LECTURE NOTES

ON

LINEAR & DIGITAL IC APPLICATIONS

B. TECH ECE II YEAR II SEMESTER

(AUTONOMOUS-R20)

Mr. C. LAXMANA SUDHEER, M. Tech
ASST. PROFESSOR

Mr. A J REUBEN THOMASRAJ, M. Tech
ASST. PROFESSOR

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

C.LAXMANASUDHEER LECTURE NOTES LINEAR & DIGITAL IC APPLICATIONS

(20EC0411) LINEAR & DIGITAL IC APPLICATIONS

COURSE OBJECTIVES

The objectives of this course:

1. *To Design of OPAMPS, Classification of OPAMPs*
2. *To study and design various linear and non linear applications of OPAMPs*
3. *To Learn VHDL programming Language.*
4. *To design Complex Combinational and Sequential circuits using Standard Digital ICs.*

COURSE OUTCOMES (COs)

On successful completion of this course, the student will be able to

1. *Able to define internal structures of the op amp and basic concepts of filters, timers and converters*
2. *Able to experiment the linear, nonlinear applications of op-amp with specialized ICs and converters.*
3. *Evaluate the applications of op-amp circuits, specialized ICs and converters.*
4. *Able to design the op amp circuits and converters for real time applications.*
5. *Understand CMOS and TTL Logic families and their interfacing.*
6. *Describe various design style of VHDL programming.*
7. *Apply the knowledge of VHDL programming to develop VHDL model for standard combinational and sequential IC structures.*

UNIT – I

Op-Amp Characteristics: Basic information of Op-amp−ideal and practical Op-amp −Op-amp block diagram−Op-amp characteristics – DC and AC characteristics−741 Op-amp and its features. **Op-Amp Linear Applications:** Modes of operation-Inverting, Non-inverting, Differential−Basic applications of Op-amp, Instrumentation amplifier, AC amplifier, V to I and I to V converters- **S**ample & Hold circuits− Differentiator and Integrator, −Comparators−Schmitt trigger.

UNIT – II

Active Filters: Introduction −1st order LPF −HPF filters −Band pass, −Band reject and all pass filters.

Oscillators: Oscillator types −Principle of operation, RC phase shift, Wien Bridge.

Timers: Introduction to 555 timer, Functional diagram, Monostable and Astable operations, Applications.

UNIT – III

Phase Locked Loops: Introduction, Block schematic, Principles and description of individual blocks of 565, Basic IC Regulators.

D/A and A/D Converters: Basic DAC techniques, Weighted resistor DAC, R-2R ladder. DAC−Different types of ADCs, Parallel comparator type ADC, Counter type ADC, Successive approximation ADC and dual slope ADC−DAC and ADC specifications.

CMOS Logic: Introduction to logic families−CMOS logic−Bipolar Logic−Transistor logic−low voltage CMOS logic and interfacing−Emitter coupled logic.

UNIT – IV

Hardware Description Languages: HDL Based Digital Design, the VHDL Hardware Description Language–Program Structure−Types−Constants and Arrays−Functions and procedures−Libraries and Packages−Structural design elements−Dataflow design elements−Behavioral design elements−The Time Dimension.

UNIT – V

Combinational Logic Design Practices- Description of basic structures like Decoders− Encoders −Comparators −Multiplexers (74 –series MSI)−Adders & sub tractors VHDL models for the above standard building block ICs.

Sequential Logic Design Practices: Latches & flip flops−counters− shift register and their VHDL models for the above standard building block ICs.

TEXT BOOKS

1. D. Roy Chowdhury, *Linear Integrated Circuits,* New Age International (p) Ltd, 2ndEdition., 2003.
2. John F. Wakerly *Digital Design Principles & Practices,* PHI/ Pearson Education Asia, 3rd Ed.2005.

REFERENCES

1. Ramakanth A. Gayakwad*, Op-amps & Linear IC, PHI*, 1987.
2. R.F. Coughlin & Fredric F. Driscoll, *Operational Amplifiers & Linear Integrated Circuits,* PHI.
3. Sergio Franco*, Design with Operational amplifiers & Analog Integrated circuits,* Mc GrawHill, 3rd Edition , 2002.
4. Floyd and Jain*, Digital Fundamentals,* Pearson Education, 8th Edition 2005.
5. J. Bhasker, *A VHDL Primer,* Pearson Education/ PHI, 3rd Edition.

UNIT-I

OP-AMP CHARACTERISTICS

1.1 OPERATIONAL AMPLIFIER (OP-AMP):

An operational amplifier is a direct-coupled high-gain amplifier usually consisting of one or more differential amplifiers and usually followed by a level translator and an output stage. An operational amplifier is available as a single integrated circuit package.

The operational amplifier is a versatile device that can be used to amplify dc as well as ac input signals and was originally designed for computing such mathematical functions as addition, subtraction, multiplication, and integration. Thus the name operational amplifier stems from its original use for these mathematical operations and is abbreviated to op-amp. With the addition of suitable external feedback components, the modern day op-amp can be used for a variety of applications, such as ac and dc signal amplification, active filters, oscillators, comparators, regulators, and others.

1.2 Ideal op-amp:

An ideal op-amp would exhibit the following electrical characteristics:

1. Infinite voltage gain
2. Infinite input resistance so that almost any signal source can drive it and there is no loading on the preceding stage.
3. Zero output resistance Ro so that output can drive an infinite number of other devices.
4. Zero output voltage when input voltage is zero.
5. Infinite bandwidth so that any frequency signal from 0 to ∞Hz can be amplified without attenuation.
6. Infinite common mode rejection ratio so that the output common-mode noise voltage is zero.
7. Infinite slew rate so that output voltage changes occur simultaneously with input voltage changes.

1.3 Equivalent circuit of an op-amp:

Fig. 1.1 shows an equivalent circuit of an op-amp. V1 and V2are the two input voltage voltages. Ri is the input impedance of OPAMP. Ad Vd is an equivalent Thevenin's voltage source and RO is the Thevenin's equivalent impedance looking back into theterminal.

This equivalent circuit is useful in analyzing the basic operating principles of op-amp and in observing the effects of standard feedback arrangements.

$$VO = Ad (V1-V2) = AdVd.$$

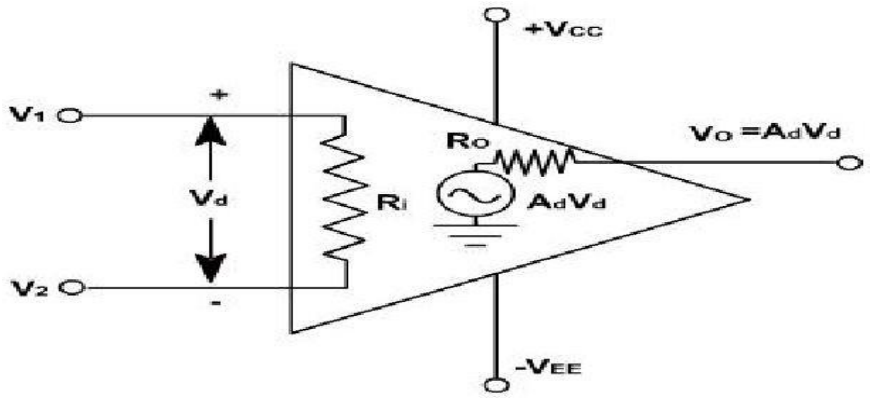


Fig 1.1: Equivalent circuit of OP-AMP

This equation indicates that the output voltage Vo is directly proportional to the algebraic difference between the two input voltages. In other words, the op amp amplifies the difference between the two input voltages. It does not amplify the input voltages themselves. The polarity of the output voltage depends on the polarity of the difference voltage Vd.

1.4 Ideal Voltage Transfer Curve:

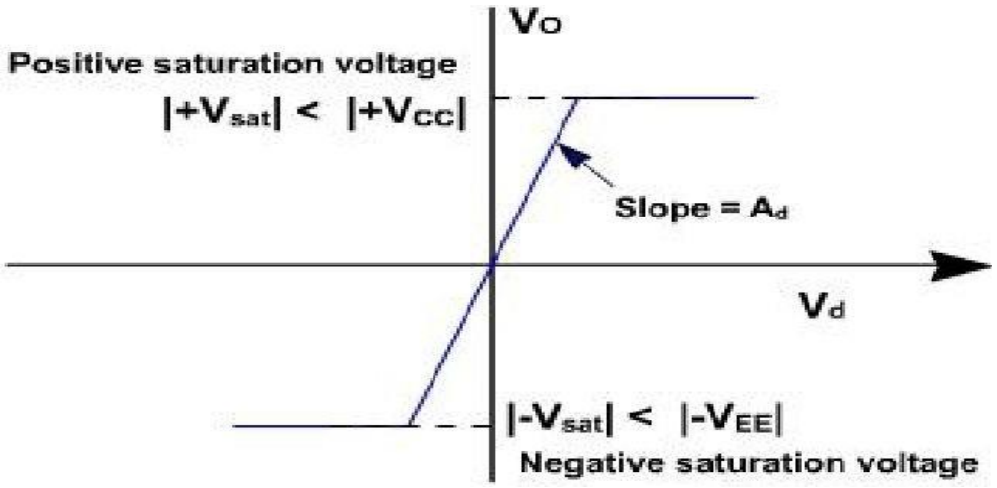


Fig 1.2: Ideal voltage transfer curve

The graphic representation of the output equation is shown infig.1.2 in which the output voltage Vo is plotted against differential input voltage Vd, keeping gain Ad constant. The output voltage cannot exceed the positive and negative saturation voltages. These saturation voltages are specified for given values of supply voltages. This means that the output voltage is directly proportional to the input difference voltage only until it reaches the saturation voltages and thereafter the output voltage remains constant. Thus, curve is called an ideal voltage transfer curve, ideal because output offset voltage is assumed to be zero. If the curve is drawn to scale, the curve would be almost vertical because of very large values of Ad.

1.5 INTERNAL CIRCUIT:

The operational amplifier is a direct-coupled high gain amplifier usable from 0 to over 1MHz to which feedback is added to control its overall response characteristic i.e. gain and bandwidth. The op-amp exhibits the gain down to zero frequency.

The internal block diagram of an op-amp is shown in the fig 1.3. The input stage is the dual input balanced output differential amplifier. This stage generally provides most of the voltage gain of the amplifier and also establishes the input resistance of the op-amp. The intermediate stage is usually another differential amplifier, which is driven by the output of the first stage. On most amplifiers, the intermediate stage is dual input, unbalanced output. Because of direct coupling, the dc voltage at the output of the intermediate stage is well above ground potential. Therefore, the level translator (shifting) circuit is used after the intermediate stage downwards to zero volts with respect to ground. The final stage is usually a push pull complementary symmetry amplifier output stage. The output stage increases the voltage swing and raises the ground supplying capabilities of the op-amp. A well-designed output stage also provides low output resistance.

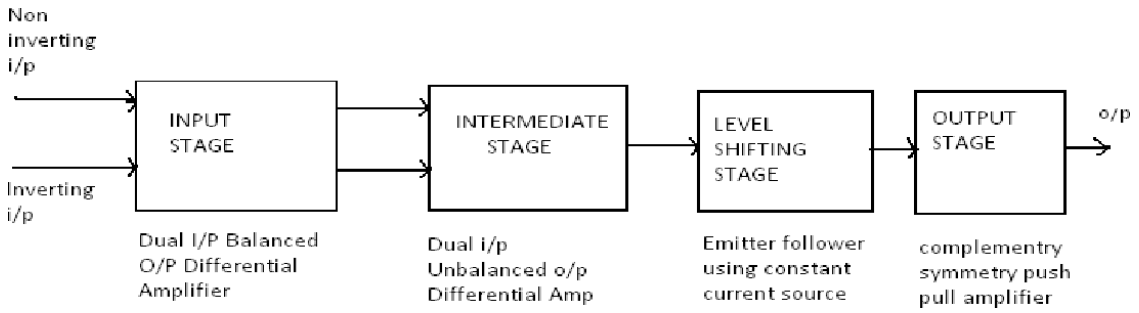


Fig 1.3: Block Diagram of OP-AMP

1.6 Differential amplifier:

Differential amplifier is a basic building block of an op-amp. The function of a differential amplifier is to amplify the difference between two input signals. The two transistors Q1 and Q2 have identical characteristics. The resistances of the circuits are equal, i.e. RE1 = R E2, RC1 = R C2 and the magnitude of +VCC is equal to the magnitude of -VEE. These voltages are measured with respect to ground.

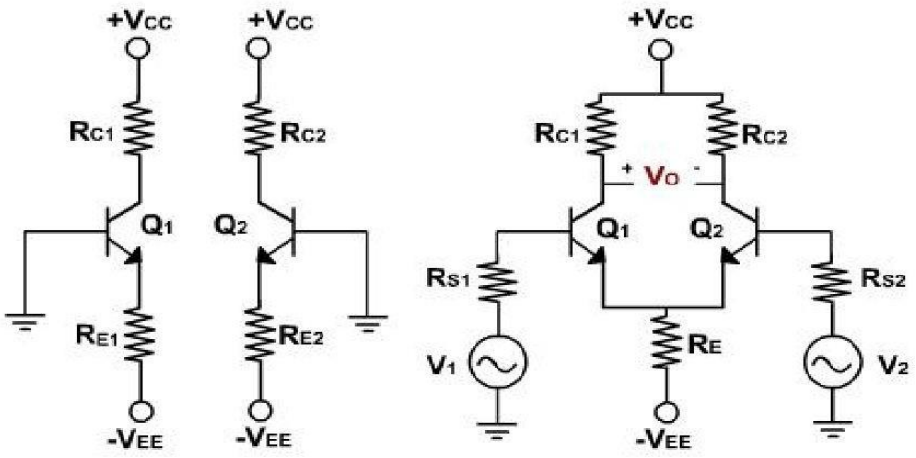


Fig 1.4: Differential Amplifier

To make a differential amplifier, the two circuits are connected as shown in fig. 1.4. The two +VCC and -VEE supply terminals are made common because they are same. The two emitters are also connected and the parallel combination of RE1 and RE2 is replaced by a resistance RE. The two input signals v1& v2 are applied at the base of Q1 and at the base of Q2. The output voltage is taken between two collectors. The collector resistances are equal and therefore denoted by RC = RC1 = RC2.

Ideally, the output voltage is zero when the two inputs are equal. When v1 is greater then v2 the output voltage with the polarity shown appears. When v1 is less than v2, the output voltage has the opposite polarity.

The four differential amplifier configurations are following:

1. Dual input, balanced output differential amplifier.
2. Dual input, unbalanced output differential amplifier.
3. Single input balanced output differential amplifier.
4. Single input unbalanced output differential amplifier.

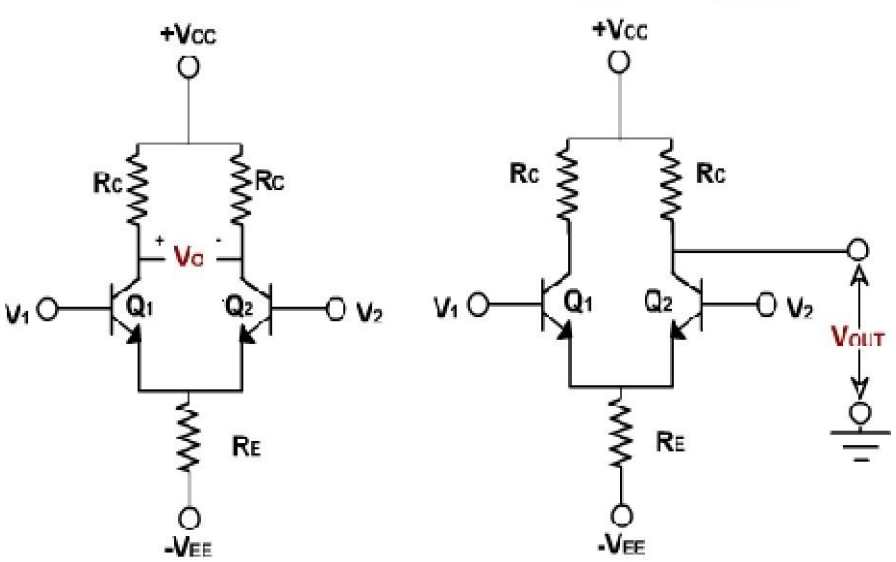


Fig 1.5: Dual input, balanced output differential amplifier.Fig.1.6. Dual input, unbalanced output differential amplifier.

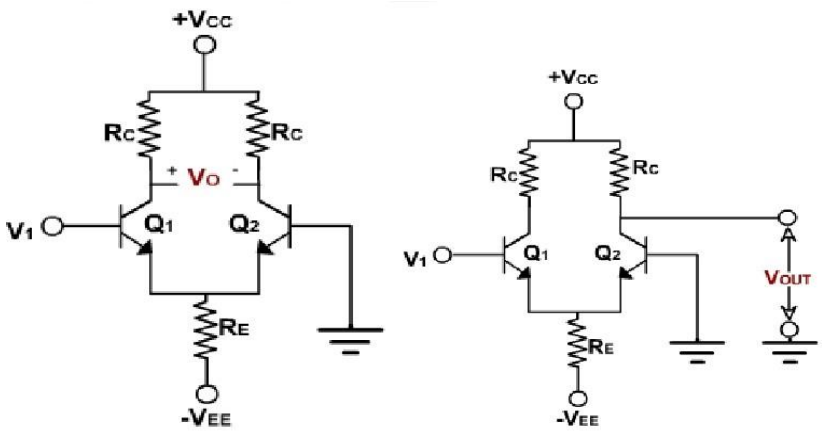


Fig 1.7: Single input, balanced output differential amplifier Fig.1.8.Single input, unbalanced output differential amplifier.

