**1**	**1**	**1**	1	1
1	0	0	1	0	1	0
1	0	**1**	**1**	**0**	1	0
1	1	0	1	1	1	1
1	1	**1**	**0**	**0**	0	0

3) **Design a sequential circuit with two D flip flop and two inputs, P and Q. If P = 0, the circuit remains in the same state regardless of the value of Q. When P = 1 and Q = 1, the circuit goes through the state transition from 00 to 01 to 10 back to 00, and repeats. When P = 1 and Q = 0, the circuit goes through the state transition from 00 to 10 to 01 back to 00, and repeats. The circuit is to be designed by treating the unused state (s) as don't care condition.** [IOE 2071 Chaitra]

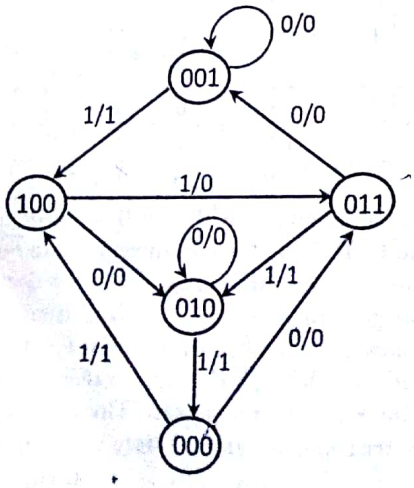
Solution:

Here the input value PQ = 11 must appear three times since it goes from 00 to **01 to 10 to 00** and repeats.
The input values PQ= 10 also appear for three times since it also goes from state 00 to **10 to 01 to 00**

Present state		Input		Next state		D_A	D_B
A	B	P	Q	A	B		
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	**1**	**0**	**1**	**0**	1	0
0	0	**1**	**1**	**0**	**1**	1	1
0	1	0	0	0	0	0	0
0	1	0	1	0	0	0	1
0	1	**1**	**0**	**0**	**1**	1	0
0	1	**1**	**1**	**1**	**0**	1	1
1	0	0	0	0	0	0	0
1	0	0	1	0	0	0	1
1	0	**1**	**0**	**0**	**0**	1	0
1	0	**1**	**1**	**0**	**0**	1	1

Design K map and circuit in similar above basis.

4) A sequential circuit has one input and one output. The state diagram is shown in figure. Design the sequential circuit with RS flip flop. [IOE 2067 Ashad]



The excitation table of the SR flip flop is;

Q_t	Q_{t+1}	S	R
0	0	0	x
0	1	1	0
1	0	0	1
1	1	x	0

The state table of given state diagram is:

Present state			Input	Next state			Output Y	S_A	R_A	S_B	R_B	S_C	R_C
A	B	C	X	A	B	C							
0	0	0	0	0	1	1	0	0	x	1	0	1	0
0	0	0	1	1	0	0	1	1	0	0	x	0	x
0	0	1	0	0	0	1	0	0	x	0	x	x	0
0	0	1	1	1	0	0	1	1	0	0	x	0	1
0	1	0	0	0	1	0	0	0	x	x	0	0	x
0	1	0	1	0	0	0	1	0	x	0	1	0	x
0	1	1	0	0	0	1	0	0	x	0	1	x	0
0	1	1	1	0	1	0	1	0	x	x	0	0	1
1	0	0	0	0	1	0	0	0	1	1	0	0	x
1	0	0	1	0	0	0	1	0	1	0	x	0	x

Now using K-map,

S_A CX / AB	00	01	11	10
00	0	1	1	0
01	0	0	0	0
10	x	x	x	x
11	0	0	x	x

$S_A = A'B'X$

R_A CX / AB	00	01	11	10
00	x	0	0	x
01	x	x	x	x
10	x	x	x	x
11	1	1	x	x

$R_A = A$

S_B CX

AB \ CX	00	01	11	10
00	1	0	0	0
01	x	0	x	0
10	x	x	x	x
11	1	0	x	x

$S_B = C'X'$

R_B CX

AB \ CX	00	01	11	10
00	0	x	x	x
01	0	1	0	1
10	x	x	x	x
11	0	x	x	x

$R_B = C'X+CX'=C\oplus X$

S_C CX

AB \ CX	00	01	11	10
00	1	0	0	x
01	0	0	0	x
10	x	x	x	x
11	0	0	x	x

$S_C = A'B'X'$

R_C CX

AB \ CX	00	01	11	10
00	0	x	1	0
01	x	x	1	0
10	x	x	x	x
11	x	x	x	x

$R_C = X$

Y CX

AB \ CX	00	01	11	10
00	0	1	1	0
01	0	1	1	0
10	x	x	x	x
11	0	1	x	x

$Y = X$

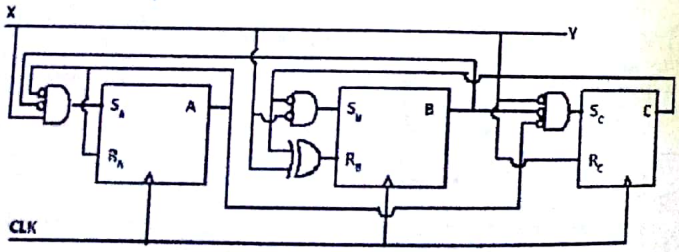


Fig: circuit diagram

5) **Design a sequential machine that can go through 2-bit gray code combination of states. The machine changes its state when serial input is one and remains in the same state when input is zero. The machine produces output one when it passes through all states and finally goes back to initial state. (use JK flip-flop)** [IOE 2071 Shrawan]

Here,

Binary	Gray
00	00
01	01
10	11
11	10

For 2-bit gray code, the sequence is 00 -- 01 -- 11 -- 10.
Then the state diagram can be drawn as:

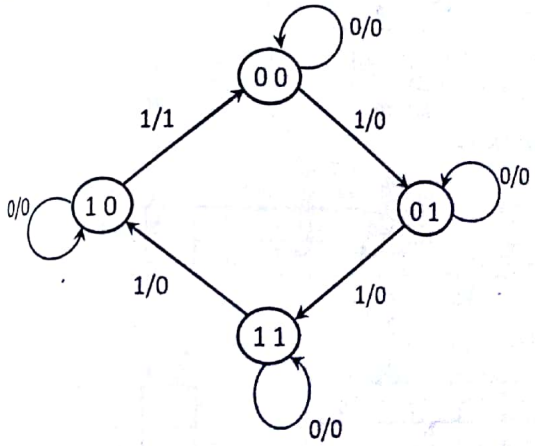


Fig: state diagram

We know the transition table for JK flip-flop

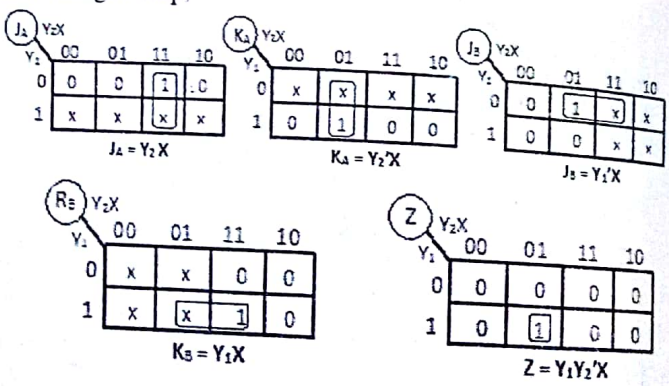
Q_t	Q_{t+1}	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

Now the state table is,

Present State Y_1Y_2	*Input X*	*Next State Y_1Y_2*	*Output Z*	J_A	K_A	J_B	K_B
00	0	00	0	0	x	0	x
00	1	01	0	0	x	1	x
01	0	01	0	0	x	x	0
01	1	11	0	1	x	x	0
10	0	10	0	x	0	0	x
10	1	00	1	x	1	0	x
11	0	11	0	x	0	x	0
11	1	10	0	x	0	x	1

Fig: state table

Now using K-map,



6) **Design the state diagram from below circuit diagram.**

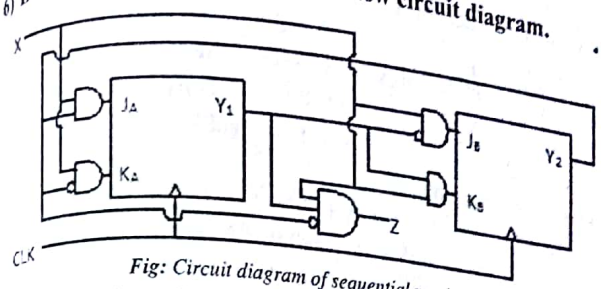


Fig: Circuit diagram of sequential machine

Here in the above figure D input determine flip-flop next state. There are basically three steps they are:

Step1: Find out the input and output equation.

The set of next state equation of circuit are:

D_A or $A(t+1) = A + B X'$

D_B or $B(t+1) = B' X + B X'$

$Z = A B' X'$

The state equation is the algebraic expression that specifies the condition for a flip-flop state transition.

Step2: Determine state table.

Present state		*Input*	*Next state*		*Output*
A	*B*	*X*	*A*	*B*	
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	1	1	0
0	1	1	0	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	1	0
1	1	1	1	0	0

Here next state A and B are D_A and D_B respectively. To obtain the value put present state A , B and input X value in above equation. For first: A = 0, B = 0 and X = 0

Putting in above equation

D_A or A(t+1) = A + B X'=0+0.1=0 **Next state (A)**

D_B or B(t+1) = B' X + B X' = 1.0+0.1=0 **Next state (B)**

Z = A B' X' = 0.1.1 =0 **Output**

For second: A = 0, B = 0 and X = 1

Putting in above equation

D_A or A(t+1) = A + B X' = 0+0.0=0 **Next state (A)**

D_B or B(t+1) = B' X + B X' = 1.1+0.0=1 **Next state (B)**

Z = A B' X' = 0.1.0 =0 **Output**

Similarly we can do for others.

Step 3: State diagram:

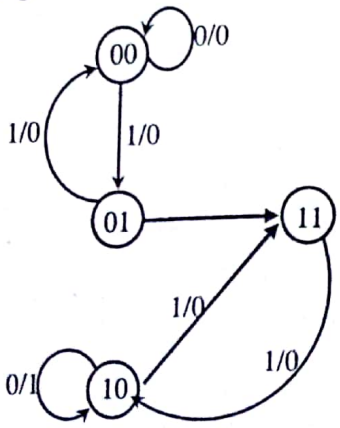


Fig: state diagram

7) The output of the machine is to be set *high* when the data in the input *110* in sequence starting from MSB (use SR flip-flop). [IOE 2068 Baishakh]

Note: For easy you can use T flip-flop instead of using other flip-flops if the question do not fix the type of flip-flop. But in this case you must use SR flip flop

Here,

Input X = 1 1 0

Output Z = 0 0 1

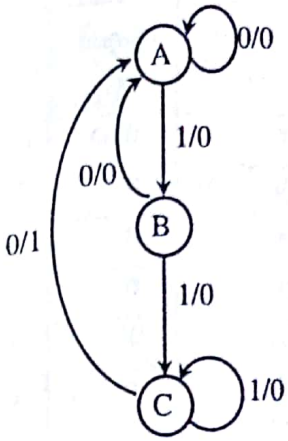


Fig: state diagram

Present State Y_1Y_2	*Next State* Y_1Y_2		*Output* Z	
	at X=0	at X=1	at X=0	at X=1
A	A	B	0	0
B	A	C	0	0
C	A	C	1	0

Fig (a): Transition table

We know the transition table for SR flip-flop

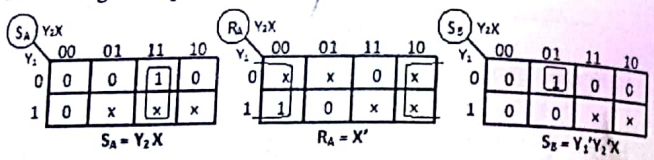
Q_t	Q_{t+1}	S	R
0	0	0	x
0	1	1	0
1	0	0	1
1	1	x	0

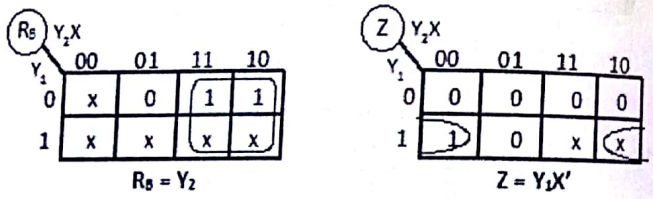
Now assigning the values on pure binary as A = 00, B = 01 and C = 10 and then merging the above transition table (*fig a*) into a single column for both input (i.e. X=0 and X=1) as follows.

Present State Y_1Y_2	Input X	Next State Y_1Y_2	Output Z	S_A	R_A	S_B	R_B
00	0	00	0	0	x	0	x
00	1	01	0	0	x	1	0
01	0	00	0	0	x	0	1
01	1	10	0	1	0	0	1
10	0	00	1	0	1	0	x
10	1	10	0	x	0	0	x

Fig: state table

Now using K-map,





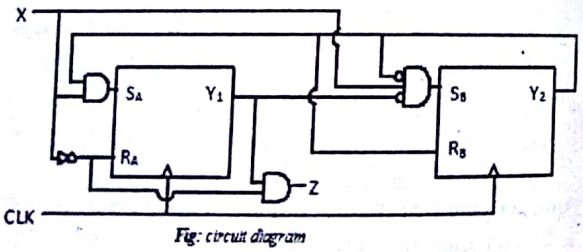


Fig: circuit diagram

8) A synchronous state machine has one bit serial input *X*. The output *Z* of machine is to be set high when the input contains the message *011*. Draw the state diagram for this machine and design the circuit. (Use T flip-flop).

[IOE 2068 shrawan]

Here,

Input X = 011

Output Z = 001

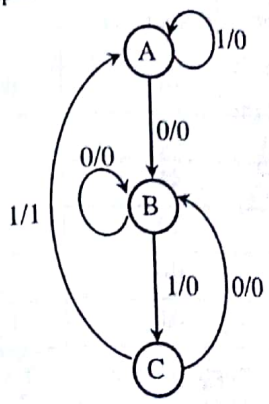


Fig: state diagram

Present State Y_1Y_2	Next State Y_1Y_2		Output Z	
	at X=0	at X=1	at X=0	at X=1
A	B	A	0	0
B	B	C	0	0
C	B	A	0	1

Fig (a): Transition table

We know the transition table for T flip-flop

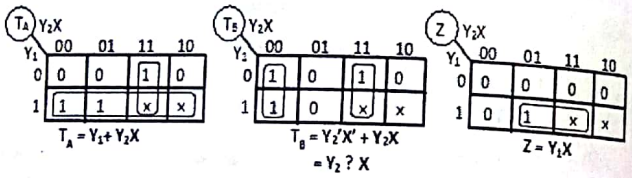
Q_t	Q_{t+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

Now assigning the values on pure binary as A = 00, B = 01 and C = 10 and then merging the above transition table (*fig a*) into a single column for both input (i.e. X=0 and X=1) as follows.

Present State Y_1Y_2	*Input X*	*Next State Y_1Y_2*	*Output Z*	T_A	T_B
00	0	01	0	0	1
00	1	00	0	0	0
01	0	01	0	0	0
01	1	10	0	1	1
10	0	01	0	1	1
10	1	00	1	1	0

Fig: state table

Now using K-map,



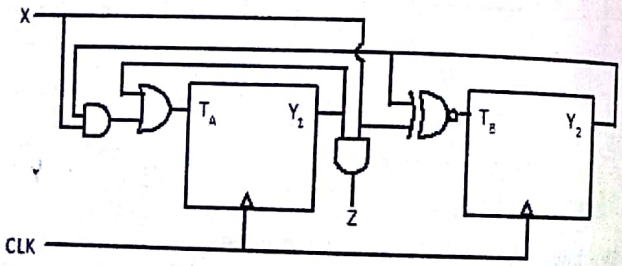


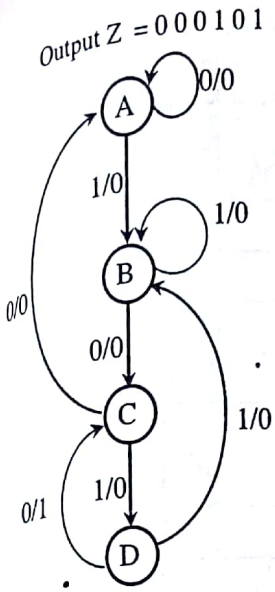
Fig: circuit diagram

9) Design a sequential machine that has one serial input and one output Z. The machine is required to give an output Z=1, when the input x contains the message 1010.

[IOE 2068 Chaitra]

Here,

Input X = 1 0 1 0 1 0 [Be careful to overlapping pattern]



A= Reset

B= 1 (Here 1 already occurs)

C= 10 (Here 10 already occurs)

D= 101 (Here 101 already occurs)

Fig: State diagram

Present State Y_1Y_2	*Next State Y_1Y_2*		*Output Z*	
	at X=0	at X=1	at X=0	at X=1
A	A	B	0	0
B	C	B	0	0
C	A	D	0	0
D	C	B	1	0

Fig (a): Transition table

We know the transition table for T flip-flop

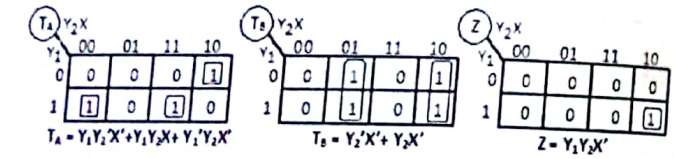
Q_t	Q_{t+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

Now assigning the values on pure binary as A = 00, B = 01 and C = 10 and then merging the above transition table (*fig a*) into a single column for both input (i.e. X=0 and X=1) as follows.

Present State Y_1Y_2	*Input X*	*Next State Y_1Y_2*	*Output Z*	T_A	T_B
00	0	00	0	0	0
00	1	01	0	0	1
01	0	10	0	1	1
01	1	01	0	0	0
10	0	00	0	1	0
10	1	11	0	0	1
11	0	10	1	0	1
11	1	01	0	1	0

***Fig:** state table*

Now using K-map,



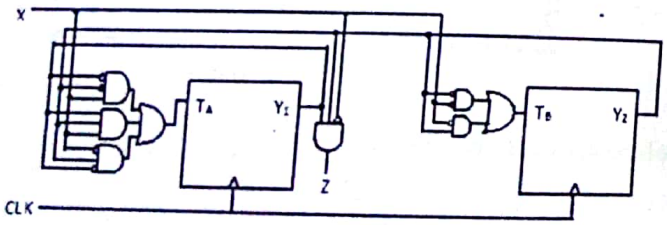


***Fig:** circuit diagram*

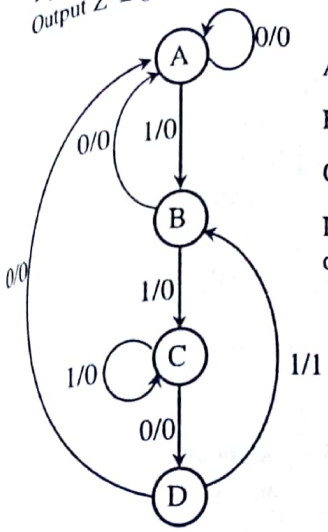
10) Design a sequential machine that has one serial input and one output Z. The machine is required to give an output Z=1, when the input x contains the message 1101.