1.7 OP-AMP CHARACTERISTICS

1.7.1 DC CHARACTERISTICS:

a) Input Offset Voltage:

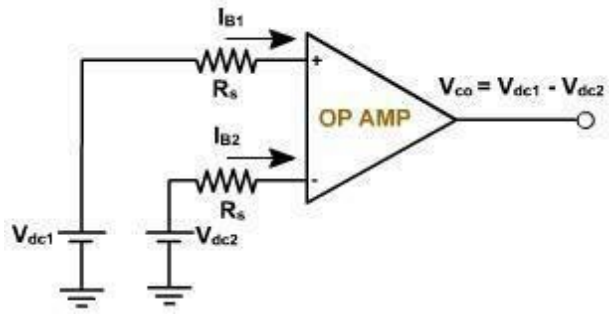


Fig 1.18: Input offset voltage

If no external input signal is applied to the op-amp at the inverting and non-inverting terminals the output must be zero. That is, if Vi=0, Vo=0. But as a result of the given biasing supply voltages, +Vcc and –Vcc, a finite bias current is drawn by the op-amps, and as a result of asymmetry on the differential amplifier configuration, the output will not be zero. This is known as offset. Since Vo must be zero when Vi=0 an input voltage must be applied such that the output offset is cancelled and Vo is made zero. This is known as input offset voltage. Input offset voltage (Vio) is defined as the voltage that must be applied between the two input terminals of an OPAMP to null or zero the output voltage. Fig 1.22 shows that two dc voltages are applied to input terminals to make the output zero.

$$Vio = Vdc1 - Vdc2$$

Vdc1 and Vdc2 are dc voltages and RS represents the source resistance. Vio is the difference of Vdc1 and Vdc2. It may be positive or negative. For a 741C OPAMP the maximum value of Vio is 6mV. It means a voltage ± 6 mV is required to one of the input to reduce the output offset voltage to zero. The smaller the input offset voltage the better the differential amplifier, because its transistors are more closely matched.

b) Input offset Current:

Though for an ideal op-amp the input impedance is infinite, it is not so practically. So the IC draws current from the source, however smaller it may be. This is called input offset current Iio. The input offset current Iio is the difference between the currents into inverting and non-inverting terminals of a balanced amplifier as shown in fig 1.22.

$$Iio = | IB1 - IB2 |$$

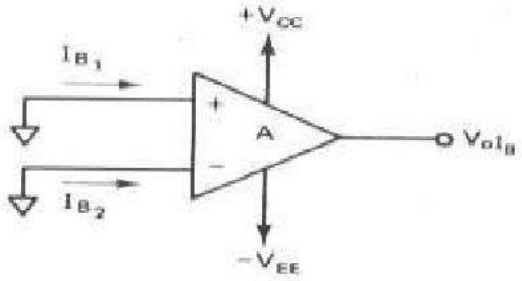
The Iio for the 741C is 200nA maximum. As the matching between two input terminals is improved, the difference between IB1 and IB2 becomes smaller, i.e. the Iio value decreases further. For a precision OPAMP 741C, Iio is 6 nA

c) Input bias current

Input bias current IB as the average value of the base currents entering into terminal of an op-amp.

$$I_B = \frac{I_{B1} + I_{B2}}{2}$$

where I_{B1} = dc bias current flowing into the noninverting input
I_{B2} = dc bias current flowing into the inverting input



$I_B = I_{B1} = I_{B2}$

Obtaining the expression for the output offset voltage caused by the input bias current IB in the inverting and non-inverting amplifiers and then devise some scheme to eliminate or minimize it.

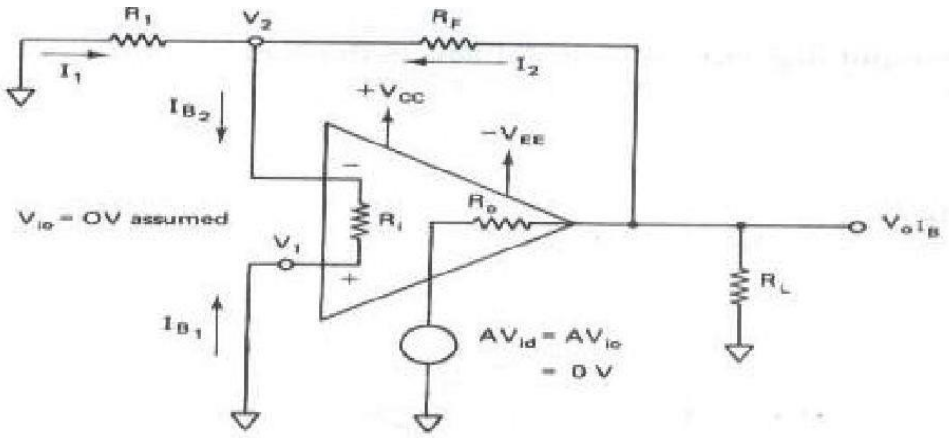


Fig: practical Op-Amp

d)Thermal Drift:

Bias current, offset current and offset voltage change with temperature. A circuit carefully nulled at 25oc may not remain so when the temperature rises to 35oc. This is called thermal drift.

1.7.2 AC CHARACTERISTICS:

a) Slew Rate

The slew rate is defined as the maximum rate of change of output voltage caused by a step input voltage. An ideal slew rate is infinite which means that op-amp's output voltage should change instantaneously in response to input step voltage.

The symbolic diagram of an OPAMP is shown in fig 1.21

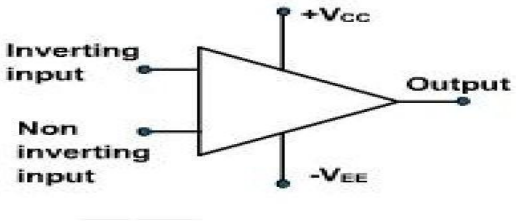


Fig 1.17: Op-Amp Symbol

b) Frequency Response

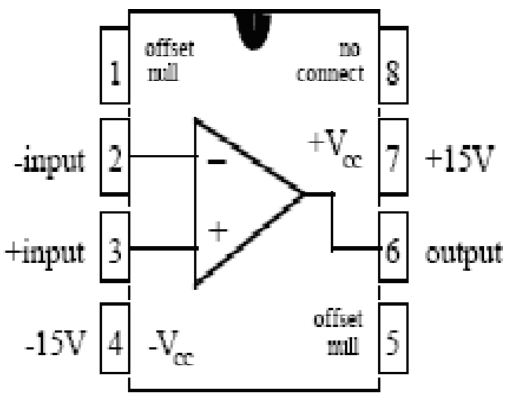
Need for frequency compensation in practical op-amps:

Frequency compensation is needed when large bandwidth and lower closed loop gain is desired. Compensating networks are used to control the phase shift and hence to improve the stability

Frequency compensation methods: a) Dominant- pole compensation b) Pole- zero compensation.

741c is most commonly used OPAMP available in IC package. It is an 8-pin DIP chip.

1.8 PIN DIAGRAM OF 741-OP AMP



Pinout for the 741 op amp.

1.9 FEATURES OF 741 OP-AMP:

1. No External frequency compensation is required
2. Short circuit Protection
3. Off Set Null Capability
4. Large Common mode and differential Voltage ranges
5. Low Power Dissipation
6. No-Latch up Problem
7. 741 is available in three packages: - 8-pin metal can, 10-pin flat pack and 8 or 14-pin DI.

1.10 MODES OF OPERATION OF OP-AMP

There are 2 modes in which an op-amp operates:

1. open loop mode
2. closed loop mode

Open loop OPAMP mode:

In the case of amplifiers, the term open loop indicates that no connection exists between input and output terminals of any type. That is, the output signal is not feedback in any form as part of the input signal. In open loop configuration, The OPAMP functions as a high gain amplifier. There are three open loop OPAMP configurations.

1. The Differential Amplifier:

The open loop differential amplifier in which input signals vin1 and vin2 are applied to the positive and negative input terminals. Since the OPAMP amplifies the difference the between the two input signals, this configuration is called the differential amplifier. The OPAMP amplifies both ac and dc input signals. The source resistance Rin1 and Rin2 are normally negligible compared to the input resistance Ri. Therefore voltage drop across these resistances can be assumed to be zero.

Therefore

$$v1 = vin1 \text{ and } v2 = vin2.\ vo = Ad\ (vin1 - vin2\ )$$

where, Ad is the open loop gain.

2. The Inverting Amplifier:

If the input is applied to only inverting terminal and non-inverting terminal is grounded then it is called inverting amplifier. This configuration is shown in fig 1.27.

v1= 0, v2 = vin. vo = -Ad vi

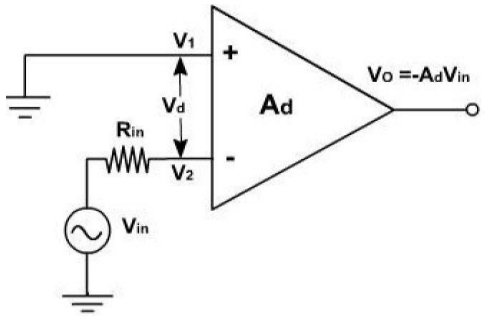


Fig 1.20: Inverting Amplifier

The negative sign indicates that the output voltage is out of phase with respect to input 180 ° or is of opposite polarity. Thus the input signal is amplified and inverted also.

3. The non-inverting amplifier:

In this configuration, the input voltage is applied to non-inverting terminals and inverting terminal is ground as shown in fig.1.28

v1 = +vin, v2 = 0 vo = +Ad vin

This means that the input voltage is amplified by Ad and there is no phase reversal at the output.

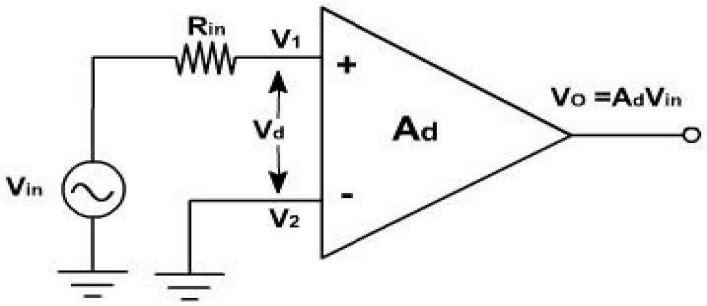


Fig 1.21: Non-Inverting Amplifier

In all their configurations any input signal slightly greater than zero drive the output to saturation level. This is because of very high gain. Thus, when operated in open-loop, the output of the OPAMP is either negative or positive saturation or switches between positive and negative saturation levels. Therefore, open loop op-amp is not used in linear applications.

Closed Loop mode:

The Open Loop Gain of an ideal operational amplifier can be very high, as much as 1,000,000 (120dB) or more. However, this very high gain is of no real use to us as it makes the amplifier both unstable and hard to control as the smallest of input signals, just a few micro-volts, (μV) would be enough to cause the output voltage to saturate and swing towards one or the other of the voltage supply rails losing complete control of the output.

As the open loop DC gain of an operational amplifier is extremely high, we can therefore afford to lose some of this high gain by connecting a suitable resistor across the amplifier from the output terminal back to the inverting input terminal to both reduce and control the overall gain of the amplifier. This then produces and effect known commonly as Negative Feedback, and thus produces a very stable Operational Amplifier based system.

Negative Feedback is the process of "feeding back" a fraction of the output signal back to the input, but to make the feedback negative, we must feed it back to the negative or "inverting input" terminal of the op-amp using an external Feedback Resistor called Rf. This feedback connection between the output and the inverting input terminal forces the differential input voltage towards zero.

This effect produces a closed loop circuit to the amplifier resulting in the gain of the amplifier now being called its Closed-loop Gain. Then a closed-loop inverting amplifier uses negative feedback to accurately control the overall gain of the amplifier, but at a cost in the reduction of the amplifier's bandwidth. This negative feedback results in the inverting input terminal having a different signal on it than the actual input voltage as it will be the sum of the input voltage plus the negative feedback voltage giving it the label or term of a *Summing Point*. We must therefore separate the real input signal from the inverting input by using an Input Resistor, Rin. As we are not using the positive non-inverting input this is connected to a common ground or zero voltage terminal as shown below, but the effect of this closed loop feedback circuit results in the voltage potential at the inverting input being equal to that at the non-inverting input producing a *Virtual Earth* summing point because it will be at the same potential as the grounded reference input. In other words, the op-amp becomes a "differential amplifier".

1.11 Inverting Amplifier Configuration

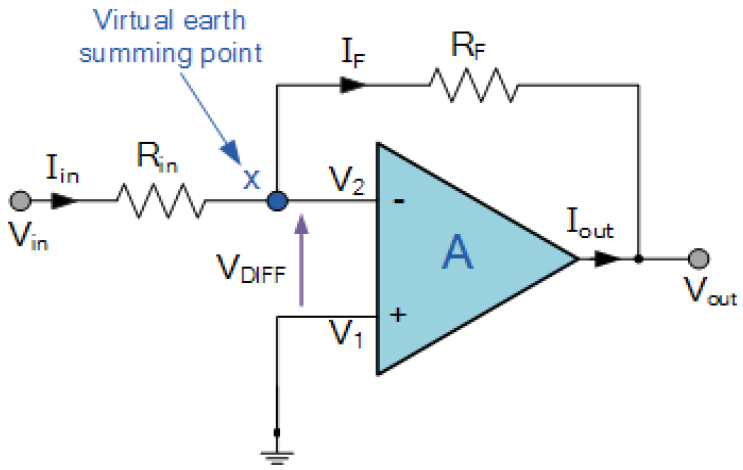


Fig 1.22: Inverting amplifier with feedback.

In this Inverting Amplifier circuit the operational amplifier is connected with feedback to produce a closed loop operation. For ideal op-amps there are two very important rules to remember about inverting amplifiers, these are: "no current flows into the input terminal" and that "V1 equals V2", (in real world op-amps both of these rules are broken).

This is because the junction of the input and feedback signal (X) is at the same potential as the positive (+) input which is at zero volts or ground then, the junction is a "Virtual Earth". Because of this virtual earth node the input resistance of the amplifier is equal to the value of the input resistor, Rin and the closed loop gain of the inverting amplifier can be set by the ratio of the two external resistors.

We said above that there are two very important rules to remember about Inverting Amplifiers or any operational amplifier for that matter and these are.

1. No Current Flows into the Input Terminals
2. The Differential Input Voltage is Zero as V1 = V2 = 0 (Virtual Earth)

Then by using these two rules we can derive the equation for calculating the closed-loop gain of an inverting amplifier, using first principles.

Current (i) flows through the resistor network as shown.

$$i = \frac{Vin - Vout}{Rin + Rf}$$

$$\text{therefore, } i = \frac{Vin - V2}{Rin} = \frac{V2 - Vout}{Rf}$$

$$i = \frac{Vin}{Rin} - \frac{V2}{Rin} = \frac{V2}{Rf} - \frac{Vout}{Rf}$$

$$\text{so, } \frac{Vin}{Rin} = V2\left[\frac{1}{Rin} + \frac{1}{Rf}\right] - \frac{Vout}{Rf}$$

$$\text{and as, } i = \frac{Vin - 0}{Rin} = \frac{0 - Vout}{Rf} \qquad \frac{Rf}{Rin} = \frac{0 - Vout}{Vin - 0}$$

$$\text{the Closed Loop Gain (Av) is given as, } \frac{Vout}{Vin} = -\frac{Rf}{Rin}$$

Then, the Closed-Loop Voltage Gain of an Inverting Amplifier is given as

$$A_V = \frac{V_{out}}{V_{in}} = -\frac{R_f}{R_{in}}$$

and this can be transposed to give Vout as: $V_{out} = -\frac{R_f}{R_{in}} * Vin$

The negative sign in the equation indicates an inversion of the output signal with respect to the input as it is 180o out of phase. This is due to the feedback being negative in value.

1.12 The Non-inverting Amplifier

The second basic configuration of an operational amplifier circuit is that of a Non-inverting Amplifier. In this configuration, the input voltage signal, (Vin) is applied directly to the non- inverting (+) input terminal which means that the output gain of the amplifier becomes "Positive" in value in contrast to the "Inverting Amplifier" circuit we saw in the last tutorial whose output gain is negative in value. The result of this is that the output signal is "in-phase" with the input signal.

Feedback control of the non-inverting amplifier is achieved by applying a small part of the output voltage signal back to the inverting (-) input terminal via a R*f* - R2 voltage divider network, again producing negative feedback. This closed-loop configuration produces a non-inverting amplifier circuit with very good stability, very high input impedance, Rin approaching infinity, as no current flows into the positive input terminal, (ideal conditions) and low output impedance, Rout as shown below.