Solution:

Input X = 1 1 0 1 1 0 1

Output Z = 0 0 0 1 0 0 1

[Be careful to overlapping pattern]



A= Reset

B= 1 (Here 1 already occurs)

C= 11 (Here 11 already occurs)

D= 110 (Here 110 already occurs)

***Fig:** State diagram*

Comparison Charts

1. Combinarional Circuit vs. Sequential Circuit

	Combinational circuit	Sequential circuit
1	Outputs can be determined using the logic function of current state input.	Outputs can be determined using the logic function of current state inputs and past state inputs
2	Don't have the capability to store a state inside them	Are capable of storing the data in a digital circuit
3	Do not contain any memory elements	Contains memory elements
4	Arithmetic and logic unit of the computers are generally comprised of combinational digital logic circuits	Used in maximum types of memory elements and also in finite state machines, which are digital circuit models with finite possible states.

5	They are fundamentally implemented using different types of devices such as multiplexers, demultiplexers, encoders, decoders, half adder, and full adders.	Latch is considered as the simplest element used to retain the earlier memory or state in the sequential digital logic
6	Don't require any feedbacks.	Utilize the feedbacks from outputs to inputs.
7	Independent of the clock.	Uses a clock for triggering the flip flops operation

2. Synchronous Sequencial Circuit vs Asynchronous Sequencial Circuit

	Synchronous sequential circuit	Asynchronous sequential circuit
1	Output changes at discrete interval of time.	Output can be changed at any instant of time by changing the input.
2	It is a circuit based on an equal state time or a state time defined by external means such as clock.	It is a circuit whose state time depends solely upon the internal logic circuit delays.
3	It is slower	It is faster as clock is not present
4	It do not suffer from Race and intermediate condition.	Suffer from Race conditions and intermediate output state.
5	Consumes more power because due to the extra circuitry required by distributing the clock to all flip-flops, and the continual switching.	Consumes less power then synchronous sequential circuit.
6	The circuit are easy to design	The circuit is quite difficult to design

Important Questions

1) Design a sequential machine that has one serial input and one output Z. The machine is required to give an output Z=1, when the input x contains the message 111.

2) A synchronous state machine has one bit serial input *X*. The output Z of machine is to be set high when the input contains the message 110. Draw the state diagram for this machine and design the circuit. (Use D flip-flop).

3) The output of the machine is to be set high when the data in the input 011 in sequence starting from MSB (use SR flip-flop).

4) Design a sequential circuit with two JK flip flops A and B and two inputs P and Q. If P=0, the circuit remains in the same state, regardless of the value of Q. When P = 1 and Q = 1, the circuit goes through the state transitions from 00 to 01 to to 10 to 11 back to 00, and repeats. When P = 1 and Q = 0, the circuit goes through the state transitions from 00 to 11 to 10 to 01 and back to 00, and repeats.

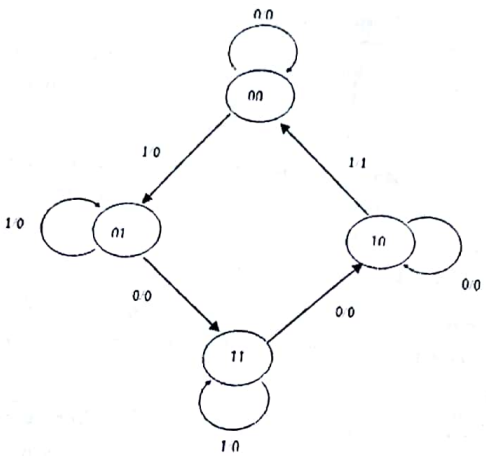
5) Suppose you have given the following word specification describing the sequential operation of some machine. The machine has the control input X and the clock and two state variable A and B and one output. If the input, is high the machine will change the state otherwise the machine is supposed to hold its present state. It also gives the output when the sequence is 101. Derive state table and state diagram. Use only T flip flop and necessary logic gates.

[IOE 2072 Kartik]

6) Design a sequential machine that detects the consecutive zeros from the input data stream X by making output, Y = 1.

[IOE 2069 Chaitra]

7) Design a sequential circuit corresponding to the given state diagram using S-R flipflop. [PoU Fall 2014]



8) Different between sequential and combinational circuit in content of digital circuit. [PU 2005]

❖❖❖

8

Chapter 10 Digital Integrated Circuits

TTL (transistor - transistor logic) was invented in 1961 by James L Buie. TTL as outlined above stands for transistor - transistor logic. It is a logic family implemented with bipolar process technology that combines or integrates NPN transistors, PN junction diodes and diffused resistors in a single monolithic structure to get the desired logic function. The NAND gate is the basic building block of this logic family. Different subfamilies in this logic family, as outlined earlier, include standard TTL, low-power TTL, high-power TTL, low-power Schottky TTL, Schottky TTL, advanced low-power Schottky TTL, advanced Schottky TTL and fast TTL.

10.1 Switching circuit

Diodes Bipolar junction transistor and metal-oxide-semiconductor transistor (MOSFETs) are the semiconductor device used in digital integrated circuits(ICS).The most popular transistor-transistor logic(TTL) in use includes the 7400 and the 74LS00 families; resistors, diodes and BJTS are the elements use to construct those circuit 74L00and 74HL00 are the most widely used families constructed using MOSFETs. Silicon oxide or transistor is used as electronic switch.

The semiconductor diode:

The diode behaves like as a one-way switch. That is, it will allow electronic circuit in one direction but not the other.

BJTs:

The BJTs behaves as an electronic switch. The switch is activated by applying a voltage between base and emitter. It works like below:

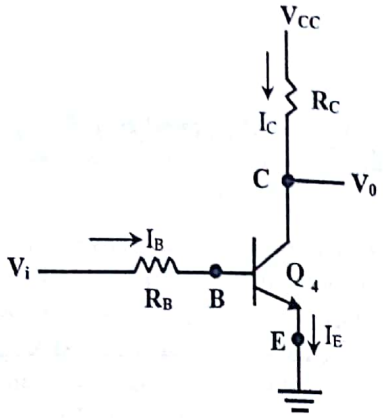


Fig: common emitter BJT

The current I_C flows through resister R_C and the collector of the transistor. Current I_B flows through resistor R_B and the base of the transistor. The emitter is connected to ground and $I_E = I_B + I_C$. The supply voltage between V_{CC} and ground. The input between V_i and ground and the output is between V_0 and ground.

Case I

If V_{BE} is less than 0.7V, the transistor is cut off with $I_B = 0$. The collector to emitter circuit when behaves like an open circuit with $I_C = 0$ and drop R_c is 0 and $V_0 = V_{CC}$.

Case II

If V_{BE} is greater than 0.7V the transistor is biased with collector to emitter voltage $V_{CE} = 0$ this leads $I_C = V_{CC}/R_C$ so, $V_0 = 0$V.This illustrates the BJT as a switch.

Q) Explain the use of BJT as a switch. *[IOE 2068 shrawan]*

C) MOSFET:-

It behaves as the electronic switch. The switch is activated by applying a voltage between gate and source. The voltage between gate and source is zero. The switch is open and no current is

allowed between source and drain. The transistor is off. The voltage is applied between gate and source. The switch is closed and a current is allowed between source and a current is allowed between source and drain. The transistor is on.

10.2 7400 Transistor-transistor logic (TTL)

The below figure show a TTL NAND gate. Here each emitter act like a diode; therefore Q_1 and the 4 KΩ resistor act like a 2-input AND gate.The rest of circuit invert the signal so that the overall circuit act like a 2-input NAND gate. The output transistor (Q_3 & Q_4) form a **totem-pole connection** (one npn in series with another). With a totem - pole output stage, either the upper or lower transistor is on. When Q_3 is on, the output is high; when Q_4 is on the output is low.

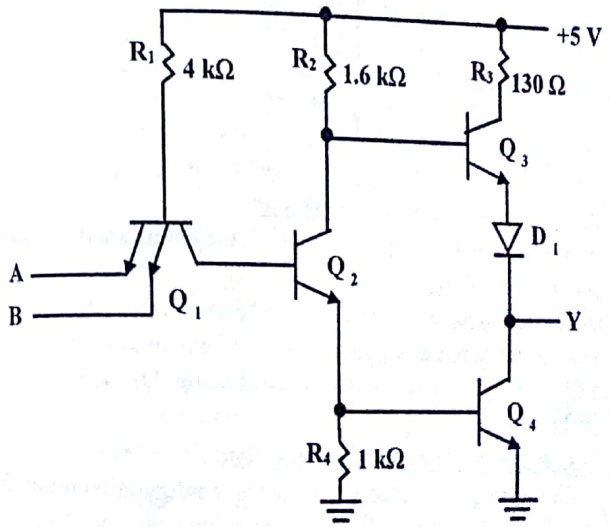


Fig: Two-input TTL NAND gate

The circuit operates as follows. Transistor Q_1 is a two-emitter NPN transistor, which is equivalent to two NPN transistors with their base and emitter terminals tied together. When both the inputs are in the logic HIGH state as specified by the TTL family the current flows through the base-collector PN junction diode of transistor Q_1

into the base of transistor Q_2. Transistor Q_2 is turned ON to saturation, with the result that transistor Q_3 is switched OFF and transistor Q_4 is switched ON. This produces a logic LOW at the output. Transistor Q_4 is also referred to as the current-sinking or pull-down transistor, for obvious reasons. Diode D_1 is used to prevent transistor Q_3 from conducting even a small amount of current when the output is LOW. When the output is LOW, Q_4 is in saturation and Q_3 will conduct slightly in the absence of D_1.

When either of the two inputs or both inputs are in the logic LOW state, the base-emitter region of Q_1 conducts current, driving Q_2 to cut-off in the process. When Q_2 is in the cut-off state, Q_3 is driven to conduction and Q_4 to cut-off. This produces a logic HIGH output. Transistor Q_3 is also referred to as the current-sourcing or pull-up transistor.

The input and output condition of TTL circuit is:

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

Advantage of totem-pole connection:

It offers low-output impedance in both the HIGH and LOW output states.

When the output is in the logic LOW state, transistor Q_4 would need to conduct a fairly large current if its collector were tied to VCC through R_3 only. A non-conducting Q_3 overcomes this problem.

Disadvantage of totem-pole connection:

A disadvantage of the totem-pole output configuration results from the switch-off action of Q_4 being slower than the switch-on action of Q_3. On account of this, there will be a small fraction of time, of the order of a few nanoseconds, when both the transistors are conducting, thus drawing heavy current from the supply.

Q) What do you mean by the totem-pole output?

[IOE 2069 Chaitra]

Characteristics

Power dissipation: The standard TTL gate has a power dissipation of about 10 milli watt (mw). It may vary from this value because of signal, level, tolerance etc.

Propagation delay: It is the time that a TTL takes from the output of gate to change after the input have changed. The propagation delay time of TTL gate is approximately 10 ns.

Temperature: TTL of 7400 series work over a temperature range of 0 to 70 °C and voltage range of 4.75 to 5.25 v.

10.3 TTL parameters

Floating inputs:

When the TTL input is high (ideally +5 volt), the emitter current is approximately 0. When the TTL input is floating i.e. unconnected to emitter current is possible because of the open circuit. Therefore, a floating TTL input is equivalent to a high output. So unused TTL input left unconnected an open input allows to the rest of the gate to function properly. So we have to connect unused TTL input to a supply voltage.

Sourcing and sinking:

TTL acts as the current source when output is high and acts as sink when output is low. When the standard TTL output is high, a reverse emitter current of 40 μA exist in the direction shown.

Worst case input and output voltage:

In TTL the worst case input voltage are

$$V_{il}(\max)=0.8 \text{ v and } V_{IH}(\max) = 2 \text{ v}$$

The worst case output voltage are

$$V_{OL}(\max)= 0.4v \text{ and } V_{OH}(\min) = 2.4v$$

Fan out:

Fan out of TTL output is the number of inputs it can be fed or connect to. It is used to form a complex circuit. The maximum fan out of an output measures its load driving capacity; it is greatest number of inputs of the same types to which the output can be safely connected. In

TTL fan out is 10 because one TTL driver can drive 10 TTL loads.

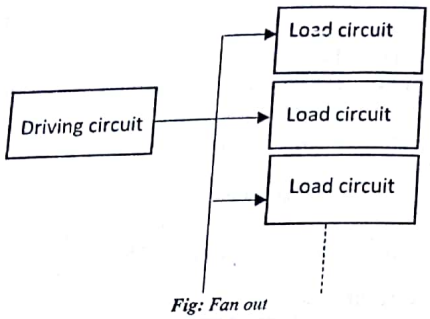


Fig: Fan out

10.4 TTL overview

NAND gates:

It is the backbone of 7400 series. All device in this series are derived from the two input NAND gate.

NOR gate:

We can get other logic function by modifying basic NAND gate design. For obtaining NOR gate Q_5 and Q_6 are added in the figure. Here Q_2 and Q_6 are in parallel, we get the OR function which is followed by inversion to get the NOR function.

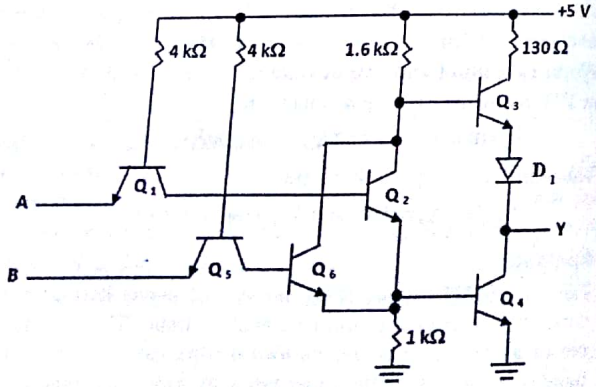


Fig: TTL NOR gate

When A and B are both low, the base of Q_1 and Q_5 are pulled low; this cuts off Q_2 and Q_6. Then Q_3 acts as an emitter follower and we get a high output. If A and B is high Q_1 and Q_5 are cut off forcing Q_2 or Q_6 to turn on. When this happens Q_4 saturates and pull the output down to the low voltage.

Q) ***Draw the schematic diagram of TTL NOR gate. Discuss the characteristics of TTL 74xx series gates. [IOE 2068 Baisakh]***

AND OR INVERT GATES:

Here Q_1, Q_2, Q_3 and Q_4 from the basic 2-input NAND gates. By adding Q_5 and Q_6 we convert the basic NAND gate to an AND – OR-INVERT gate. Both Q_1 and Q_5 act as 2 input and gates; Q_2 and Q_4 produce OR ing and inversion.

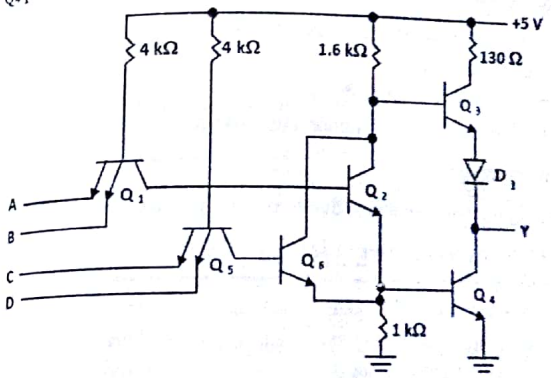


Fig: AND- OR inverter

10.5 Open collector gates

The below figure a) is a 2-input NAND gate with an open collector output i.e. Q_4 is called as open collector gate. Because of the collector of Q_4 is open, a gate like this will not work properly until you connect an external pull up resistor like in figure b) The output of open collector gate can be wired together and connected to a common pull of resistor. A connection like this has the advantage of combining the output of three devices without using final OR gate (or AND gate). Here figure c) shows three TTL devices connected to the pull up resistor

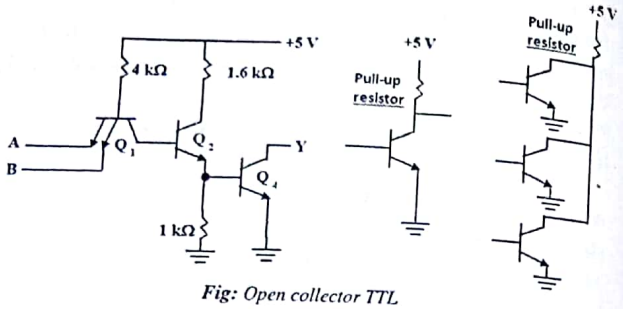


Fig: Open collector TTL

Application:

They are used in three major application they are:
Driving a lamp or relay
Performing a wired logic
For the construction of common bus system

Disadvantage:

It has slow switching speed due to pull up resistor.

10.6 Three state TTL devices

The output of two TTL gate with totem pole structure cannot be connected together as in open collector output. There is however a special type of totem pole device that allows wired connection of output for the purpose of performing a common bus system. When the totem pole output TTL gate has this property it is called a three state gate. Three state gate eliminate the need of open collector gate in bus configuration.

A three state gate in bus configuration exhibits three output state:

A low level state when the lower transistor in the totem pole is on and upper transistor is off

A high level state when the upper transistor in the totem pole is on and the lower transistor is off.

A third state when both transistor in totem pole are off.

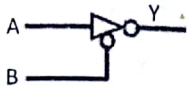
The three state TTL, a modified TTL design that allows us to connect output directly.

Three state buffer:

The below fig is three state buffer gates. When the control input C is high the gate is enabled and behaves like as a normal buffer gates with the output equal to input binary value. When the control input is low the output is an open circuit which gives a high impedance. Three state gate produce a high impedance state when the control input is low.

Fig: three state buffer gate

Here fig below, is three state inverter gate, when C is low it gives output equal to invert of input binary value. When C is high it gives high impedance state.



Common bus:

The complex system like as microcomputer and microprocessor, a number of gate output may be required to connect in a common line which is referred as " bus" which in turn may be required to drive a number of gate input.

Some difficulties when a number of gates output are connected to bus are:

The problem of loading and operation speed is encountered when open collector output is connected to form a bus.

Totem - pole output cannot be connected together due to the very large current drain from the supply and consequent heating of IC's which may get damaged.

To overcome these difficulties, special circuit has been developed in which there is one more output referred to as third state or high impedance state. In addition to low and high state such circuit are called tri state or three state logic.