Non-inverting Amplifier Configuration

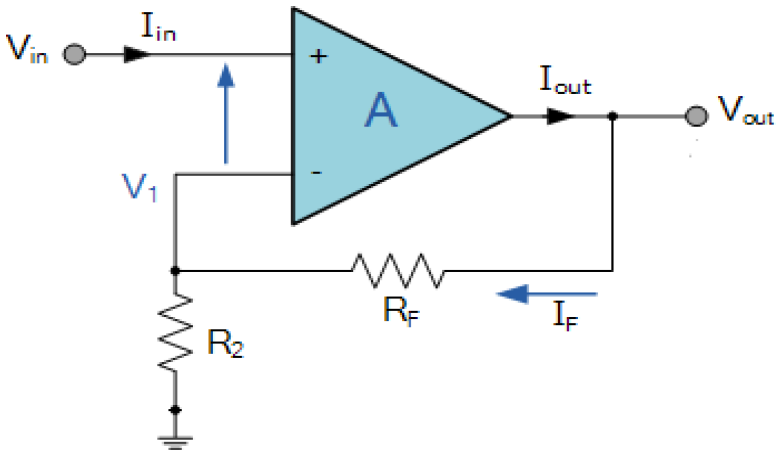


Fig 1.23: Non-inverting amplifier with feedback.

As said in the Inverting Amplifier that "no current flows into the input" of the amplifier and that "V1 equals V2". This was because the junction of the input and feedback signal (V1) are at the same potential. In other words the junction is a "virtual earth" summing point. Because of this virtual earth node the resistors, R*f* and R2 form a simple potential divider network across the non-inverting amplifier with the voltage gain of the circuit being determined by the ratios of R2 and R*f* as shown below.

Equivalent Potential Divider Network

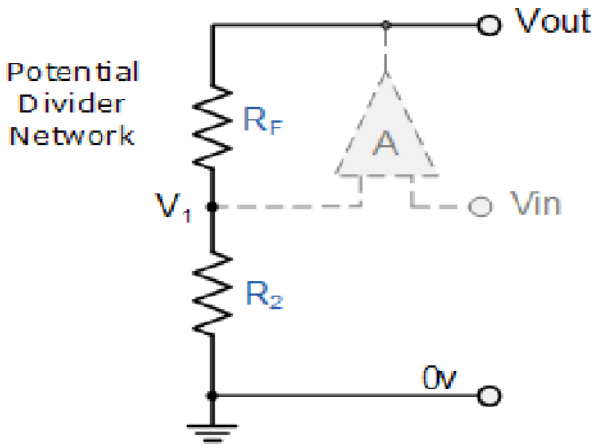


Fig 1.24: potential divider in non-inverting op-amp

From the fig 1.31 using the formula to calculate the output voltage of a potential divider network, we can calculate the closed-loop voltage gain (A V) of the **Non-inverting Amplifier** as follows:

$$V_1 = \frac{R_2}{R_2 + R_F} \times V_{OUT}$$

$$\text{Ideal Summing Point: } V_1 = V_{IN}$$

$$\text{Voltage Gain, } A_{(V)} \text{ is equal to: } \frac{V_{OUT}}{V_{IN}}$$

$$\text{Then, } A_{(V)} = \frac{V_{OUT}}{V_{IN}} = \frac{R_2 + R_F}{R_2}$$

$$\text{Transpose to give: } A_{(V)} = \frac{V_{OUT}}{V_{IN}} = 1 + \frac{R_F}{R_2}$$

We can see from the equation above, that the overall closed-loop gain of a non-inverting amplifier will always be greater but never less than one (unity), it is positive in nature and is determined by the ratio of the values of R*f* and R2. If the value of the feedback resistor R*f* is zero, the gain of the amplifier will be exactly equal to one (unity). If resistor R2 is zero the gain will approach infinity, but in practice it will be limited to the operational amplifiers open-loop differential gain, (Ao).

Voltage Follower (Unity Gain Buffer)

If we made the feedback resistor, R*f* equal to zero, (R*f* = 0), and resistor R2 equal to infinity, (R2 = ∞) as shown in fig 1.32, then the circuit would have a fixed gain of "1" as all the output voltage would be present on the inverting input terminal (negative feedback). This would then produce a special type of the non-inverting amplifier circuit called a Voltage Follower or also called a "unity gain buffer".

As the input signal is connected directly to the non-inverting input of the amplifier the output signal is not inverted resulting in the output voltage being equal to the input voltage, Vout = Vin. This then makes the voltage follower circuit ideal as a *Unity Gain Buffer* circuit because of its isolation properties as impedance or circuit isolation is more important than amplification while maintaining the signal voltage. The input impedance of the voltage follower circuit is very high, typically above 1MΩ as it is equal to that of the operational amplifiers input resistance times its gain (Rin x Ao). Also, its output impedance is very low since an ideal op-amp condition is assumed.

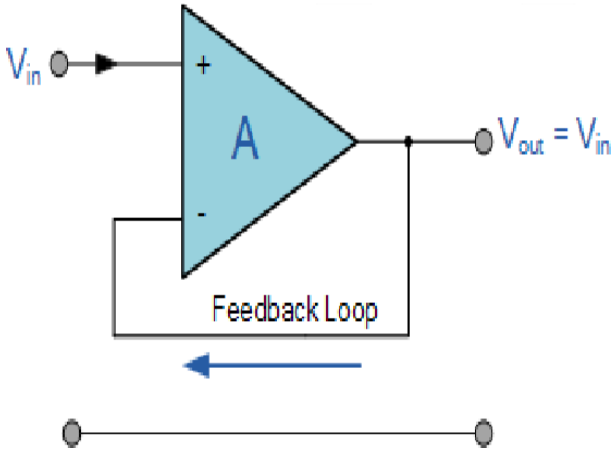


Fig 1.25: voltage follower

In this non-inverting circuit configuration, the input impedance Rin has increased to infinity and the feedback impedance Rf reduced to zero. The output is connected directly back to the negative inverting input so the feedback is 100% and Vin is exactly equal to Vout giving it a fixed gain of 1 or unity. As the input voltage Vin is applied to the non-inverting input the gain of the amplifier is given as:

$$V_{out} = A \times V_{in}$$

$$V_{in} = V + \ and \ V_{out} = V$$

One final thought, the output voltage gain of the voltage follower circuit with closed loop gain is **Unity**, the voltage gain of an ideal operational amplifier with open loop gain (no feedback) is **Infinite**. Then by carefully selecting the feedback components we can control the amount of gain produced by an operational amplifier anywhere from one to infinity.

1.13 INSTRUMENTATION AMPLIFIER:

In many industrial and consumer applications the measurement and control of physical conditions are very important. For example measurements of temperature and humidity inside a dairy or meat plant permit the operator to make necessary adjustments to maintain product quality. Similarly, precise temperature control of plastic furnace is needed to produce a particular type of plastic.

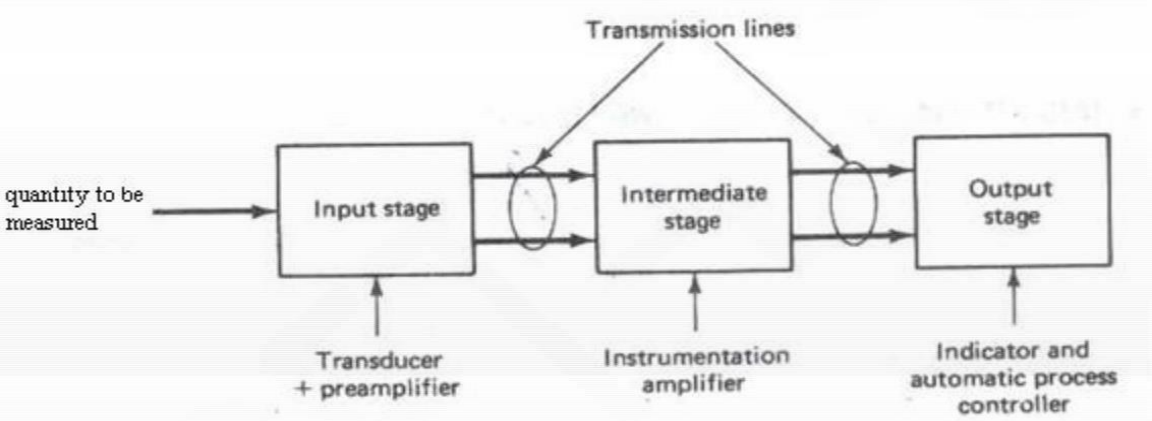


Fig.1.26: Instrumentation Amplifier

The transducer is a device that converts one form of energy into another. For example a strain gage when subjected to pressure or force undergoes a change in its resistance (electrical energy).An instrumentation system is used to measure the output signal produced by a transducer and often to control the physical signal producing it. Above fig shows a simplified form of such a system. The input stage is composed of a pre-amplifier and some sort of transducer, depending on the physical quantity to be measured. The output stage may use devices such as meters, oscilloscopes, charts, or magnetic records.

In Figure 1.33 the connecting lines between the blocks represent transmission lines, used especially when the transducer is at a remote test site monitoring hazardous conditions such as high temperatures or liquid levels of flammable chemicals. These transmission lines permit signal transfer from unit to unit. The length of the transmission lines depends primarily on the physical quantities tobe monitored and on system requirements.

The signal source of the instrumentation amplifier is the output of the transducer. Although some transducers produce outputs with sufficient strength to per- m.; their use directly, many do not. To amplify the low-level output signal of the transducer so that it can drive the indicator or display is the major function of the instrumentation amplifier. In short, the instrumentation amplifier is intended for precise, low-level signal amplification where low noise, low thermal and time drifts, high input resistance, and accurate closed-loop gain are required. Besides, low power consumption, high common-mode rejection ratio, and high slew rate are desirable for superior performance.

There are many instrumentation operational amplifiers, such as the /LA 725, ICL7605, and LH0036, that make a circuit extremely stable and accurate. These ICs are, however, relatively expensive; they are very precise special-purpose circuits in which most of the electrical parameters, such as offsets, drifts, and power consumption, are minimized,

whereas input resistance, CMRR, and supply range are optimized. Some instrumentation amplifiers are even available in modular form to suit special installation requirements.

Obviously, the requirements for instrumentation op-amps are more rigid than those for general-purpose applications. However, where the requirements are not too strict, the general-purpose op-amp can be employed in the differential mode.

We will call such amplifiers differential instrumentation amplifiers. Since most instrumentation systems use a transducer in a bridge circuit, we will consider a simplified differential instrumentation system arrangement using a transducer bridge circuit.

1.14 AC AMPLIFIER

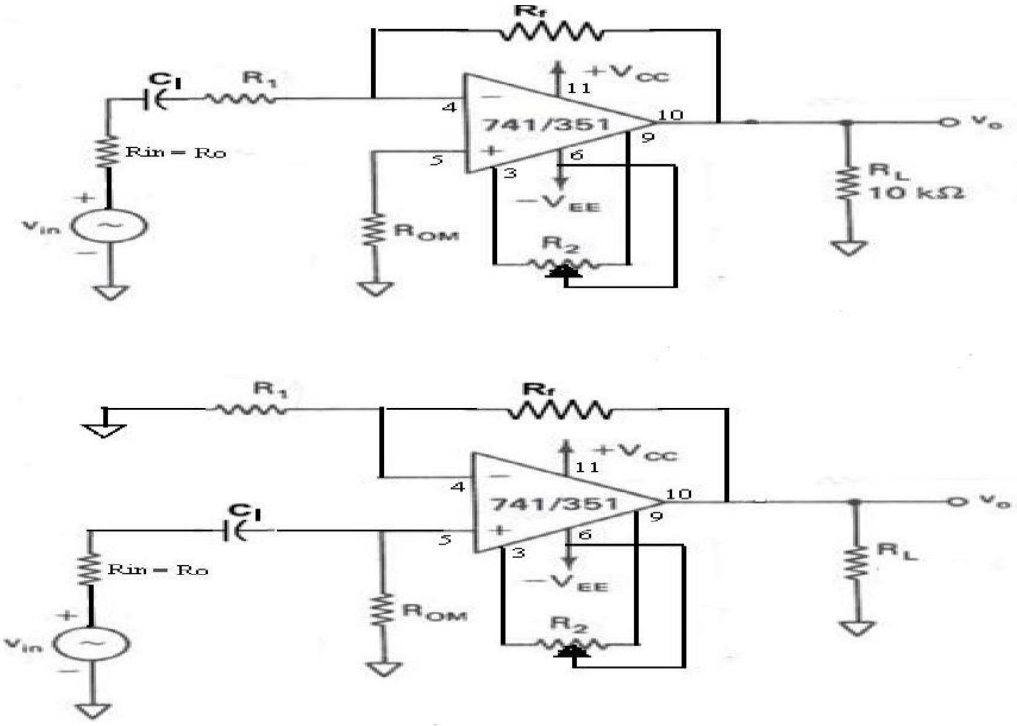


Fig 1.27: (a) AC Inverting Amplifier (b) AC Non-Inverting Amplifier

1.15 V to I Converter:

Fig.1.35 shows a voltage to current converter in which load resistor RL is floating (not connected to ground). The input voltage is applied to the non-inverting input terminal and the feedback voltage across R drives the inverting input terminal. This circuit is also called a current series negative feedback, amplifier because the feedback voltage across R depends on the output current iL and is in series with the input difference voltage Vd.

Writing the voltage equation for the input loop.

$$vin = vd + vf$$

But vd » since A is very large, therefore, vin = vf vin = Rin

Iin = v in / R.

and since input current is zero.

iL = iin = vin. / R

The value of load resistance does not appear in this equation. Therefore, the output current is independent of the value of load resistance. Thus the input voltage is converted into current, the source must be capable of supplying this load current.

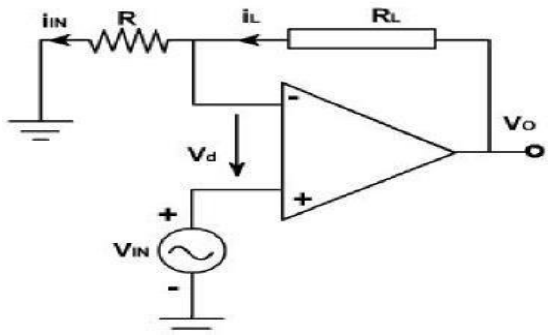


Fig 1.28: Circuit Diagram of V to I Converter

The maximum load current is VCC/ R. In this circuit v in may be positive or negative.

1.16 I to V Converter:

Current to voltage converter:

The circuit shown in fig 1.36 is a current to voltage converter.

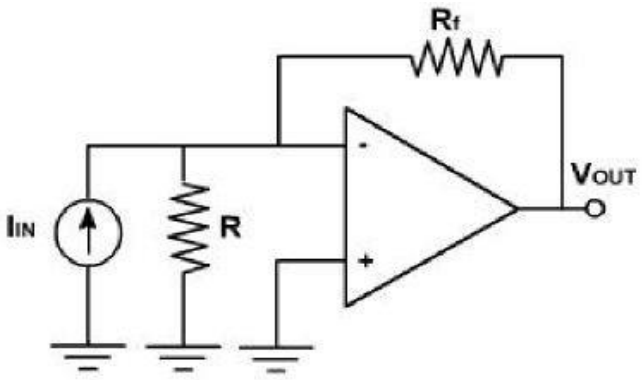


Fig 1.29: Circuit Diagram of I to V Converter

Due to virtual ground the current through R is zero and the input current flows through Rf.

Therefore, vout =-Rf * iin

The lower limit on current measure with this circuit is set by the bias current of the inverting input.

1.17 SAMPLE AND HOLD CIRCUITS:

The sample and hold circuit, as its name implies samples an i/p signal and holds on to it last sampled value until the i/p is sampled again. Below fig shows a sample and hold circuit using an op-amp with an E- MOSFET. In this circuit the E-MOSFET works as a switch that is controlled by the sample and control voltage Vs, and the capacitor C serves as a storage element.

The analog signal Vin to be sampled is applied to the drain, and sample and hold control voltage Vs is applied to the gate of the E-MOSFET. During the positive portion of the Vs, the EMOSFET conducts and acts as a closed switch. This allows i/p voltage to charge capacitor C. In other words, input voltage appears across C and in turn at the o/p as shown in above fig.2.9. On the other hand, when Vs is zero, the EMOSFET is off and acts as open switch. The only discharge path for C is, through the op-amp. However, the i/p resistance of the op-amp voltage follower is also very high; hence the voltage across C is retained.

The time periods Ts of the sample-and-hold control voltage Vs during which the voltage across the capacitor is equal to the i/p voltage are called sample periods. The time periods TH of Vs during which the voltage across the capacitor is constant are called hold periods. The o/p of the op-amp is usually processed/ observed during hold periods. To obtain the close approximation of the i/p waveform, the frequency of the sample-and-hold control voltage must be significantly higher than that of the i/p.

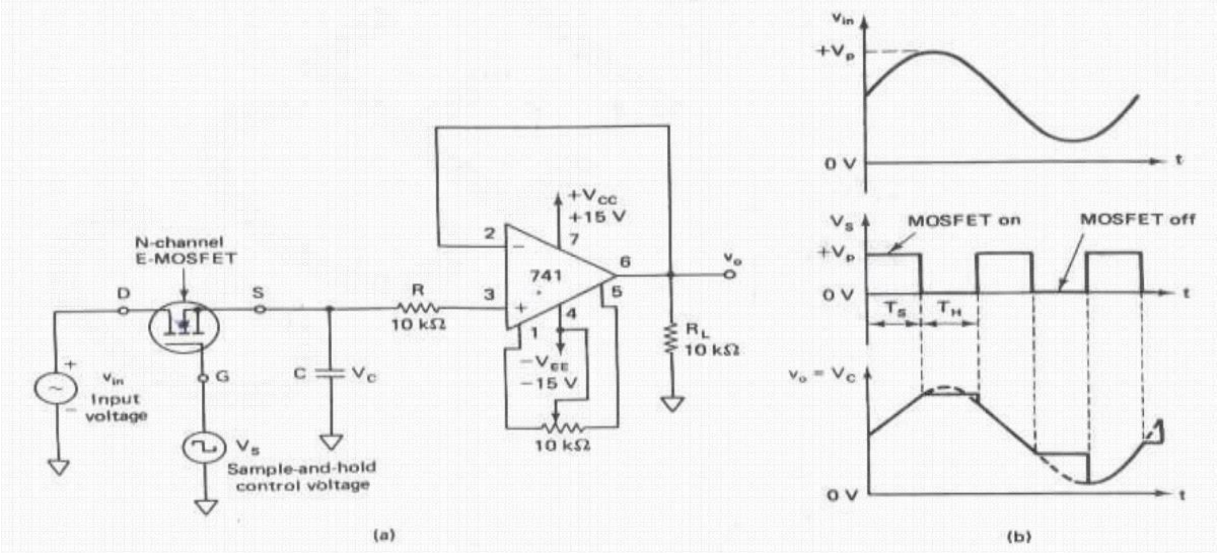


Fig.1.30: sample and hold circuit Fig 1.38 I/P and O/P wave forms

1.18 DIFFERENTIATOR:

A circuit in which the output voltage waveform is the differentiation of input voltage is called differentiator as shown infig.2.10.

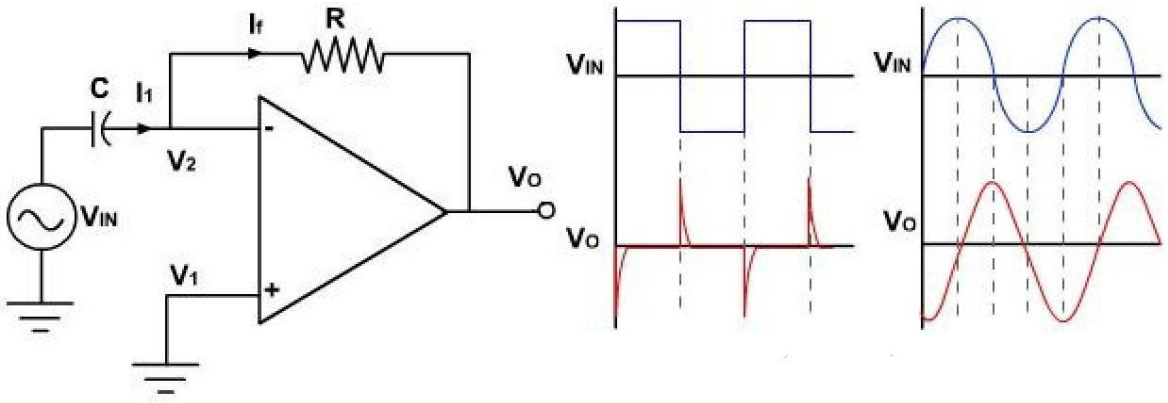


Fig.1.31: Circuit Diagram of Differentiator

The expression for the output voltage can be obtained from the Kirchoff's current equation written at node v2.

$$\text{Since,} \qquad i_{in} = i_f$$

$$\text{Therefore,} \quad C\frac{d}{dt}(V_{in}-0)=\frac{0-V_o}{R}$$

$$V_O = -RC\frac{dV_{in}}{dt}$$

Thus the output vo is equal to the RC times the negative instantaneous rate of change of the input voltage vin with time. A cosine wave input produces sine output. Fig.1.39 also shows the output waveform for different input voltages.

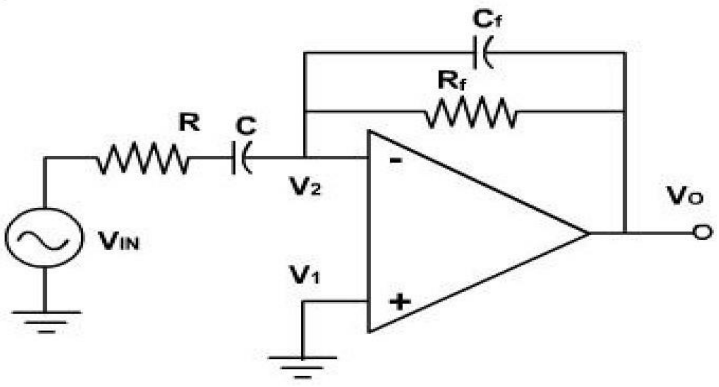


Fig.1.32: Circuit Diagram of Differentiator

The input signal will be differentiated properly if the time period T of the input signal is larger than or equal to Rf C. As the frequency changes, the gain changes. Also at higher

frequencies the circuit is highly susceptible at high frequency noise and noise gets amplified. Both the high frequency noise and problem can be corrected by adding, few components. as shown in fig.1.40.

1.19 Integrator:

A circuit in which the output voltage waveform is the integral of the input voltage waveform is called integrator. Fig.1.41**,** shows an integrator circuit using OPAMP.

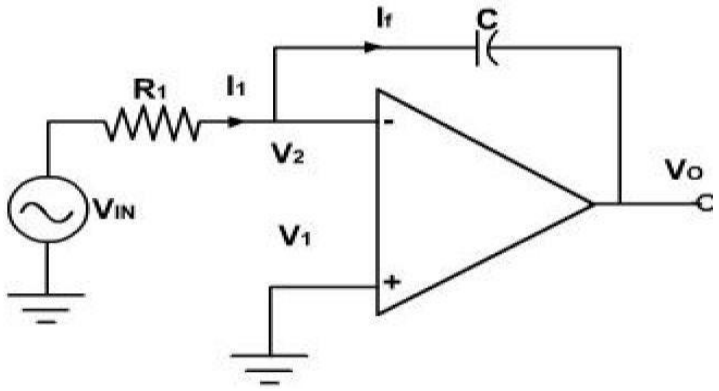


Fig.1.33: Circuit Diagram of Integrator

Here, the feedback element is a capacitor. The current drawn by OPAMP is zero and also the V2 is virtually grounded.

Therefore, i1 = if and v2 = v1 = 0

$$\frac{v_{in} - 0}{R} = C\,\frac{d(0 - v_o)}{dt}$$

Integrating both sides with respect to time from 0 to t, we get

$$\int_0^t \frac{v_{in}}{R}dt = \int_0^t C\frac{d(-v_o)}{dt}dt$$

$$= C(-v_o) + v_o\Big|_{t=0}$$

if $V_O|_{t=0} = 0\ V$, then

$$v_o = \frac{-1}{R}\int_0^t v_{in}\,dt$$

The output voltage is directly proportional to the negative integral of the input voltage and inversely proportional to the time constant RC. If the input is a sine wave the output will be cosine wave. If the input is a square wave, the output will be a triangular wave. For accurate integration, the time period of the input signal T must be longer than or equal to RC.

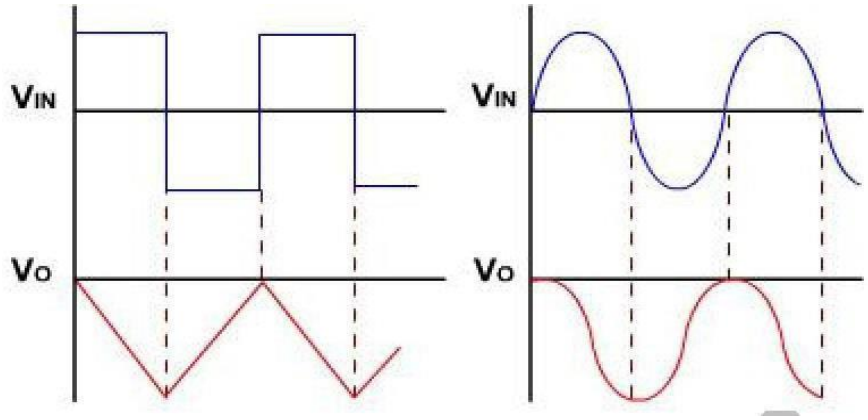


Fig 1.34: Input and Output wave forms

1.20 COMPARATOR:

Voltage comparator circuit:

Voltage comparator is a circuit which compares two voltages and switches the output to either high or low state depending upon which voltage is higher. A voltage comparator based on opamp is shown here. Fig2.14 shows a voltage comparator in inverting mode and Fig shows a voltage comparator in non inverting mode.

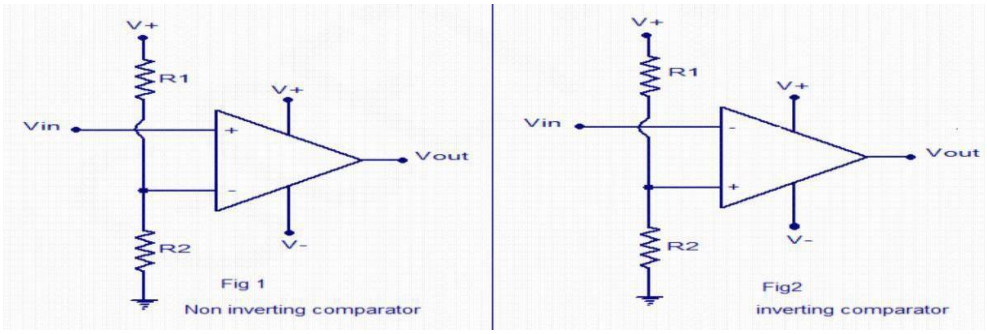


Fig 1.35: Circuit Diagram of Comparator

Non inverting comparator:

In non inverting comparator the reference voltage is applied to the inverting input and the voltage to be compared is applied to the non inverting input. Whenever the voltage to be compared (Vin) goes above the reference voltage , the output of the opamp swings to positive saturation (V+) and vice versa. Actually what happens is that, the difference between Vin and Vref, (Vin – Vref) will be a positive value and is amplified to infinity by the opamp. Since there is no feedback resistor Rf, the opamp is in open loop mode and so the voltage gain (Av) will be close to infinity. So the output voltage swings to the maximum possible value ie; V+. Remember the equation Av = 1 + (Rf/R1).

When the Vin goes below Vref, the reverse occurs.

Inverting comparator.

In the case of an inverting comparator, the reference voltage is applied to the non-inverting input and voltage to be compared is applied to the inverting input. Whenever the input voltage (Vin) goes above the Vref, the output of the op-amp swings to negative saturation. Here the difference between two voltages (Vin-Vref) is inverted and amplified to infinity by the op-amp. Remember the equation Av = -Rf/R1. The equation for voltage gain in the inverting mode is Av = -Rf/R1.Since there is no feedback resistor, the gain will be close to infinity and the output voltage will be as negative as possible i.e., V-.

Practical voltage comparator circuit.

A practical non inverting comparator based on uA741 opamp is shown below. Here the reference voltage is set using the voltage divider network comprising of R1 and R2. The equation is Vref = (V+/ (R1 + R2)) x R2. Substituting the values given in the circuit diagram into this equation gives Vref = 6V. Whenever Vin goes above 6V, the output swings to ~+12V DC and vice versa. The circuit is powered from a +/- 12V DC dual supply.

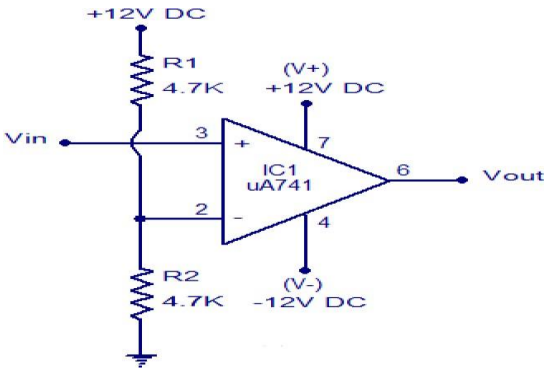
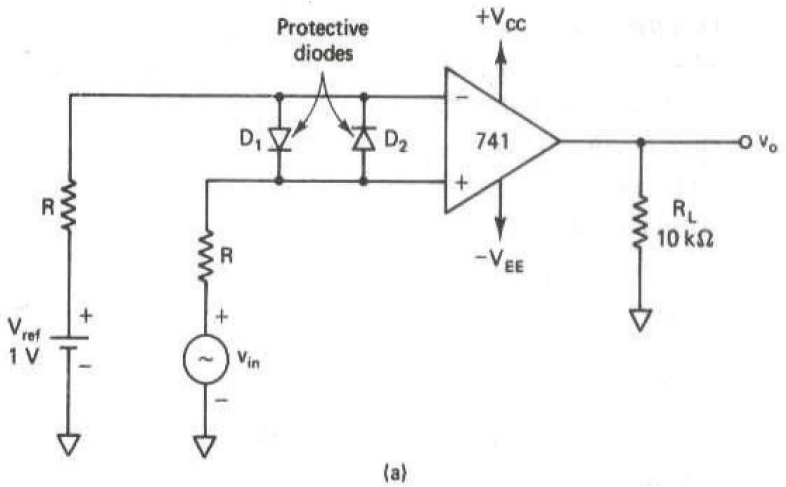


Fig 1.36: Circuit diagram of Practical voltage comparator.

Op-amp voltage comparator



(a)

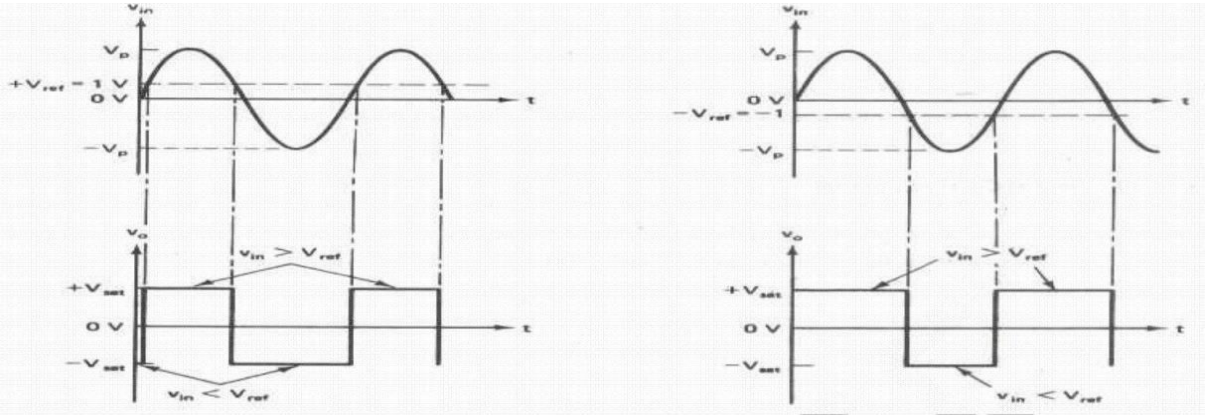


Fig 1.37: OP-AMP voltage comparator input and output wave forms(a,b,c)

1.21 SCHMITT TRIGGER:

Below fig shows an inverting comparator with +ve feedback. This circuit converts an irregular shaped wave forms to a square wave form or pulse. The circuit is known as schmitt trigger or squaring circuit. The i/p voltage being triggers the o/p Vo every time it exceeds certain voltage levels called the upper threshold voltage Vut and lower threshold voltage Vlt as shown in fig 1.45 (b).In fig 1.45 (a) these threshold voltages are obtained by using the voltage divider R1, R2, where the voltage across R1 is F/B to +ve i/p. The voltage across R1 is a variable reference, threshold voltage that depends on the value and polarity of the output voltage Vo. when Vo=+Vsat, the voltage across R1 is called the upper threshold voltage Vut.

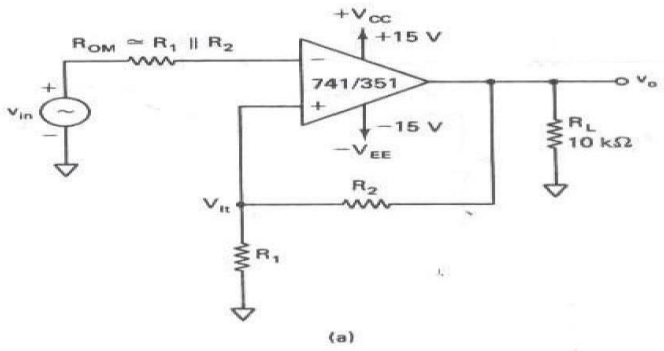


Fig 1.38 Schmitt Trigger

The input voltage Vin must be slightly more positive then Vut in order to cause the output Vo to switch from +Vsat to –Vsat.as long as Vin less then Vut,Vo is at +Vsat. using the voltage divider rule, On the other hand,when Vo=-Vsat, the voltage across R1 is referred to as the lower threshold voltage,Vlt.Vin must be slightly more negative than Vlt.in order to cause Vo to switch from-Vsat to +Vsat.in other words,for Vin values greater than Vlt,Vo is at –Vsat.Vlt is given by the following equation;

Thus if the threshold voltages Vut and Vlt are made large than the input noise voltages, the positive fed back will eliminate the false output transitions.Also the +ve feedback because of its regenerative action will make Vo switch faster between +Vsat and –Vsat.

UNIT – II
ACTIVE FILTERS, OSCILLATORS & TIMERS

2.1 LOW PASS FILTER:

An electric filter is often a frequency-selective circuit that passes a specified band of frequencies and blocks or attenuates signals of frequencies outside this band. Filters may be classified in a number of ways:

1. Analog or digital

2. Passive or active

3. Audio (AF) or radio frequency (RF)

Analog filters are designed to process analog signals, while digital filters process analog signals using digital techniques. Depending on the type of elements used in their construction, filters maybe classified as passive or active. Elements used in passive filters are resistors, capacitors, and inductors. Active filters, on the other hand, employ transistors or op-amps in addition to the resistors and capacitors. The type of element used dictates the operating frequency range of the filter.