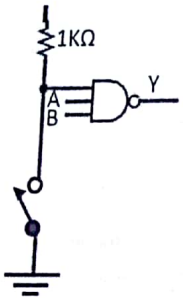
Q) What do you mean by the term 'bus' as used in digital technology? Describe the factor to be considered while designing the bus. *[IOE 2065 Shrawan]*

10.7 External drive to TTL load

To drive TTL load with an external source, you need to satisfy the TTL input requirement for voltage and current. For standard TTL in low state this means an input voltage between 0 & 0.8volt with the current of approximately 1.6mA. In high state the voltage has to be formed 2 to 5volt with the current of approximately 40μA some of the ways to drive TTL load are:

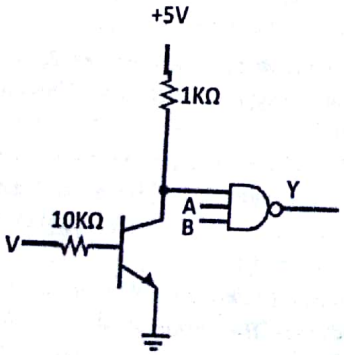
Switch drive:

Fig below shows the preferred method for driving a TTL input from a switch. With the switch open the input is pulled up to +5v when the switch is closed the input is pulled down to the ground.



Transistor drive:

Fig below shows way to drive a TTL. Here we are using a transistor switch to control the state of TTL input.

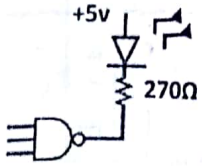


10.8 TTL driving external load

Because standard TTL can sink up to 16mA, we can use a TTL driver to control an external load as relay, LED etc.

Driving an LED

In fig below a TTL circuit drives an LED. When the TTL output is high, the LED is dark. When the TTL output is low the LED lights up.



10.9 CMOS logic family

The CMOS (Complementary Metal Oxide Semiconductor) logic family uses both N-type and P-type MOSFETs to realize different logic functions. The two types of MOSFET are designed to have matching characteristics. That is, they exhibit identical characteristics in switch-OFF and switch-ON conditions. The main advantage of the CMOS logic family over bipolar logic families discussed so far lies in its extremely low power dissipation, which is near-zero in static conditions. In fact, CMOS devices draw power only when they are switching. CMOS technology today is the dominant semiconductor technology used for making microprocessors, memory devices and application-specific integrated circuits (ASICs). The CMOS logic family, like TTL, has a large number of subfamilies.

74C00 CMOS

NAND gates:

The fig below is a CMOS NAND gate. Q_1 and Q_2 form one complementary connection and Q_3 and Q_4 form other. The low A input will close Q_1 and open Q_2; a high A input will open Q_1 and close Q_2. Similarly a low B input will open Q_3 and close Q_4; a high B input will close Q_3 and open Q_4.

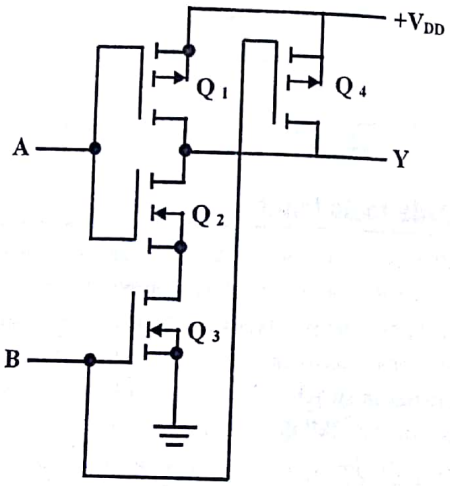


Fig: CMOS NAND gate

Q) Draw the schematic circuit for CMOS NAND gates.

[IOE 2069 Chaitra]

NOR gates: Figure below is a CMOS NOR gate. Here the output goes high when Q_1 and Q_2 are closed. The output goes low if either Q_3 or Q_4 is closed.

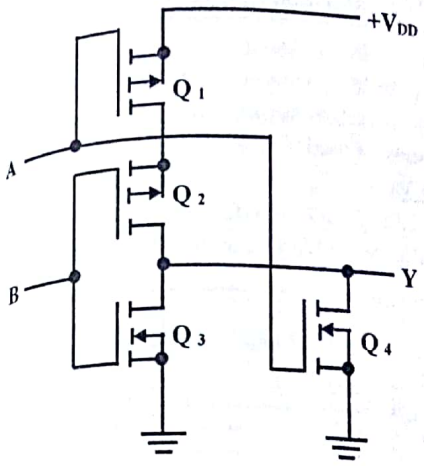


Fig: CMOS NOR gate

10.10 CMOS Characteristics

a) **Propagation delay:** The standard CMOS has a propagation delay time of approximately 25 ns to 100 ns, with the exact value depends upon the power system.

b) **Power dissipation:** The static power dissipation of a device is its average power dissipation when the output is constant the power dissipation is approximately 10 Nano watt (nW) per gate.

c) **Floating input:** It is equivalent to high input. It is better to connect unused TTL input to a supply voltage.

d) **Easily damaged:** Because of thin layer of silicon dioxide between the gate and the substrate CMOS device have a very high resistance, approximately infinite. The insulating layer is kept as thin as possible to give the gate more control over the drain current. Because this layer is so thin, it is easily destroyed by excessive gate voltage.

e) **Fan out:** CMOS device can drive up to 10 CMOS load. So, the fan out of CMOS is 10.

10.11 TTL to CMOS interface

Interface is the way a driving device is connected to loading device. We know that the TTL deice need a supply voltage of +5v, while CMOS device can use any supply requirement differ, several interfacing schemes may be used.

Supply Voltage at 5 V

One approaching to TTL/ CMOS interfacing is to use +5V as the supply voltage for both the TTL driver & the CMOS load.

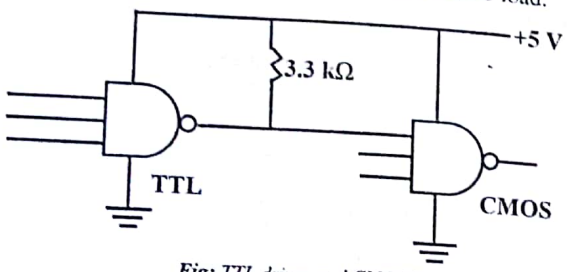
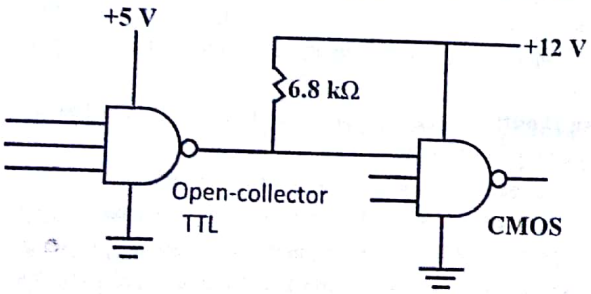


Fig: TTL driver and CMOS load

Different supply voltage

At lower voltage CMOS performance deteriorates because the propagation delay time increases & the noise immunity decreases. So, CMOS devices now has a supply voltage of +12 V.



Open-collector TTL driver allows higher CMOS supply voltage

Q) Explain the operation of TTL to CMOS interface.

[IOE 2068 Chaitra]

10.12 CMOS-to-TTL interface

Here driving device is CMOS & loading device is TTL. We have to make sure that the CMOS low—state output is always less than 0.8 V, the maximum allowable TTL low- state input voltage. Also the CMOS high- state output must always be greater than 2 V, the minimum allowable TTL high state input voltage.

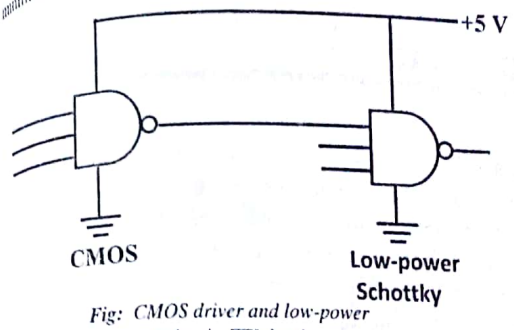


Fig: CMOS driver and low-power Schottky TTL load

The figure above is one of the approach to use +5V as the supply voltage for the driver and the load.

Important Questions

1) What are the characteristics of TTL circuit for logic high and low level? Explain the operation of TTL NAND gate.
2) Draw the schematic diagram of TTL NAND gate explain about the CMOS characteristics. [IOE 2069 Ashad]
3) Explain the use of BJT as a switch. Draw the schematic diagram of TTL NAND gate. [IOE 2068 Shrawan]
4) What do you understand by the term 'common bus'? Explain the construction of common bus using tristate buffer with a clear diagram. What issue should be considered when design a bus with tristate buffer? [IOE 2066 Bhadra]
5) What are the parameter of TTL? Explain the operation of 74C00 CMOS. [IOE 2070 Ashad]
6) Draw the schematic diagram of TTL two input NOR gate. [IOE 2070 Chaitra]

Chapter 11

Applications

Digital electronics is crucial in this current world. Because, without digital electronics it's not possible to speed up our work. You can take a look of previous analog devices- those are very slow and takes lots of time to find out the result. Digital electronics or digital circuits are electronics that handle digital signals- discrete bands of analog levels, rather than by continuous ranges (as used in analogue electronics). All levels within a band of values represent the same numeric value. Because of this discretization, relatively small changes to the analog signal levels due to manufacturing tolerance, signal attenuation or parasitic noise do not leave the discrete envelope, and as a result are ignored by signal state sensing circuitry. Digital techniques are useful because it is easier to get an electronic device to switch into one of a number of known states than to accurately reproduce a continuous range of values. Digital electronic circuits are usually made from large assemblies of logic gates, simple electronic representations of Boolean logic functions. Digital electronics involves in logic of 0 and 1. Where 0 means Off and 1 means On. It involves the devices and circuits such as gates, flip-flops, latches etc. The basic idea of Digital electronics is mandatory for microprocessor & micro controller and also for computer architecture. So, any application which involves microprocessor or a computer have digital electronics in it. It can be security applications, error correcting & error detecting, power saving circuits etc.