For example, RC filters are commonly used for audio or low-frequency operation, whereas LC or crystal filters are employed at RF or high frequencies. Especially because of their high Q value (figure of merit), the crystal provides more stable operation at higher frequencies.

An active filter offers the following advantages over a passive filter:

1. Gain and frequency adjustment flexibility: Since the op-amp is capable of providing again, the input signal is not attenuated as it is in a passive filter. In addition, the active filter is easier to tune or adjust.

2. No loading prob1em: Because of the high input resistance and low output resistance of the op- amp, the active filter does not cause loading of the source or load.

3. Cost: Typically, active filters are more economical than passive filters. This is because of the variety of cheaper op- amps and the absence of inductors.

The most commonly used filters are these:

1. Low-pass filter
2. High-pass filter
3. Band-pass filter
4. Band-reject filter
5. All-pass filter

Butterworth, Chebyshev, and Cauer filters are some of the most commonly used practical filters that approximate the ideal response. The key characteristic of the Butterworth filter is that it has a flat pass band as well as stop band. For this reason, it is sometimes called a flat-flat filter. The Chebyshev filter has a ripple pass band and flat stop band i.e. the Cauer filter has a ripple pass band and a ripple stop band. Generally, the Cauer filter gives the best stop band response among the three. Because of their simplicity of design, the low-pass and high-pass Butterworth filters are discussed here.

2.1.1 FIRST-ORDER LOW-PASSBUTTER WORTH FILTER

Fig. shows a first-order low-pass Butterworth filter that uses an RC network for filtering. Note that the op- amp is used in the non-inverting configuration; hence it does not load down the RC network. ResistorsR1 and RF determine the gain of the filter. According to the voltage-divider rule, the voltage at the non- inverting terminal (across capacitor C) is

$$V_1 = \frac{-jX_C}{R - jX_C} V_{in}$$

$$j = \sqrt{-1} \; and - jX_C = \frac{1}{j2\pi fC}$$

$$V_1 = \frac{V_{in}}{1 + j2\pi fRC}$$

$$V_o = 1 + \frac{R_F}{R_1} V_1$$

$$V_o = \left(1 + \frac{R_F}{R_1}\right) \frac{V_{in}}{1 + j2\pi fRC}$$

$$\frac{V_o}{V_{in}} = \frac{A_F}{1 + j\left(f / f_H\right)} \quad \text{....eq 3.1}$$

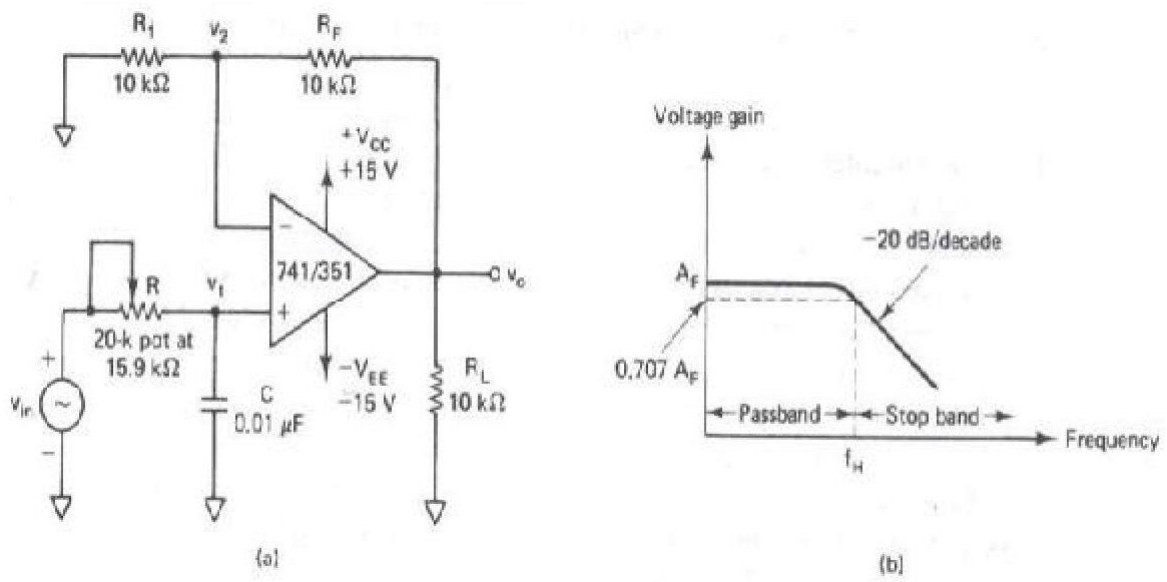


Fig 2.2: First order Low Pass Butter Worth Filter (a)circuit (b)Response

Where Vo/Vin = gain of the filter as a function of frequency

$A_F = \left(1 + \frac{R_F}{R_1}\right)$ = pass band gain of the filter

f = input frequency of the filter

$f_H = \frac{1}{2\pi RC}$ = upper cut-off frequency of the filter.

The gain magnitude and phase angle equations of the low-pass filter can be obtained by converting Equation 3.1into its equivalent polar form, as follows:

$$\left|\frac{V_o}{V_{in}}\right| = \frac{A_F}{\sqrt{1 + (f/f_H)^2}}$$

$$\Phi = -tan^{-1}\left(\frac{f}{f_H}\right)$$

Where φ is the phase angle in degrees.

The operation of the low pass filter can be verified from the gain magnitude equation:

2.1.1.1 At very low frequencies that is , $f<f_H$, $\left|\frac{V_o}{V_{in}}\right| = A_F$

2.1.1.2 At $f = f_H$, $\left|\frac{V_o}{V_{in}}\right| = \frac{A_F}{\sqrt{2}} = 0.707\ A_F$

2.1.1.3 At $f > f_H$, $\left|\frac{V_o}{V_{in}}\right| < A_F$

2.1.2 Filter Design

A low-pass filter can be designed by implementing the following steps:

1. Choose a value of high cutoff frequency f_H.
2. Select a value of C less than or equal to 1 μF. Mylar or tantalum capacitors are recommended for better performance.
3. Calculate the value of R using $R = \frac{1}{2\pi f_H C}$
4. Finally, select values of R_1 and R_F dependent on the desired pass band gain A_F using

$$A_F = 1 + \frac{R_f}{R_1}$$

2.1.3 Frequency Scaling

Once a filter designed; there may sometimes be a need to change its cut-off frequency. The procedure used to convert an original cut-off frequency *fH* to a new cut-off frequency *f'H* is called frequency scaling. Frequency scaling is accomplished as follows. To change a high cutoff frequency, multiple R or C, but not both, by the ratio of the original cutoff frequency to the new cutoff frequency.

2.1.4 SECOND-ORDER LOW-PASSBUTTER WORTH FILTER

A stop-band response having a 40-dB/decade roll-off is obtained with the second order low- pass filter. A first-order low-pass filter can be converted into a second order type simply by using an additional RC network, as shown in Fig.3.3.

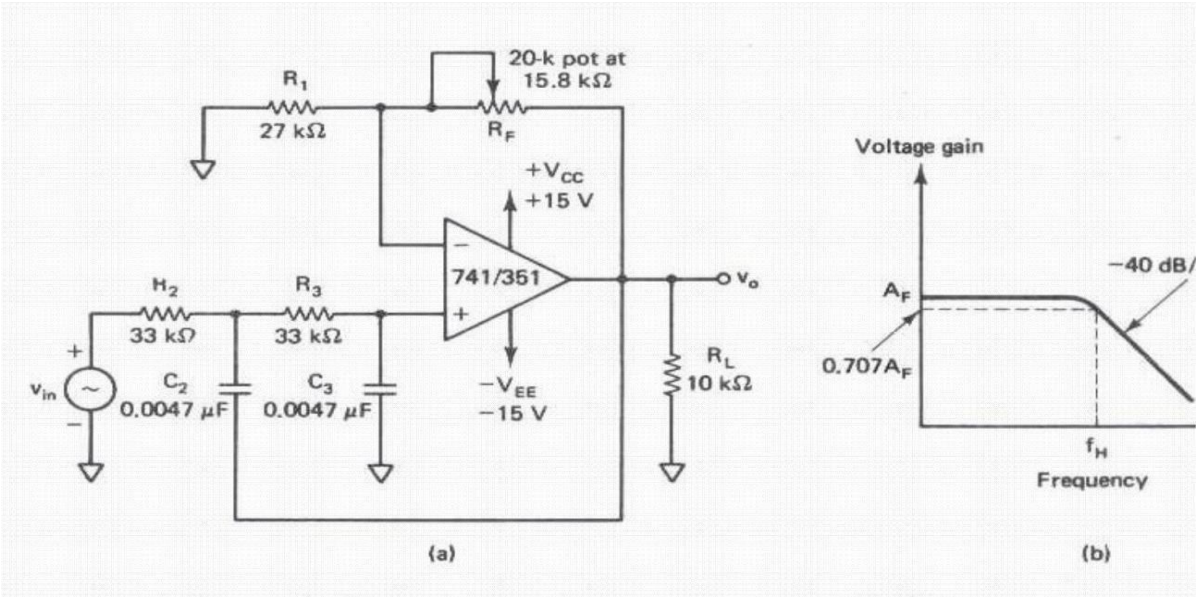


Fig 3.3: Second order Low Pass Butter Worth Filter (a)Circuit(b)Frequency Response

Second-order filters are important because higher-order filters can be designed using them. The gain of the second-order filter is set by R1 and RF, while the high cutoff frequency fH is determined by R2, C2, R3, and C3, as follows

$$f_H = \frac{1}{2\pi\sqrt{R_2R_3C_2C_3}}$$

Furthermore, for a second-order low-pass Butterworth response, the voltage gain magnitude equation is

$$\left|\frac{V_o}{V_{in}}\right| = \frac{A_F}{\sqrt{1+(f/f_H)^4}}$$

Where Vo/Vin = gain of the filter as a function of frequency

$A_F = \left(1+\frac{R_F}{R_1}\right)$ = pass band gain of the filter

f = input frequency of the filter

$f_H = \frac{1}{2\pi\sqrt{R_2R_3C_2C_3}}$ = upper cut-off frequency of the filter

2.2 HIGH PASS FILTER

2.2.1 FIRST-ORDER HIGH-PASSBUITERWORTH FILTER

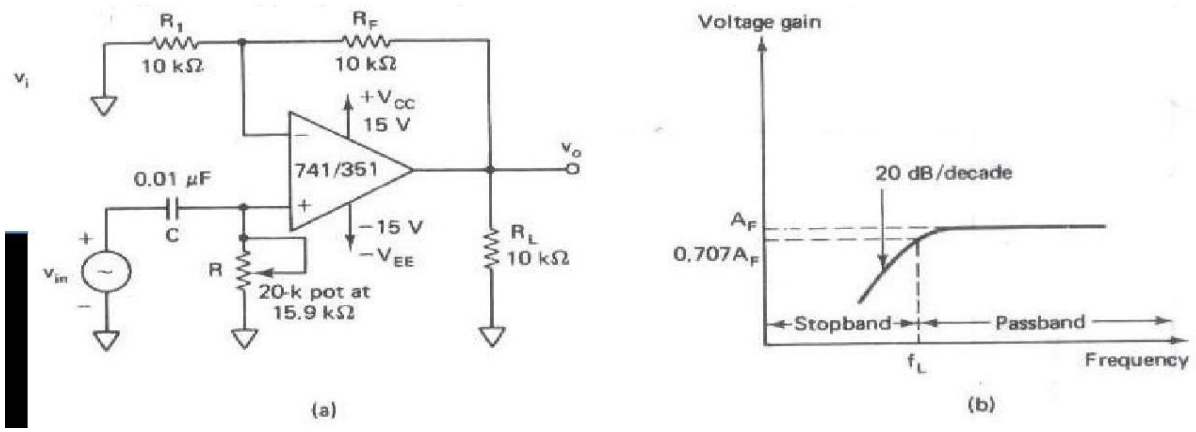


Fig 2.5: (a) First order High Pass Butterworth Filter (b) Frequency Response

High-pass filters are often formed simply by interchanging frequency-determining resistors and capacitors in low-pass filters. That is, a first-order high-pass filter is formed from a first-order low-pass type by interchanging components Rand C.

Similarly, a second-order high-pass filter is obtained from a second-order low-pass filter if R and Care interchanged, and so on. Figure3.4 shows a first-order high-pass Butterworth filter with a low cutoff frequency of fL.

This is the frequency at which the magnitude of the gain is 0.707 times its pass band value. Obviously,

All frequencies higher than *fL* are pass-band frequencies, with the highest frequency determined by the closed- loop bandwidth of the op-amp.

Note that the high-pass filter of Figure 3.4(a) and the low-pass filter of Figure3.4(a) are the same circuits, except that the frequency-determining components (R and C) are interchanged.

For the first-order high-pass filter of Figure 3.4(a), the output voltage is

$$V_o = \left(1 + \frac{R_f}{R_1}\right)\frac{j2\pi fRC}{1 + j2\pi fRC}V_{in}$$

$$\frac{V_o}{V_{in}} = A_F\frac{j(f/f_L)}{\sqrt{1 + j(f/f_L)}}$$

Hence the magnitude of the voltage gain is

$$\left|\frac{V_o}{V_{in}}\right| = A_F\frac{(f/f_L)}{\sqrt{1 + (f/f_L)^2}}$$

2.3 BAND-PASS FILTERS

A band-pass filter has a pass band between two cutoff frequencies *fH* and fL such that *fH>fL*. Any input frequency outside this pass band is attenuated. Basically, there are two types of band-pass filters:

(1) Wide band pass, and

(2) Narrow band pass.

Unfortunately, there is no set dividing line between the two. However, we will define a filter as wideband pass if its figure of merit or quality factor Q<10.On the other hand, if we will call the filter a narrow band-pass filter. Thus Q is a measure of selectivity, meaning the higher the value Q, the more selective is the filter or the narrower its bandwidth (BW). The relationship between Q, the3- dB bandwidth, and the center frequency fc is given by For the wideband-pass filter the center frequency *fc* can be defined as where *fH* =high cut off

frequency (Hz) *fL*=low cut off frequency of the wideband-pass filter (Hz) In a narrowband-pass filter, the output voltage peaks at the center frequency.

2.3.1 Wide-band pass filter

A wide band-pass filter can be formed by simply cascading high-pass and low-pass sections and is generally the choice for simplicity of design and performance. To obtain ±20dB/decade band-pass, first-order high pass and first order low-pass sections are cascaded; fora±40-dB/decade band-pass filter, second-order high- pass and second- order low-pass sections are connected in series. Figure3. 6 showsthe±20-dB/decade wideband pass filter, which is composed of first-order high- pass and first-order low-pass filters. To realize a band-pass response, however, fH must be larger than *fL*.

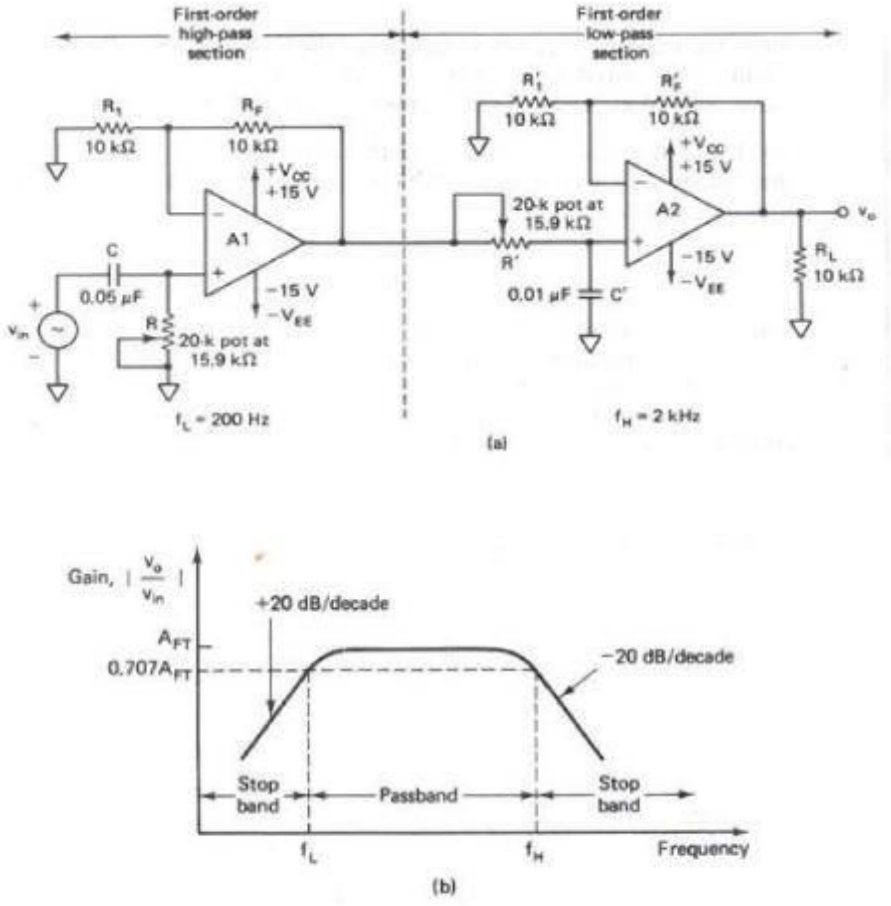


Fig 3.6(a)±20dB/decade Wide Band Pass Filter (b)Frequency Response

Since the band-pass gain is 4, the gain of the high-pass as well as low-pass sections could be set equal to 2. That is, input and feedback resistors must be equal in value, say10 kΩ each. The complete band-pass filter is shown in Fig 3.6(a).(b)The voltage gain magnitude of the

band-pass filter is equal to the product of the voltage gain magnitudes of the high-pass and low-pass filters.

$$\left|\frac{V_o}{V_{in}}\right| = \frac{A_F(f/f_L)}{\sqrt{1+(f/f_L)^2}}$$

$$\left|\frac{V_o}{V_{in}}\right| = \frac{A_{FT}(f/f_L)}{\sqrt{[1+(f/f_L)^2][1+(f/f_H)^2]}}$$

Where *AFT* =total pass band gain

f= frequency of the input signal (Hz)

fL=low cut off frequency(Hz)

fH=high cut off frequency(Hz)

2.3.2 Narrow Band-Pass Filter

The narrow band-pass filter using multiple feedback is shown in Figure8-13. As shown in this figure, the filter uses only one op-amp. Compared to all the filters discussed so far, this filter is unique in the following respects:

2.3.2.1 It has two feedback paths, hence the name multiple-feedback filter.

2.3.2.2 The op-amp is used in the inverting mode.

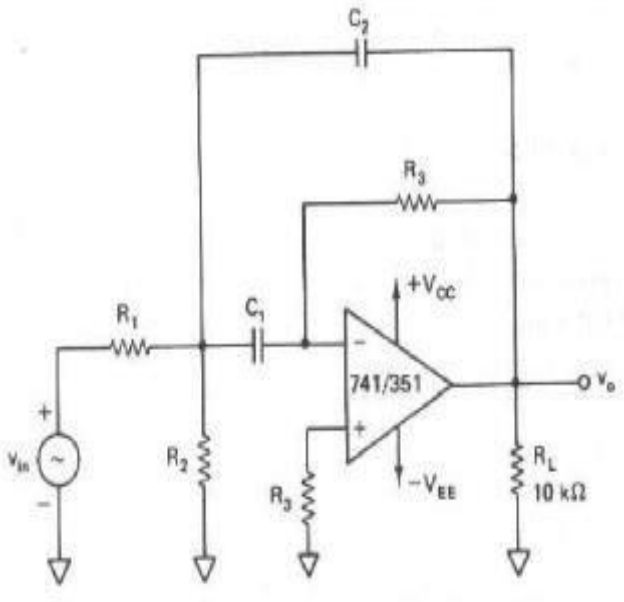


Fig 2.7 Multiple Feedback Narrow Band Pass Filter

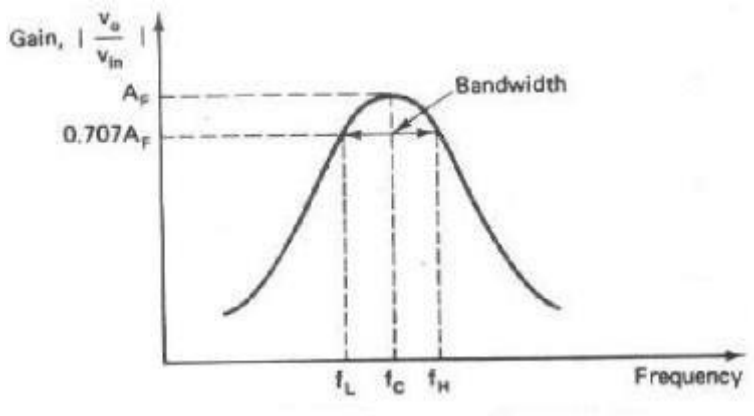


Fig 2.8: (b) Frequency Response

Generally, the narrow band-pass filter is designed for specific values of center frequency *fc* and Q or *fc* and bandwidth. The circuit components are determined from the following relationships. To simplify the design calculations, chooseC1 =C2 =C.

$$R_1 = \frac{Q}{2\pi f_C C A_F}$$

$$R_2 = \frac{Q}{2\pi f_C C(2Q^2 - A_F)}$$

$$R_3 = \frac{Q}{\pi f_C C}$$

Where AF is the gain at *fc*, given by

$$A_F = \frac{R_3}{2R_1}$$

The gain *AF*, however, must satisfy the condition

$$A_F = 2Q^2$$

Another advantage of the multiple feedback filter of Figure8-13 is that its center frequency *fc* can be changed to a new frequency *f'c* without changing the gain or bandwidth. This is accomplished simply by changing R2 to R'2 so that

$$R_2' = R_2\left(\frac{f_C}{f_C'}\right)^2$$

2.4 BAND-REJECT FILTERS

The band-reject filter is also called a band-stop or band-elimination filter. In this filter, frequencies are attenuated in the stop band while they are passed outside this band, as shown in Figure3.1 (d).

As with band-pass filters, the band-reject filters can also be classified as (1)wideband-reject or (2)narrowband-reject. The narrow band-reject filter is commonly called the notch filter. Because of its higher Q (>10), the bandwidth of the narrow band-reject filter is much smaller than that of the wideband- reject filter.

2.4.1 Wide Band-Reject Filter

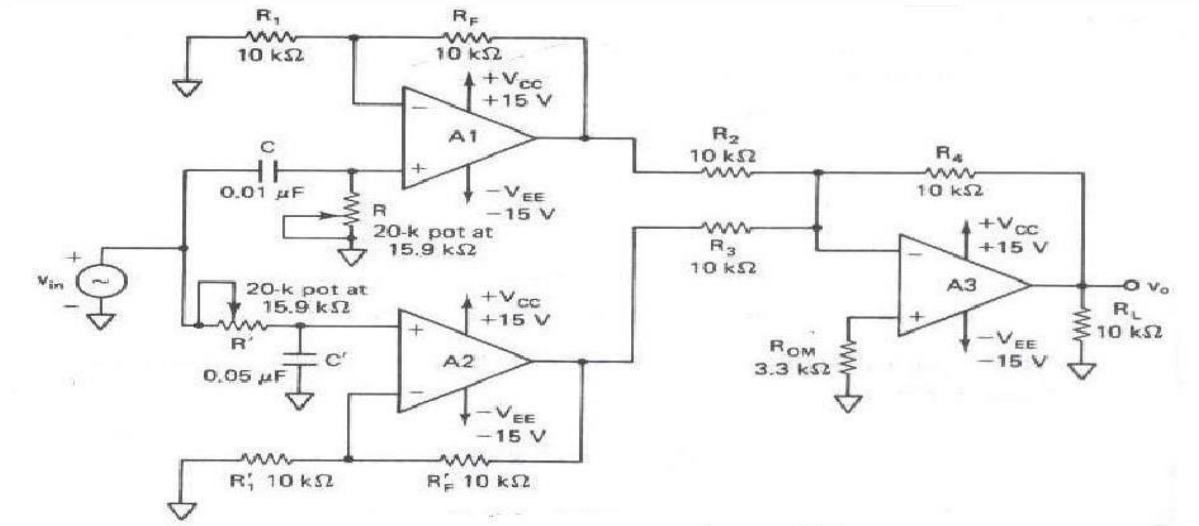


Fig 2.9(a).Wide Band Reject Filter

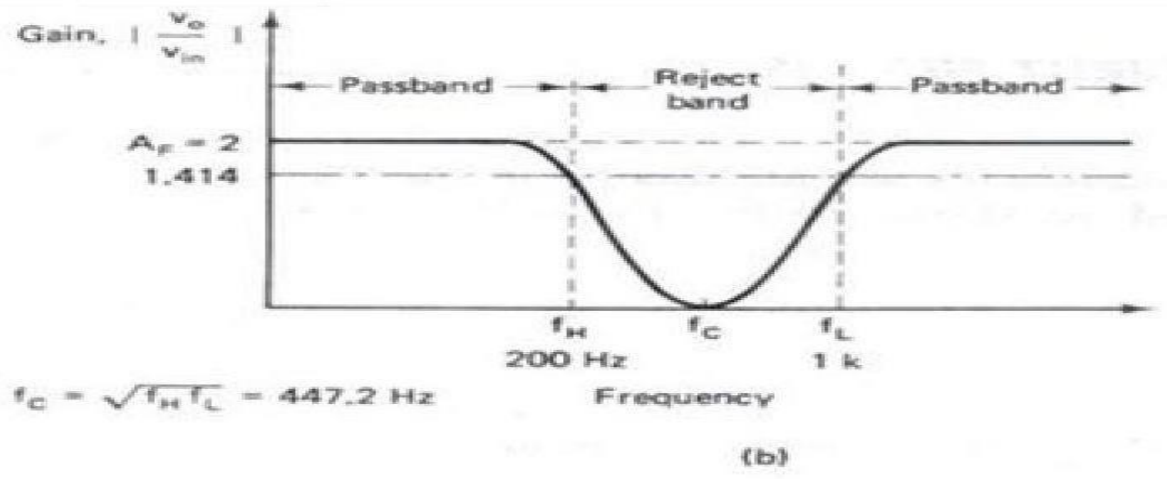


Fig 2.9(b) Frequency Response

Figure 2.9(a) shows a wide band-reject filter using a low-pass filter, a high-pass filter, and assuming amplifier. To realize a band-reject response, the low cut off frequency *fL* of the high- pass filter must be larger than the high cut off frequency *fH* of the low-pass filter. In addition, the pass band gain of both the high-pass and low-pass sections must be equal. The frequency response of the wideband-reject filter is shown in Fig3.9 (b).

2.5 ALL-PASSFILTER

As the name suggests, an all-pass filter passes all frequency components of the input signal without attenuation, while providing predictable phase shifts for different

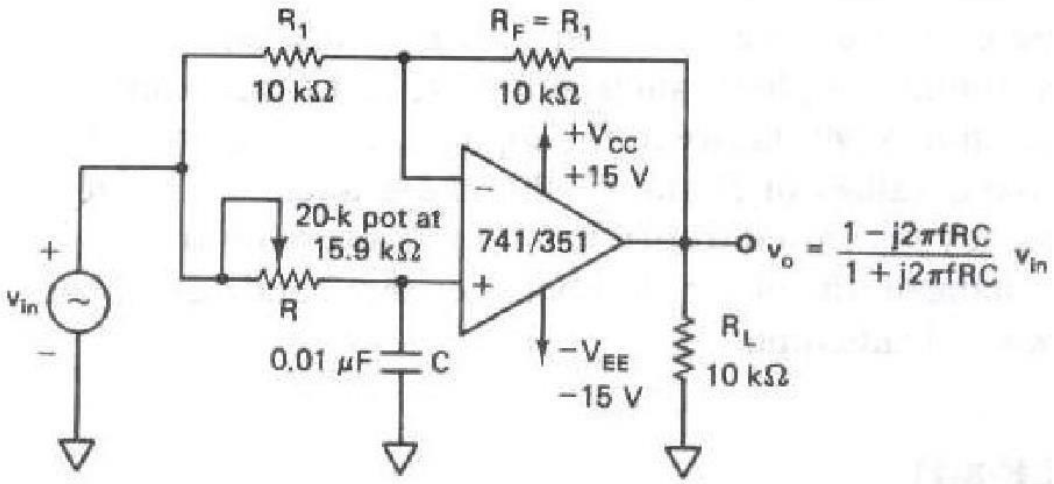


Fig2.11(a)All Pass Filter

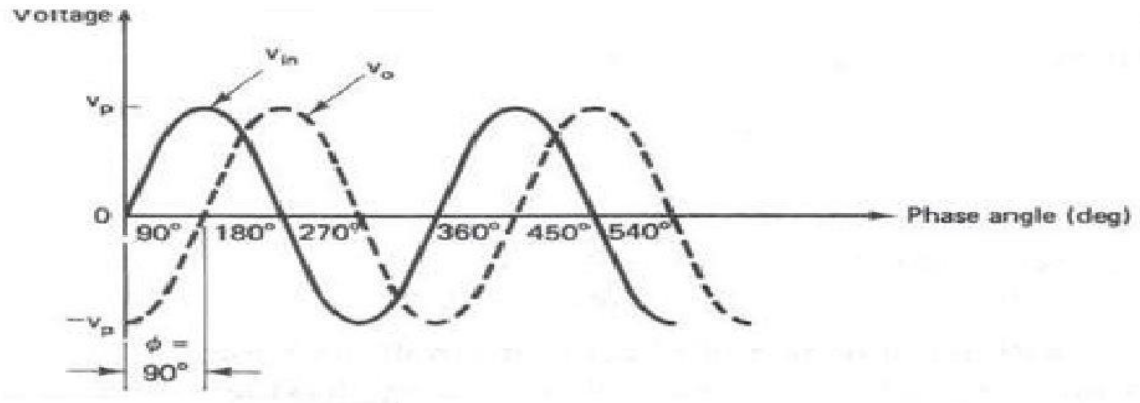


Fig2.11 (b) Phase Shift between Input and Output frequencies of the input signal.

When signals are transmitted over transmission lines, such as telephone wires, they undergo change in phase. To compensate for these phase changes, all-pass filters are required. The all- pass filters are also called delay equalizers or phase correctors. Figure2.11

(a) shows an all-pass filter where in RF =R1. The output voltage Vo of the filter can be obtained by using the superposition theorem:

But -j =1/j and XC =1/2ΠfC. Therefore, substituting for XC and simplifying, we get

$$V_o = V_{in}\left(-1 + \frac{2}{j2\pi fRC + 1}\right)$$

$$\frac{V_o}{V_{in}} = \frac{1 - j2\pi fRC}{1 + j2\pi fRC}$$

Where fis the frequency of the input signal in hertz.

Equation indicates that the amplitude of Vo/Vin is unity; that is, |Vo|=|Vin | throughout the useful frequency range, and the phase shift between Vo and Vin is a function of input frequency f. The phase angle φ is given by $\phi = -2tan^{-1}\left(\frac{2\pi fRC}{1}\right)$

Where φ is in degrees, in hertz, R in ohms, and C in farads. Equation is used to find the phase angle φ if f, R, and C are known. Figure2 .12 (b) shows a phase shift of 90° between the input Vin and output Vo. That is, Vo lags Vin by90°.For fixed values of R and C, the phase angle φ changes from 0to 180°asthe frequency f is varied from 0to∞.InFigure2.12 (a), if the positions of R and C are interchanged, the phase shift between input and output becomes positive. That is, output Vo leads input Vin.

2.6 OSCILLATOR TYPES AND PRINCIPLE OF OPERATION

The use of op-amps as oscillators capable of generating a variety of output waveforms. Basically, the function of an oscillator is to generate alternating current or voltage waveforms. More precisely, an oscillator is a circuit that generates are positive waveform of fixed amplitude and frequency without an y external input signal. Oscillators are used in radio, television, computers, and communications. Although there are different types of oscillators, they all work on the same basic principle.

2.6.1 Oscillator Principle

An oscillator is a type of feedback amplifier in which part of the output is fed back to the input via a feedback circuit. If the signal fed back is of proper magnitude and phase, the circuit produces alternating currents or voltages. To visualize the requirements of an oscillator, consider the block diagram of Figure3.12.

However, here the input voltages zero *(Vin*=0). Also, the feedback is positive because most oscillators use positive feedback. Finally; the closed-loop gain of the amplifier is denoted by *Av* rather than *AF*.

Using these relationships, using these relationships, the following equation is obtained:

$$\frac{V_o}{V_{in}} = \frac{A_V}{1 - A_V\beta}$$ However, Vin =0 andVo≠0 implies that ***Avβ=1***

Equation gives the two requirements for oscillation:

(1) The magnitude of the loop gain ***AvB*** must be at least1, and

(2) The total phase shift of the loop gain ***AvB*** must be equal to 0°or360°.

If the amplifier uses a phase shift of180°, the feedback circuit must provide an additional phase shift of 180° so that the total phase shift around the loop is 360°.Thewaveforms shown in Figure3.13aresinusoidaland are used to illustrate the circuits action.

The type of waveform generated by an oscillator depends on the components in the circuit and hence maybe sinusoidal, square, or triangular; In addition, the frequency of oscillation is determined by the components in the feedback circuit.

OSCILLATOR TYPES

Types of components used	Frequency of oscillation	Types of waveform generated
RC oscillator *LC* oscillator Crystal oscillator	Audio frequency (AF) Radio frequency (RF)	Sinusoidal Square wave Triangular wave Sawtooth wave, etc.

2.6.2 RC-PHASE SHIFT OSCILLATOR

Figure3.13 shows a phase shift oscillator, which consists of an op-amp as the amplifying stage and three RC cascaded networks as the feedback circuit. The feedback circuit provides feedback voltage from the output back to the input of the amplifier. The op-amp is used in the inverting mode; therefore, any signal that appears at the inverting terminal is shifted by 180° at the output.