11.1 Multiplexing Displays

A large number of embedded systems offer some form of display device to convey information to the user. The display can consist of anything form of light indicating that power is on, to a complex graphical display showing a representation of the process. Simple control systems can be equipped with complex displays while more complex systems can offer limited information to its user; there are no set rules as to how much information has to be displayed or how it has to be presented. The world of information display is becoming extremely complex, especially when you consider new technologies such as virtual reality.

Interfacing Seven-Segment Displays

Seven-segment displays commonly contain LED segments arranged as a figure-of-eight pattern, with one common lead (anode or cathode) and seven individual leads for each segment. When the common lead is the anode it is referred to as the common anode (CA), and when the common lead is the cathode it is referred to as the common cathode (CC). Figure (a) shows one of the possible configurations of interfacing a CC display with the microcontroller. The IC CD4511 is a BCD to seven-segment decoder/driver. The microcontroller feeds the BCD equivalent of the digit to be displayed to the 4511 IC.

Seven-segment displays can also be connected directly without the use of a BCD to seven-segment decoder. In this case the seven-segment code of the digit is generated by the microcontroller program itself. Figure (b) shows the direct circuit connection for CA display.

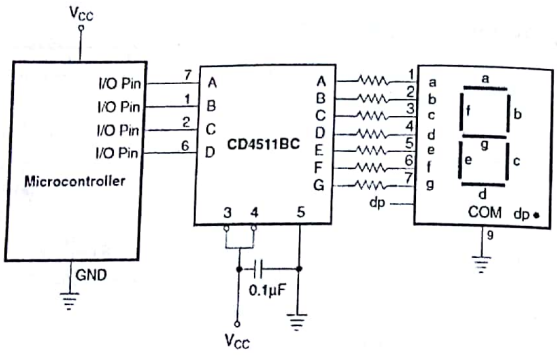


Fig: a) Possible configurations of interfacing a CC display with a microcontroller

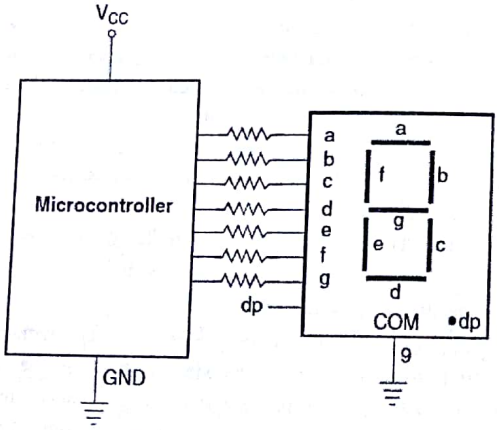


Fig: b) Direct circuit connection for CA display

If more than one display is to be used, the displays are time multiplexed. The human eye cannot detect the blinking display if each display is relit every 10 ms or so. The 10 ms time is divided by the number of displays used to find the interval between updating each display. In the case of CC displays the display is selected by driving the common cathode to logic LOW, and in the case of CA displays the display is selected by driving the common

anode to logic HIGH. Figure (c) shows the multiplexed circuit for two CC displays. The IC 74138 is a 3-to-8 line decoder used for selecting the display. Figure (d) shows the multiplexing in the case of CA displays for direct connection without the use of a BCD to seven-segment driver.

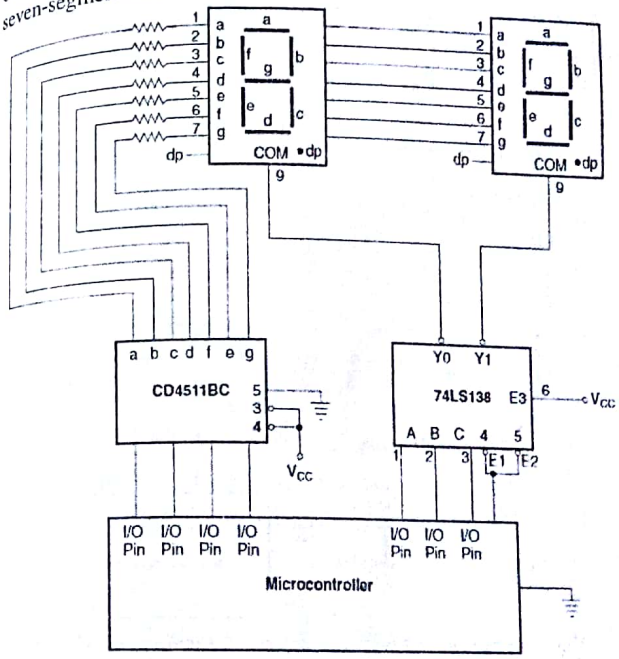


Fig: c) Multiplexed circuit for two CC displays

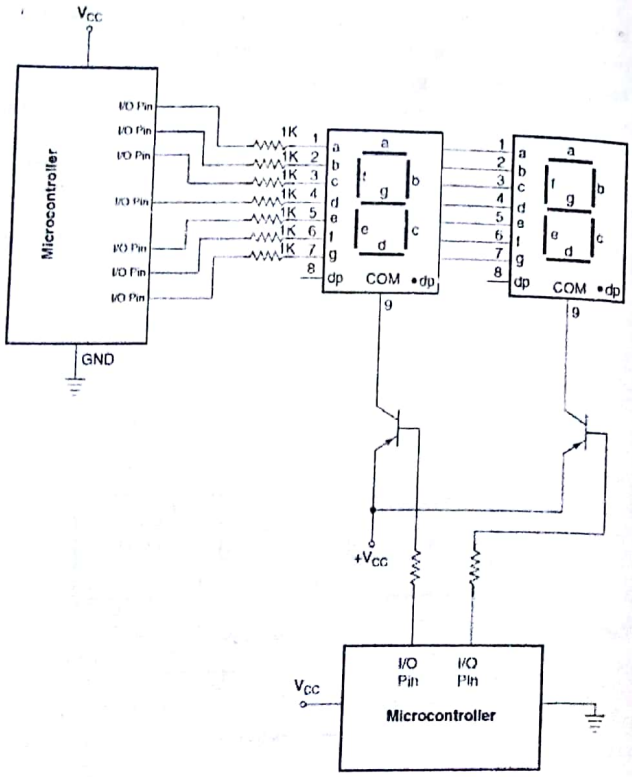


Fig: d) Multiplexed circuit for two CA displays for direct connection

11.2 Frequency Counters

A frequency counter is an electronic instrument, or component of one, that is used for measuring frequency. Frequency counters usually measure the number of oscillations or pulses per second in a repetitive electronic signal.

Figure below shows the architecture of a frequency counter when it is being used in the frequency measurement mode. The oscillator section, comprising a crystal-based oscillator and a frequency divider chain, generates the clock pulses. The clock pulses are used

to trigger a flip-flop whose output serves to enable or disable the AND gate. When the AND gate is enabled, the input signal, after passing through the signal conditioning section comprising level shifting amplifiers, comparators, etc., reaches the counter. In the simplest case, if the AND gate is enabled for 1 s (which is the case when the flip-flop clock input is 1 Hz), then the counter count will represent the signal frequency. The measurement resolution in this case would be 1 Hz. The measurement resolution can be improved by enabling the AND gate for a longer time.

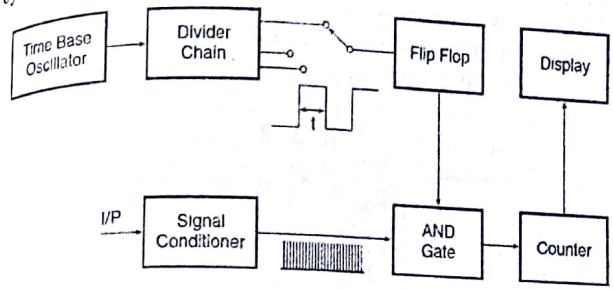


Fig: Block diagram of frequency counter

Basics of using a frequency counter

In many ways using a frequency counter is very simple. It is a matter of turning the counter on and applying the signal to the input.

Frequency counters and timers can be used for measuring many signals from digital logic signals through to RF and microwaves. Counter technology has developed so that these frequency counters

and timers are able to measure time intervals as well as frequency. Using digital processing technology it is possible to measure both time interval and frequency as one is the inverse of the other. To make a simple measurement of frequency or time interval it is necessary to apply the signal to the input.

When using a frequency counter, it is necessary to select the time base interval. Typically options of 0.1s, 1s, 10s are the most common. These refer to the length of time over which the frequency counter gate is open and incoming pulses are counted. Thus, for a gate time of 1 second, 1 000 000 pulses will be counted for a 1 MHz signal, or in the simple example below, if five pulses are counted, with a gate time of a second, then the frequency is 5Hz.