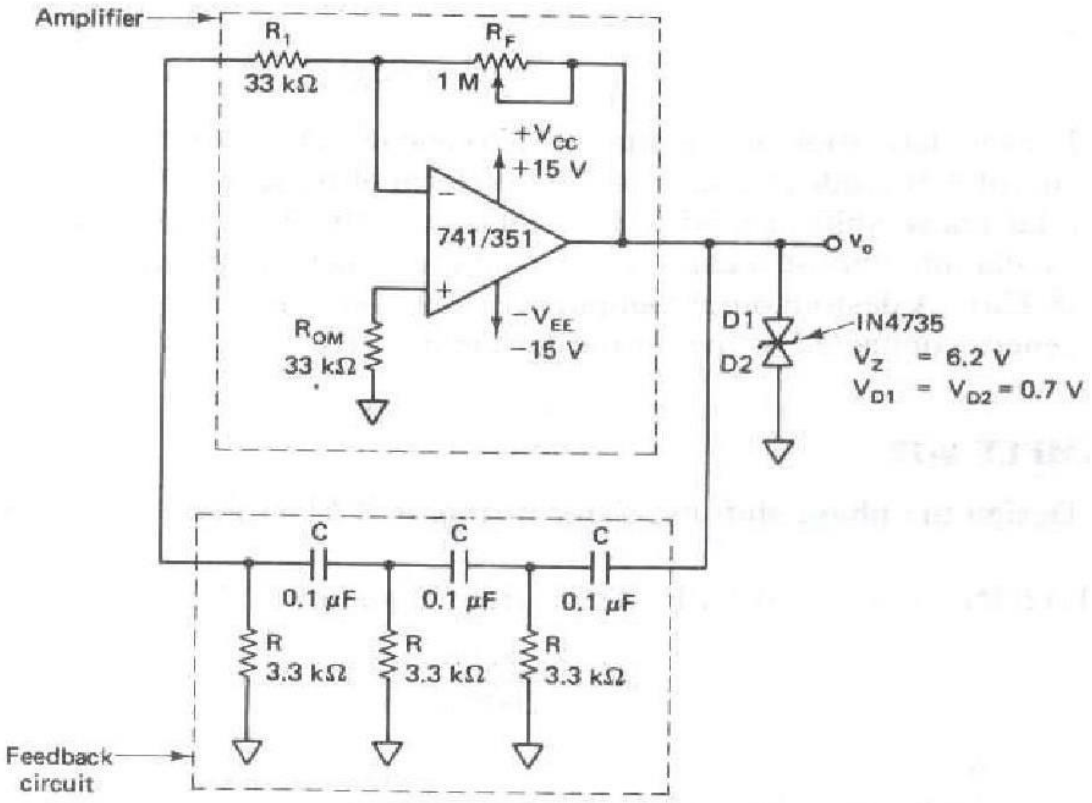


Fig 2.13 RC phase shift Oscillator

An additional 180° phase shift required or oscillation is provided by the cascaded RC networks. Thus the total phase shift around the loop is 360°(or0°). At some specific frequency when the phase shift of the cascaded RC networks is exactly 180°and the gain of the amplifier is sufficiently large, the circuit will oscillate at that frequency. This frequency is called the frequency of oscillation *fo* and is given by

$$f_o = \frac{1}{2\pi\sqrt{6}RC} = \frac{0.065}{RC}$$

At this frequency, the gain *Av* must be at least29.That is,

$$\left|\frac{R_F}{R_1}\right| = 29$$

2.6.3 WIEN BRIDGE OSCILLATOR

Because of its simplicity and stability, one of the most commonly used audio-frequency oscillators is the Wien bridge. Figure3.14 shows the Wien bridge oscillator in which the Wien bridge circuit is connected between the amplifier input terminals and the output terminal. The bridge as a series RC network in one arm and a parallel RC network in the adjoining arm. In

the remaining two arms of the bridge, resistorsR1 and RF, are connected. The phase angle criterion for oscillation is that the total phase shift around the circuitmustbe00.Thiscondition occurs only when the bridge is balanced, that is, at resonance. The frequency of oscillation f0 is exactly the resonant frequency of the balanced Wien bridge and is given by

$$f_o = \frac{1}{2\pi RC} = \frac{0.159}{RC}$$

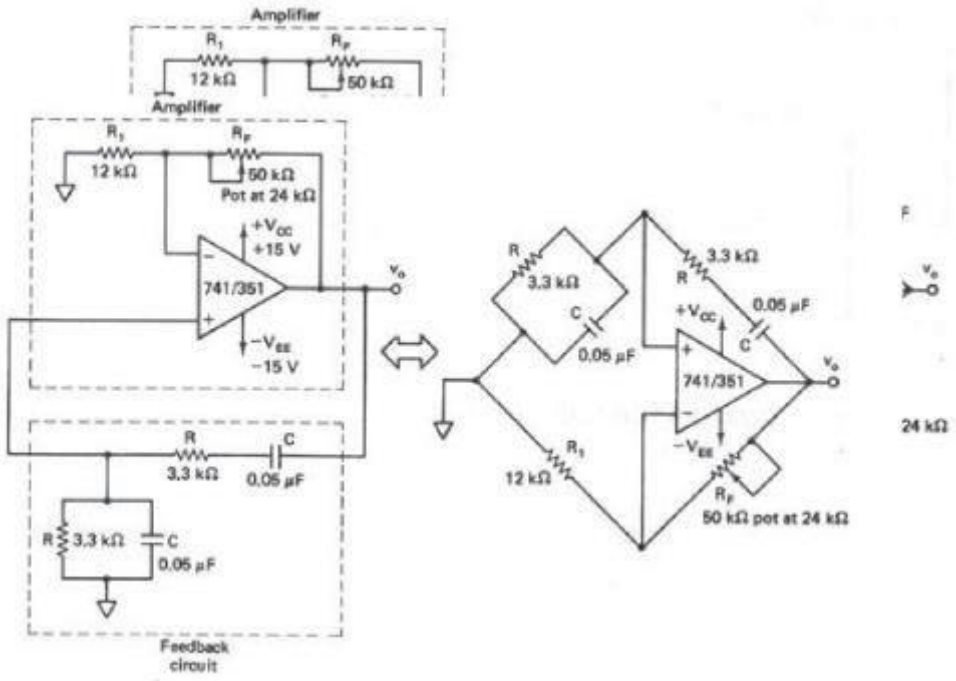


Fig2.14Wien Bridge

Assuming that the resistors are equal in value, and capacitors are equal in value in the reactive leg of the Wien Bridge. At this frequency the gain required for sustained oscillation is given by

$$A_V = \frac{1}{\beta} = 3$$

$$1 + \frac{R_F}{R_1} = 3$$

$$R_F = 2R_1$$

2.7 INTRODUCTION TO 555 TIMER:

One of the most versatile linear integrated circuits is the *555* timer. A sample of these

applications includes mono-stable and astable multivibrators, dc-dc converters, digital logic probes, waveform generators, analog frequency meters and tachometers, temperature measurement and control, infrared transmitters, burglar and toxic gas alarms, voltage regulators, electric eyes, and many others.

The 555 is a monolithic timing circuit that can produce accurate and highly stable time delays or oscillation. The timer basically operates in one of the two modes: either as monostable (one-shot) multivibrator or as an astable (free running) multivibrator. The device is available as an 8-pin metal can, an 8-pin mini DIP, or a 14-pin DIP.

The SE555 is designed for the operating temperature range from -55°Cto + 125°C, while the NE555 operates over a temperature range of 0° to +70°C. The important features of the 555 timer are these: it operates on +5 to + 18 V supply voltage in both free-running (astable) and one- shot (monostable) modes; it has an adjustable duty cycle; timing is from microseconds through hours; it has a high current output; it can source or sink 200 mA; the output can drive TTL and has a temperature stability of 50 parts per million (ppm) per degree Celsius change in temperature, or equivalently 0.005%/°C.

Like general-purpose op-amps, the 555 timer is reliable, easy to us, and low cost.

Pin 1: Ground.

All voltages are measured with respect to this terminal.

Pin 2: Trigger.

The output of the timer depends on the amplitude of the external trigger pulse applied to this pin. The output is low if the voltage at this pin is greater than 2/3 VCC. However,when a negative-going pulse of amplitude larger than 1/3 VCC is applied to this pin, the comparator 2 output goes low, which in turn switches the output of the timer high. The output remains high as long as the trigger terminal is held at a low voltage.

Pin 3: Output.

There are two ways a load can be connected to the output terminal: either between pin 3 and ground (pin 1) or between pin 3 and supply voltage + VCC (pin 8). When the output is low, the load current flows through the load connected between pin 3 and + VCC into the output terminal and is called the sink current.

However, the current through the grounded load is zero when the output is low. For this reason, the load connected between pin 3 and + VCC is called the normally on load and that connected between pin 3 and ground is called the normally off load.

On the other hand, when the output is high, the current through the load connected between pin 3and + VCC (normally on load) is zero. However, the output terminal supplies current to the normally off load. This current is called the source current. The maximum value of sink or source current is 200 mA.

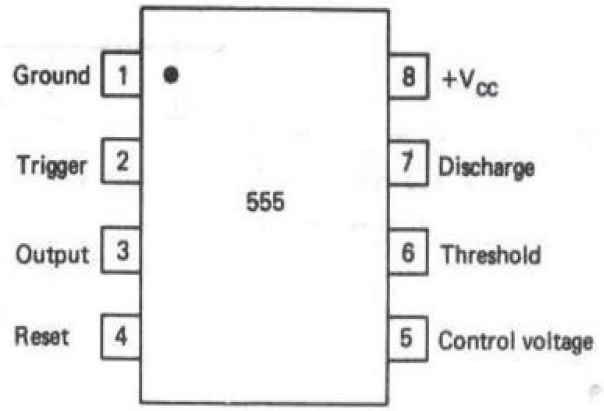


Fig 2.1: Pin diagram of 555Timer

Pin 4: Reset.

The 555 timer can be reset (disabled) by applying a negative pulse to this pin. When the reset function is not in use, the reset terminal should be connected to + VCC to avoid any possibility of false triggering.

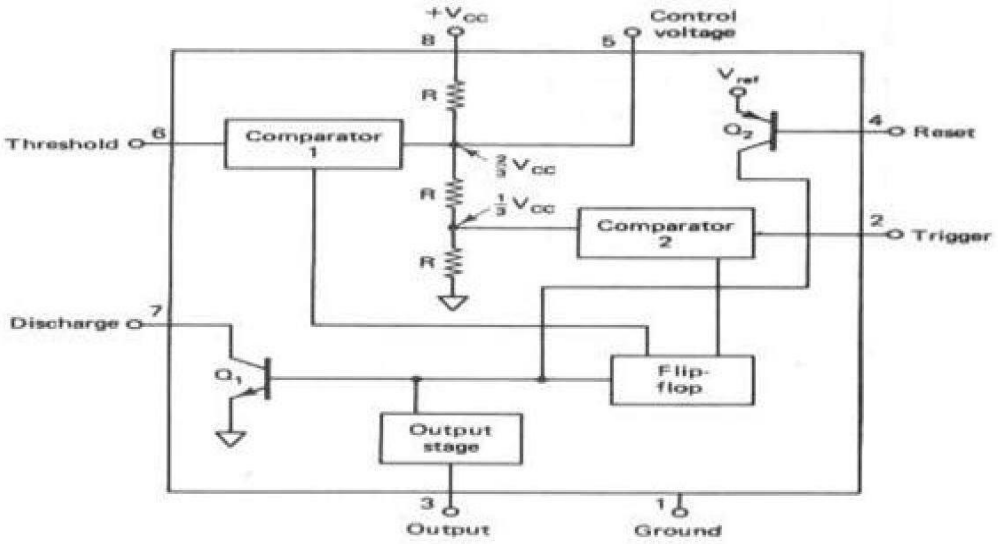


Fig 2.2: Block Diagram

Pin 5: Control voltage.

An external voltage applied to this terminal changes the threshold as well as the trigger voltage . In other words, by imposing a voltage on this pin or by connecting a pot between this pin and ground, the pulse width of the output waveform can be varied. When not

used, the control pin should be bypassed to ground with a 0.01-μF capacitor to prevent any noise problems.

Pin 6: Threshold. This is the non-inverting input terminal of comparator 1, which monitors the voltage across the external capacitor. When the voltage at this pin is threshold voltage 2/3 V, the output of comparator 1 goes high, which in turn switches the output of the timer low.

Pin 7: Discharge. This pin is connected internally to the collector of transistor Q1, as shown in Figure 2.1(b). When the output is high, Q1 is off and acts as an open circuit to the external capacitor C connected across it. On the other hand, when the output is low, Q1 is saturated and acts as a short circuit, shorting out the external capacitor C to ground.

Pin 8: + VCC.

The supply voltage of +5 V to +18 is applied to this pin with respect to ground (pin 1).

2.8 FUNCTIONAL BLOCK DIAGRAM OF 555 TIMER:

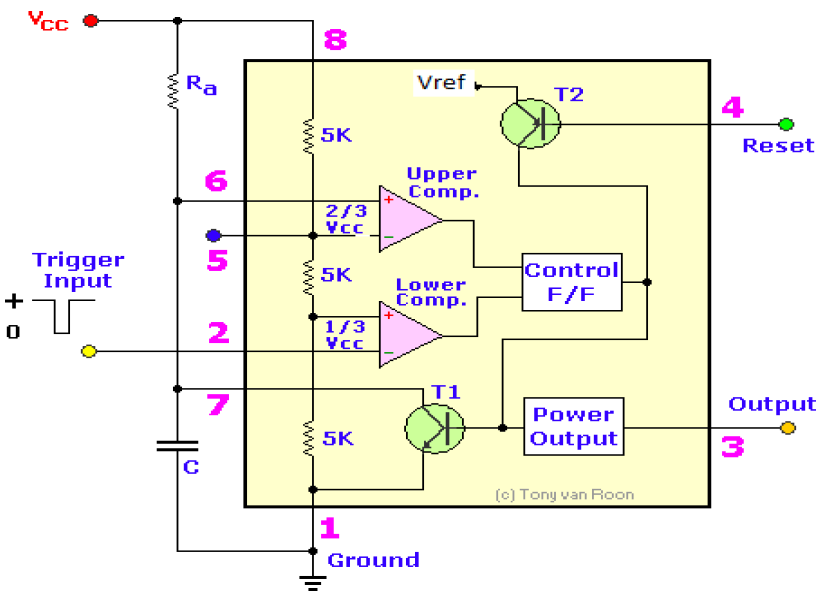


Fig 2.3: Block diagram of timer

2.9 THE 555 AS A MONOSTABLE MULTIVIBRATOR

A monostable multivibrator, often called a one-shot multivibrator, is a pulse-generating circuit in which the duration of the pulse is determined by the RC network connected externally to the 555 timers.

In a stable or standby state, the output of the circuit is approximately zero or at logic-low level. When an external trigger pulse is applied, the output is forced to go high (≈VCC). The time the output remains high is determined by the external RC network connected to the

timer. At the end of the timing interval, the output automatically reverts back to its logic-low stable state. The output stays low until the trigger pulse is again applied. Then the cycle repeats.

The monostable circuit has only one stable state (output low), hence the name mono-stable. Normally, the output of the mono- stable multivibrator is low. Fig 2.2 (a) shows the 555 configured for monostable operation. To better explain the circuit's operation, the internal block diagram is included in Fig 2.2(b).

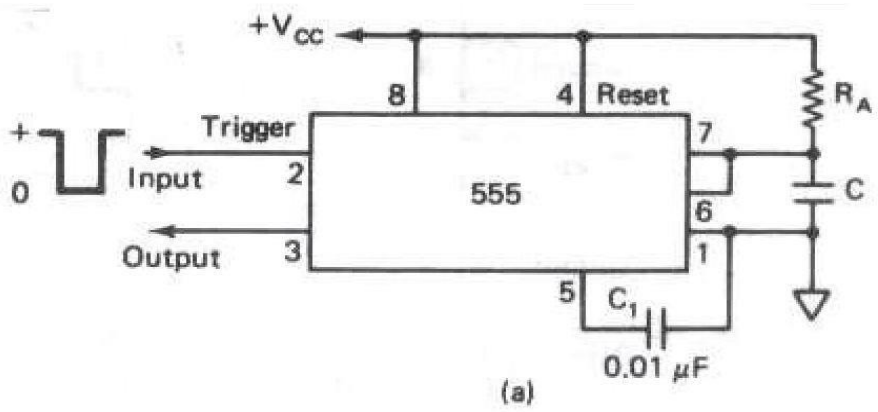


Fig 2.4: IC555 as monostable multivibrator

Mono-stable operation:

According to Fig 2.2(b), initially when the output is low, that is, the circuit is in a stable state, transistor Q is on and capacitor C is shorted out to ground. However, upon application of a negative trigger pulse to pin 2, transistor Q is turned off, which releases the short circuit across the external capacitor C and drives the output high. The capacitor C now starts charging up toward Vcc through RA.

However, when the voltage across the capacitor equals 2/3 Va., comparator I 's output switches from low to high, which in turn drives the output to its low state via the output of the flip-flop. At the same time, the output of the flip-flop turns transistor Q on, and hence capacitor C rapidly discharges through the transistor.