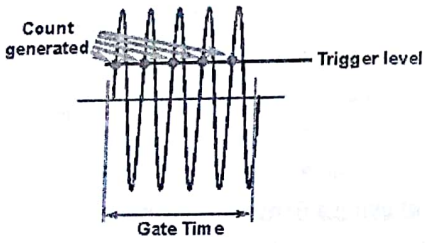


Fig: Frequency counter counting

Here it can be seen that each time the signal passes through the frequency counter trigger level in the positive direction, the counter counts the transition. The higher the frequency, the more pulses that are counted. As seen five pulses are counted. Assuming a gate time of 1 second, this would indicate a frequency of 5Hz. Had the gate time been 0.1 s then it would indicate a frequency of 50 Hz.

Longer gate times will naturally give more accurate results. Take the example of a gate time of 0.1 s and five pulses counted indicating a 50 Hz signal. Cutting the explanation to its simplest, the system cannot differentiate between a signal that is running at 50 Hz and one that is at 55 Hz for example. Both will give five counts in a time of 0.1s. To give a better indication of the frequency, a 1 second gate time will enable 55 counts if the signal is running at 55 Hz for example.

Although longer gate times give better levels of accuracy, the choice of gate time is normally more dependent upon the need for rapid updates in the frequency. A longer gate time, for example with a slow and automated test, or for the engineer working on the bench the additional waiting time can be annoying. Accordingly short gate times are normally used unless high levels of accuracy are needed.

11.3 Time measurement

The frequency counter building blocks, when slightly rearranged as shown in figure below, can be used to measure the time period. Enabling and disabling of the AND gate are now determined by the frequency of the input signal and not by the clock frequency. The number stored in the counter here is proportional to the number of clock pulses that reach the counter during the period of the input signal. The same set-up can be used for time interval measurement by having two input signal channels, with one enabling the AND gate by, say, setting the flip-flop and the other disabling the same by resetting the flip-flop.

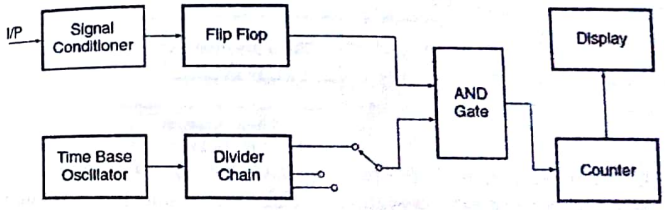
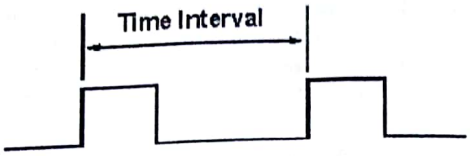


Fig: Time period measurement using frequency counter

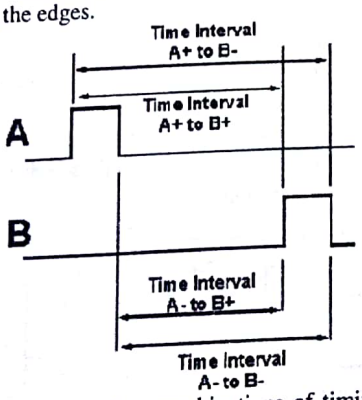
Single input time interval measurements

The basic form of counter timer is one with a single input. Typically any measurements will be made on the rising edge of a waveform. The basic measurement will be made from the rising edge of the waveform to the next rising edge that is seen. This gives the time interval for the waveform. On occasions it may be possible to make a measurement using a single input counter from the rising or positive edge to the falling or negative edge. Switches on the front panel or remote programming techniques are used to select the required edges.



Dual input time interval measurements

Some counter timers that can be used have dual inputs. This form of time interval measurement can be used with two waveforms and it is normal for there to be selectable positive (rising) or negative going (falling) edge selection on both channels. To set this up, it is simply a matter of connecting the required waveforms to the inputs and then selecting the edges.



It can be seen that there are four combinations of timing. In the diagram, A+ indicates waveform A rising edge, and B- for example means waveform B falling edge.

Important Questions

1) What is frequency counter? Explain with block diagram. [IOE 2069 Ashad]
2) Describe the block diagram of the instrument to measure time period.
3) What is multiplexing displays? Describe briefly.

❁❁❁

Chapter 12

VHDL

12.1 Introduction

VHDL is a hardware description language used to describe the behavior and structure of digital system. VHDL stands for VHSIC hardware description language and VHSIC in turns stands for very high speed integrated circuit. However VHDL is a general purpose hardware description language that can be used to describe and simulate the operation of the wide variety of the digital system, ranging in complexity from a few gates to an interconnection of many complex integrated circuit. VHDL was originally developed for the military to allow a uniform method for specifying digital system. The VHDL language has since become an IEEE standard and it is widely used in industry.

VHDL can describe a digital system at a several different levels behavioral, data flow and structural. For example a binary adder could describe at its behavioral level in term of its function of adding two binary numbers without giving any implementation details. The same adder could be described at the data flow level by giving the logical equation for the adder. Finally the adder could be described at the structure level by specifying the inter connection of the gates that comprise the adder. If a design is described in VHDL and implemented in today's technology the same VHDL description could be used as a starting point for a design in some future.

Hardware description languages that have evolved over the years include ABEL-HDL, VHDL (VHSIC HDL), Verilog and JHDL (Java HDL). VHSIC stands for Very High-Speed Integrated Circuit. The second type of software program is a computer program, called a logic compiler that is used to transform a source code written in HDL into a bit stream. Logic compilers are available from

Operation category	Operation symbol	Operation performed
logical	AND	AND
	OR	OR
	NAND	Not AND
	NOR	Not OR
	XOR	XOR
	XNOR	Not XNOR
	NOT	NOT
rational	=	Equality
	\=	Inequality
	>	Greater then
	<	Less then
	>=	Greater than or equal to
	<=	Less than or equal to
arithmetic	+	Addition
	-	Subtraction
	*	Multiplication
	/	division
concatenation	&	concatenation
Shift and rotate	SLL	Shift left logic
	SRL	Shift right logic
	SLA	Shift left arithmetic
	SRA	Shift right arithmetic
	ROL	Rotate left
	ROR	Rotate right

12.2 Writing simple VHDL code

Let's us suppose an example to illustrate how to write a simple VHDL source code. Consider a logic circuit as below.

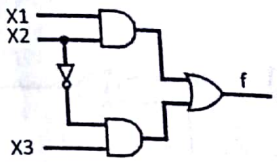


Fig: a simple logic function

Code:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY Example IS
PORT(X1, X2, X3: IN BIT ;
     f          : OUT BIT);
END Example;
ARCHITECTURE LogicFunc OF example IS
BEGIN
f <= (X1 AND X2) OR (NOT X2 AND X3);
END LogicFunc;
```

In above code first of all we have to declare input and output signal. This is done using a construct called an entity. Entity must assigned a name; we have chosen the name 'example' for this. The input and output signals for the entity are called its PORTS. Each PORT an associated mode that specify whether it is an input (IN) to the entity or an output (OUT) from the entity The entity 'example' has four parts in total. The first three X1, X2, and X3 are inputs signal of type BIT. The port named f is an output of type BIT.The entity specifies the input and output signals for a circuit, but it does not give any details as to what the circuit represent. The circuits functionality must be specifies with a VHDL construct called an architecture. It must be given a name, and we have chosen the name Logic Func. Although the name can be aby text string, it is sensible to assign a name that is meaningful to the designer.

VHDL has built in support for the Boolean operators AND, NOT, OR etc. and signal assignment operator <=.

Full adder:

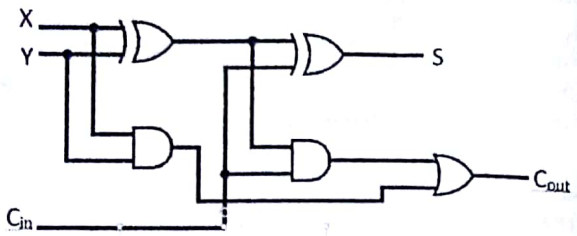


Fig: full adder

Code:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY full add IS
PORT(X, Y, Cin : IN STD_LOGIC;
     S, Cout   : OUT STD_LOGIC);
END full add;
ARCHITECTURE Logicfunc OF full add IS
BEGIN
S<=X XOR Y XOR Cin ;
Cout<=(X AND Y) OR (Cin AND X) OR (Cin AND Y);
END Logic func
```

Fig: full adder

Q) Design full adder circuit using VHDL.

[BE IOE 2069 ashad]

Multiplexer:

A multiplexers is a circuit having number of data inputs, one or many select inputs and one output. It passes the single value on one of the data inputs to the output. The data inputs are selected by the values of the select inputs.

2 to 1 mux:

D_0	D_1	S	Y
1	0	0	1
0	1	1	1

$Y = S'W_0 + SW_1$

CODE:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY mux 2 to 1 IS
PORT(D0, D1, S: IN STD_LOGIC;
     f        : OUT STD_LOGIC);
END mux 2 to 1;

ARCHITECTURE Behavior OF mux 2 to 1 IS
BEGIN
WITH S SELECT
F<= D0 WHEN '0',
    D1 WHEN OTHERS;
END Behavior;
```

4 to 1 mux:

Data select input		Input select	output
S_0	S_1	Y	
0	0	D_0	$\overline{S_0}\,\overline{S_1}D_0$
0	1	D_1	$\overline{S_0}S_1D_1$
1	0	D_2	$S_0\overline{S_1}D_2$
1	1	D_3	$S_0S_1D_3$

CODE:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY mux 4 to 1 IS
PORT (D0, D1, D2, D3: IN STD_LOGIC;
      S             : IN STD_LOGIC VECTOR (1 DOWN TO 0);
      F             : OUT STD_LOGIC);
END mux 4 to 1;

ARCHITECTURE behavior OF mux 4 to 1 IS
```

```
BEGIN
WITH S SELECT
F<= D0 WHEN '00',
    D1 WHEN '01',
    D2 WHEN '10',
    D3 WHEN OTHERS;
END Behavior;
```

Priority encoder:

CODE:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY priority IS
PORT (w: IN STD_LOGIC_ VECTOR(3 DOWN TO 0);
      y: OUT STD_LOGIC_VECTOR(1 DOWN TO 0);
      z: OUT STD_LOGIC);
END priority;

ARCHITECTURE Behavior OF priority IS
BEGIN
WITH S SELECT
y<= "11"WHEN w(3)='1'ELSE
    "10"WHEN w(2)='1'ELSE
    "01"WHEN w(1)='1'ELSE
    "00";
Z<='0'WHEN w="0000"ELSE'1'
END Behavior;
```

Gated D latch:

The code below defines an entity named latch, which has the input D and clk and the output Q. the process uses an if- then-else statement to define the value of Q output. When clk=1, Q takes the value of D. For the case when clk is not 1, the code doesnot specify what value Q should have. Hence Q will returns its current value in this case and the code describe the gated D latch. The process sensitivity lists include both clk and D because these signals a change in the value of the Q output.

CODE:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY latch IS
PORT(D, clk : IN STD_LOGIC;
     Q      : OUT STD_LOGIC);
END latch;
ARCHITECTURE Behavior OF latch IS
BEGIN
PROCESS (D, clk)
BEGIN
   IF CLK ='1' THEN
      Q<=D;
END IF;
END PROCESS;
END Behavior;
```

D – flip flop:

The code is similar to the D – latch with the two exception. Firstly the process sensitivity list contains only the clock signal because it is the only signal that can cause a change in the Q output. Second, the if-then-else statement used a different condition from the one used in latch. Here combining the Clock'EVENT condition with a condition CLOCK=1 means that "the value of clock signal has just changed, and the value is now equal to 1". Hence the condition refers to a positive clock edge. The below code describe a positive edge triggered D flip flop.

CODE:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY flip flop IS
PORT (D, Clock : IN STD_LOGIC;
      Q        : OUT STD_LOGIC);
END flip flop;
```

```
ARCHITECTURE Behavior OF flip flop IS
BEGIN
PROCESS (Clock)
BEGIN
    IF Clock'EVENT AND Clock = '1' THEN
     Q<=D;
END IF;
END PROCESS;
END Behavior;
```

Four bit shift register:

Here the line code are numbered for ease of reference. Instead of using subcircuit, the shift register is described using a sequential statement. Due to the WAIT UNTIL statement in line 13, any signal that is assigned a value inside the process has to be implemented as the output of flip flop. Line 14 and 15 specifies the parallel loading of the shift register when L = 1. The ELSE clause in the line 16 to 20 specifies the shifting operation. Line 17 shifts the value of Q_1 in to the flip flop with the output Q_0. Line 18 and 19 shifts the value of Q_2 and Q_3 in to the flip flop with the output Q_1 and Q_2 respectively. Finally line 20 shifts the value of w in to the left most flip flop, which has the output Q_3. Hence all four flip flop changes their value at the same time, as required in the shift register.

CODE:

```
1.  LIBRARY ieee;
2.  USE ieee.std_logic_1164.all;
3.  ENTITY shift4 IS
4.  PORT (R          : IN     STD_LOGIC_VECTOR(3
    DOWNTO 0);
5.  Clock : IN   STD_LOGIC ;
6.  L, w  : IN   STD_LOGIC ;
7.  Q  : BUFFER STD_LOGIC_VECTOR(3 DOWNTO 0));
8.  END shift4;
9.  ARCHITECTURE Behavior OF shift4 IS
10. BEGIN
```



```
11. PROCESS
12. BEGIN
13.  WAIT UNTIL Clock'EVENT AND Clock = '1'
14. IF L= '1' THEN
15. Q<=R;
16. ELSE
17. Q(0)<= Q(1);
18. Q(1)<= Q(2);
19. Q(2)<= Q(3);
20. Q(3)<= w;
21. END IF;
22. END PROCESS;
23. END Behavior;
```

Up – counter:

Here four bit counter has a reset input, Resetn and an enable input E. In the architecture body the flip flop in the counter are represented by the signal name *Count*. The process statement specifies an asynchronous reset of *count* if Resetn =0. The ELSIF clause specifies that on the positive clock edge, if E= 1, the count is incremented. If E=0, the code explicitly assigns count<=*count*.

CODE:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY upcount IS
PORT (Clock, Resetn, E      : IN   STD_LOGIC;
      Q                     : OUT STD_LOGIC_VECTOR(3
DOWNTO 0));
END upcount;

ARCHITECTURE Behavior OF upcount IS
SIGNAL Count : STD_LOGIC_VECTOR(3 DOWNTO 0));
BEGIN
PROCESS (Clock, Resetn)
BEGAIN
```

```
IF Resetn='0' THEN
Count<="0000";
ELSEIF (Clock'EVENT AND Clock = '1') THEN
IF E = '1' THEN
Count <=Count +1
ELSE
    Count <= Count;
END IF;
END IF;
END PROCESS;
Q<=Count;
END Behavior;
```

Down - counter:

The below code is for down counter named downcnt. To make it easy to change the starting count, it is defined as a GENERIC parameter named modulus. On the positive clock edge, if L=1, the counter is loaded with the value modulus-1, and if L=0, the count is decremented. The counter also includes an enable input E. Setting E=1 allows the count to be decremented when an active clock edge occurs.

CODE:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY downcnt IS
GENERIC (modulus: INTEGER:=8);
PORT (Clock, L, E : IN     STD_LOGIC;
      Q           : OUT INTEGER RANGE 0 TO modulus - 1);
END downcount;

ARCHITECTURE Behavior OF downcount IS
SIGNAL Count : INTEGER RANGE 0 TO modulus - 1;
BEGIN
PROCESS
BEGAIN
   WAIT UNTIL (Clock'EVENT AND Clock = '1');
```



```
   IF L = '1' THEN
   Count <=modulus-1;
ELSE
   IF E = '1' THEN
   Count <=count -1;
END IF;
END IF;
END PROCESS;
Q<=Count;
END Behavior;
```

12.3 Common error in VHDL code

Some error that student have made when writing VHDL code.

a) **ENTITY and ARCHITECTURE name:**
The name used in an ENTITY declaration and the corresponding ARCHITECTURE must be identical

Code:

```
ENTITY adder IS
.
.
END adder;
ARCHITECTURE structure OF adder4 IS
.
.
END structure;
```

Is an erroneous because the ENTITY declaration use the name adder, whereas the ARCHITECTURE uses the name adder 4.

b) **Missing semicolon:**
Every VHDL statement must ne end with a semicolon.

c) **Use of quotes:**
Single quotes are used for the single bit data double quotes for multiple data, and no quotes are used for integer data.

d) **Component instantiation:**
The following statement contains two errors

```
Control: shift GENERIC MAP(k=>3);
                PORT MAP ('1',Clock, w, Q);
```

There should be no semicolon at the end of the first line, because the two lines represent a single VHDL statement. Also it is illegal to associate a constant value ('1') with a port on a component. The following code shows how the two error can be fixed

```
Control: shift GENERIC MAP(k=>3)
         PORT MAP (High,Clock, w, Q);
```

e) **Label, signal and variable name:**
It is illegal to use any VHDL keyword as a label, signal or variable name. for example it is illegal to name a signal In or Out. Also it is illegal to use the same name multiple time for any label, signal or variable in a given VHDL design. It is illegal to define a single name I or i because VHDL doesnot distinguish between lower and upper case letter.

Important Questions

1) Implement 1:4 demux using VHDL. [IOE 2069 Ashad]
2) What do you mean by HDL? Design 2 to 4 line decoder circuit using HDL. [IOE 2068 ashad]

References

Donald P Leach, Albert Paul Malvino, Goutam Saha *"Digital principal and application" Tata McGraw-Hill, 2011.*

M. Morris Mano *"Digital logic and computer design" Prentice Hill of India.*

Anil K Maini *"Digital electronics principle device and application" John Wiley and sons, 2007.*

David J Corner *"Digital Logic and State Mechine Design" 3rd edition, oxford university press, 2002.*

D.L. Perry, *"VHDL" 3rd CMC Graw-hill: New York, 1998.*

Stephen Brown and Zvonko Vranesic *"Fundamental of digital logic with VHDL design" (second edition), MC Graw Hill.*

Cook, N. P. *"Practical Digital Electronics", Prentice-Hall, NJ, USA, 2003.*

Koren, I. *Computer Arithmetic Algorithms, A. K. Peters Ltd, Natick, MA, USA, 2001.*

Ercegovac, M. D. and Lang, T. *"Digital Arithmetic, Morgan Kaufmann Publishers", CA, 2003.*

Tocci, R. J. *"Digital Systems – Principles and Applications", Prentice-Hall Inc., NJ, USA, 2006.*

Floyd, T. L. *"Digital Fundamentals", Prentice-Hall Inc., USA, 2005.*

Tokheim, R. L. *"Schaum's Outline Series of Digital Principles", McGraw-Hill Companies Inc., USA, 1994.*

Rafiquzzaman, M. *"Fundamentals of Digital Logic and Microcomputer Design", Wiley-Interscience, NewYork, USA, 2005.*

Holdsworth, B. and Woods, C. *"Digital Logic Design", Newnes, Oxford, UK, 2002.*