The output of the monostable remains low until a trigger pulse is again applied. Then the cycle repeats. Figure 4-2(c) shows the trigger input, output voltage, and capacitor voltage waveforms. As shown here, the pulse width of the trigger input must be smaller than the expected pulse width of the output waveform. Also, the trigger pulse must be a negative-going input signal with amplitude larger than 1/3 the time during which the output remains high is given by where

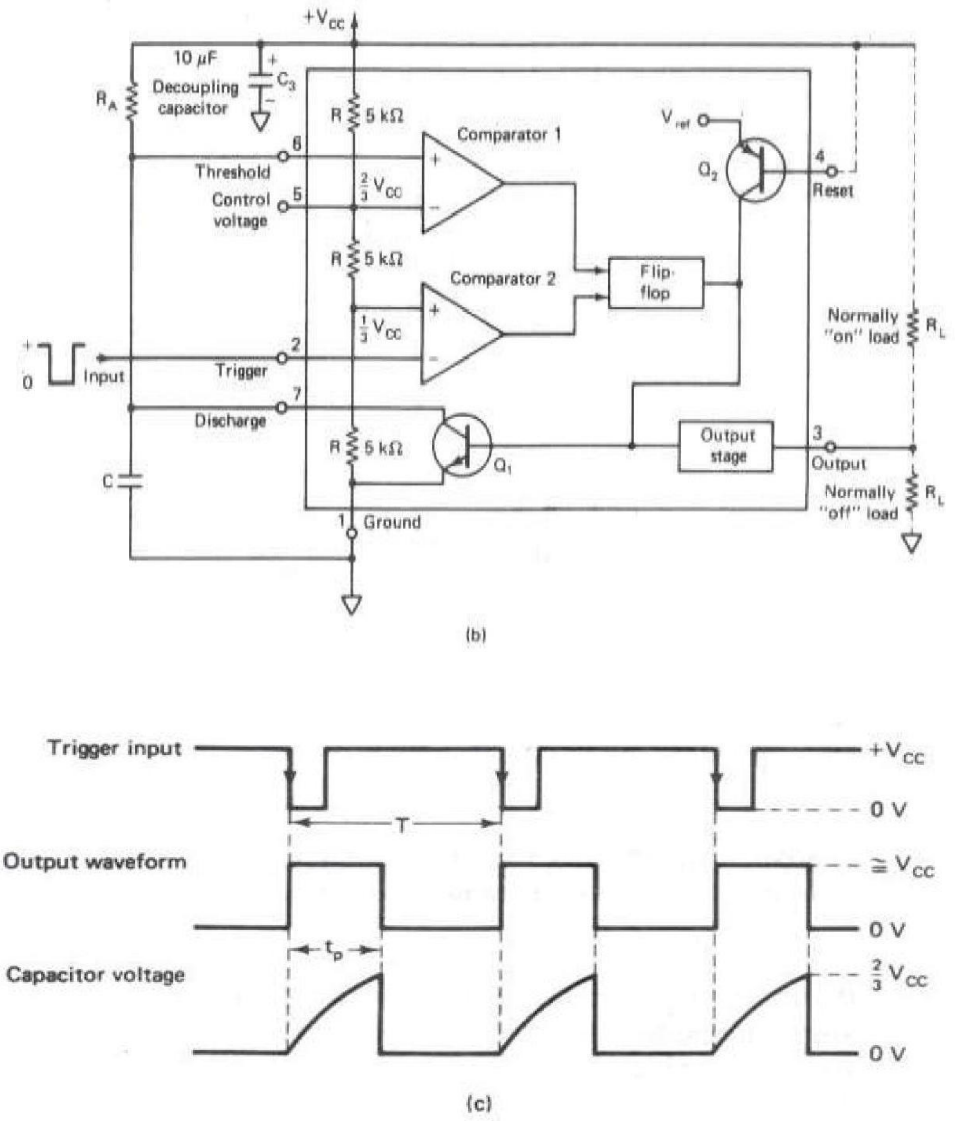


Fig.2.5 (b) 555 connected as a Monostable Multivibrator (c) input and output waveforms

Where RA is in ohms and C is in farads. Figure 2.2(c) shows a graph of the various combinations of RA and C necessary to produce desired time delays. Note that this graph can only be used as a guideline and gives only the approximate value of RA and C for a given time delay. Once triggered, the circuit's output will remain in the high state until the set time 1, elapses. The output will not change its state even if an input trigger is applied again during this time interval T. However, the circuit can be reset during the timing cycle by applying a negative pulse to the reset terminal. The output will then remain in the low state until a trigger is again applied.

Often in practice a decoupling capacitor (10 F) is used between + (pin 8) and ground (pin 1) to eliminate unwanted voltage spikes in the output waveform. Sometimes, to prevent

any possibility of mistriggering the monostable multivibrator on positive pulse edges, a wave shapingcircuit consisting of R, C2, and diode D is connected between the trigger input pin 2 and pin 8, as shown in Figure 4-3. The values of R and C2 should be selected so that the time constant RC2 is smaller than the output pulse width.

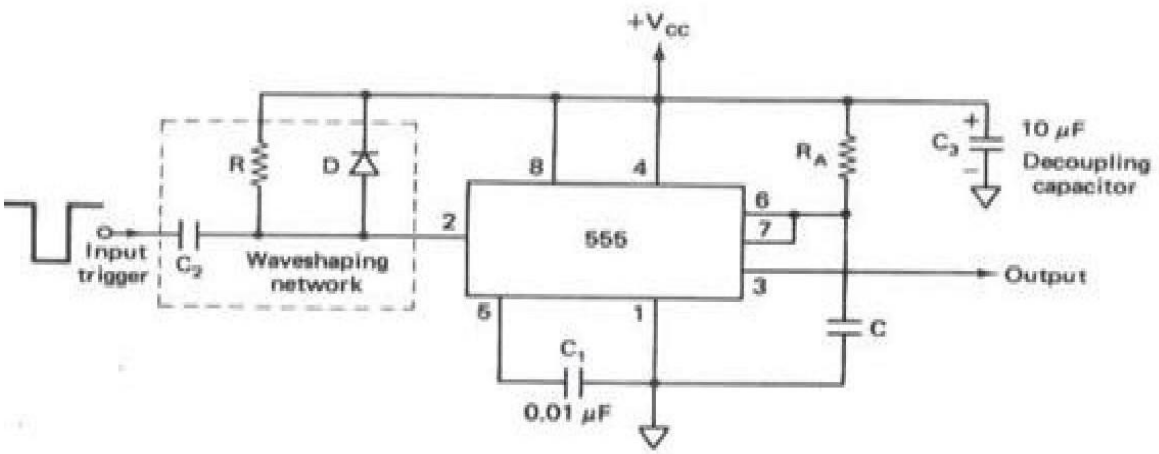


Fig.2.6: Monostable Multivibrator with wave shaping network to prevent +ve pulse edge triggering

Monostable Multivibrator Applications

(a) Frequency divider: The monostable multivibrator of Figure 2.2(a) can be used as a frequency divider by adjusting the length of the timing cycle tp, with respect to the tine period T of the trigger input signal applied to pin 2. To use monostable multivibrator as a divide-by-2 circuit, the timing interval tp must be slightly larger than the time period T of the trigger input signal, as shown in Figure 2.4. By the same concept, to use the monostable multivibrator as a divide-by-3 circuit, tp must be slightly larger than twice the period of the input trigger signal, and so on. The frequency-divider application is possible because the monostable multivibrator cannot be triggered during the timing cycle.

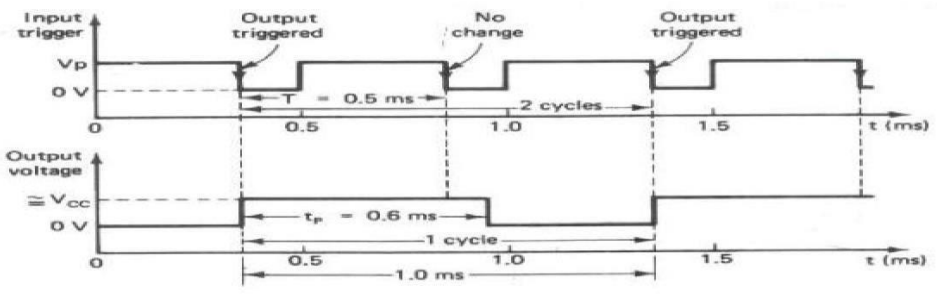


Fig 2.7 input and output waveforms of a monostable multi vibrator as a divide-by-2 network

(b) Pulse stretcher: This application makes use of the fact that the output pulse width (timing interval) of the rnonostable multivibrator is of longer duration than the negative pulse width of the input trigger. As such, the output pulse width of the monostable multivibrator can be viewed as a stretched version of the narrow input pulse, hence the name pulse stretcher. Often, narrow-pulse- width signals are not suitable for driving an LED display, mainly because of their very narrow pulse widths. In other words, the LED may be flashing but is not visible to the eye because its on time is infinitesimally small compared to its off time. The 555 pulse stretcher can be used to remedy this problem

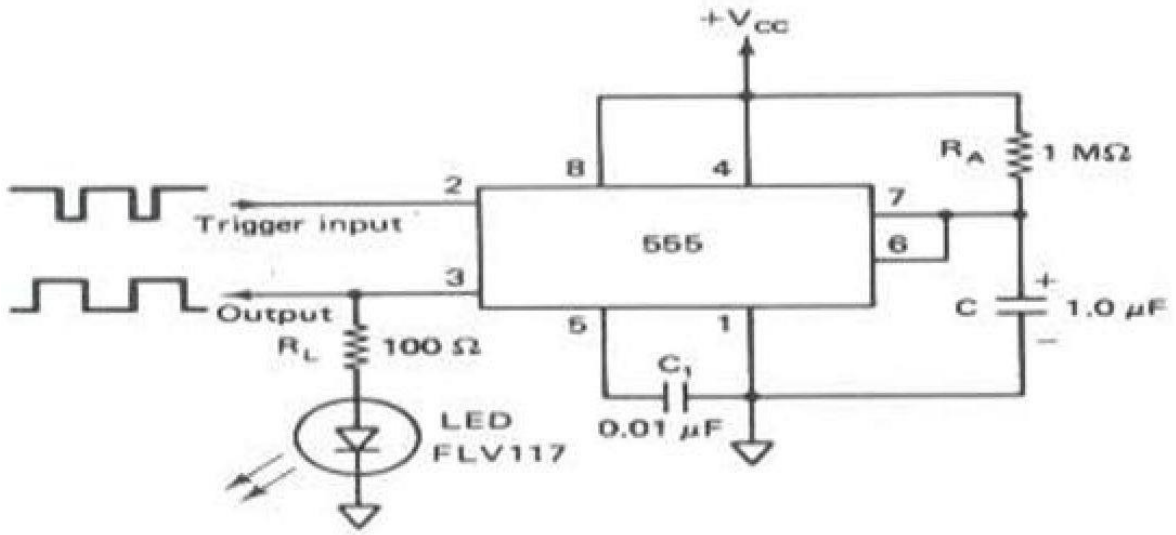


Fig 2.8 Monostable multi vibrator as a Pulse stretcher

Figure 2.8 shows a basic monostable used as a pulse stretcher with an LED indicator at the output. The LED will be on during the timing interval tp = 1.1RAC, which can be varied by changing the value of RA and/or C.

2.10 THE 555 AS AN ASTABLE MULTIVIBRATOR:

The 555 as an Astable Multivibrator, often called a free-running multivibrator, is a rectangular- wave-generating circuit. Unlike the monostable multivibrator, this circuit does not require an external trigger to change the state of the output, hence the name free running. However, the time during which the output is either high or low is determined by the two resistors and a capacitor, which are externally connected to the 555 timer. Fig 4-6(a) shows the 555 timer connected as an astable multivibrator. Initially, when the output is high, capacitor C starts charging toward V through RA and R8. However as soon as voltage across the capacitor equals 2/3 Vcc, comparator I triggers the flip flop, and the output switches low. Now capacitor C starts discharging through R8 and transistor Q. When the voltage across C

equals 1/3 comparator 2's output triggers the flip-flop, and the output goes high. Then the cycle repeats.

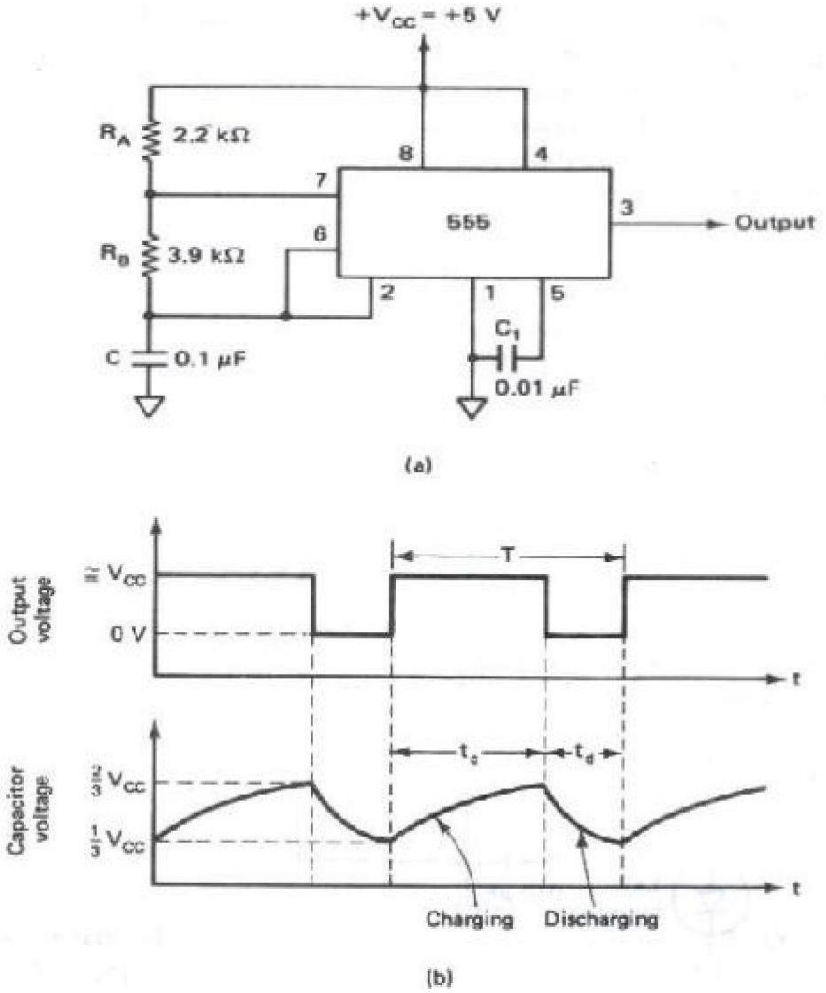


Fig 2.9: The 555 as a Astable Multivibrator (a)Circuit(b)Voltage across Capacitor and O/P waveforms.

The output voltage and capacitor voltage waveforms are shown in Figure 2.6(b). As shown in this figure, the capacitor is periodically charged and discharged between 2/3 Vcc and 1/3 V, respectively. The time during which the capacitor charges from 1/3 V to 2/3 V. is equal to the time the output is high and is given by

$$t_C = 0.69(R_A + R_B)C$$

where RA and R3 are in ohms and C is in farads. Similarly, the time during which the capacitor discharges from 2/3 V to 1/3 V is equal to the time the output is low and is given by

$$t_d = 0.69(R_B)C$$

where RB is in ohms and C is in farads. Thus the total period of the output waveform is

$$T = t_C + t_d = 0.69(R_A + 2R_B)C$$

This, in turn, gives the frequency of oscillation as

$$f_o = \frac{1}{T} = \frac{1.45}{(R_A + 2R_B)C}$$

Above equation indicates that the frequency fo is independent of the supply voltage V. Often the term duty cycle is used in conjunction with the astable multivibrator . The duty cycle is the ratio of the time t during which the output is high to the total time period T. It is generally expressed as a percentage. In equation form,

$$\%\ duty\ cycle = \frac{t_C}{T} \times 100$$

$$= \frac{R_A + 2R_B}{R_A + 2R_B} \times 100$$

Astable Multivibrator Applications:

Square-wave oscillator: Without reducing RA = 0 , the astable multivibrator can be used to produce a square wave output simply by connecting diode D across resistor RB, as shown in Figure 4-7. The capacitor C charges through RA and diode D to approximately 2/3 Vcc and discharges through RB and terminal 7 until the capacitor voltage equals approximately 1/3 Vcc; then the cycle repeats. To obtain a square wave output (50% duty cycle), RA must be a combination of a fixed resistor and potentiometer so that the potentiorneter can be adjusted for the exact square wave.

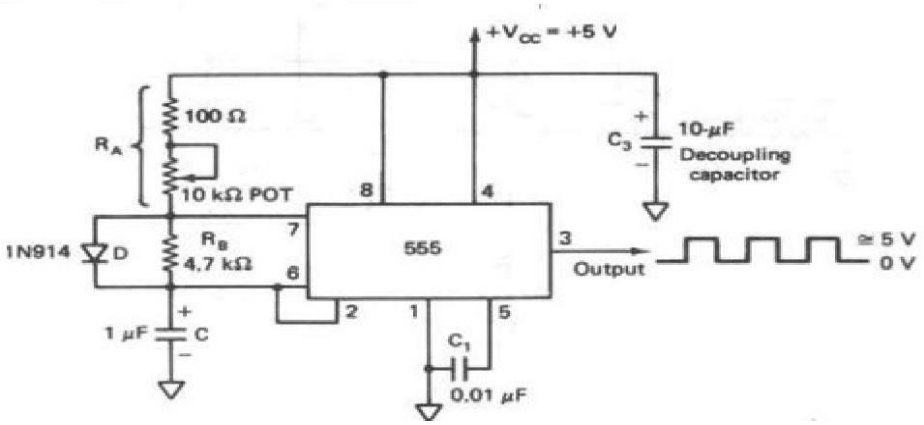


Fig 2.10: Astable Multivibrator as a Square wave generator

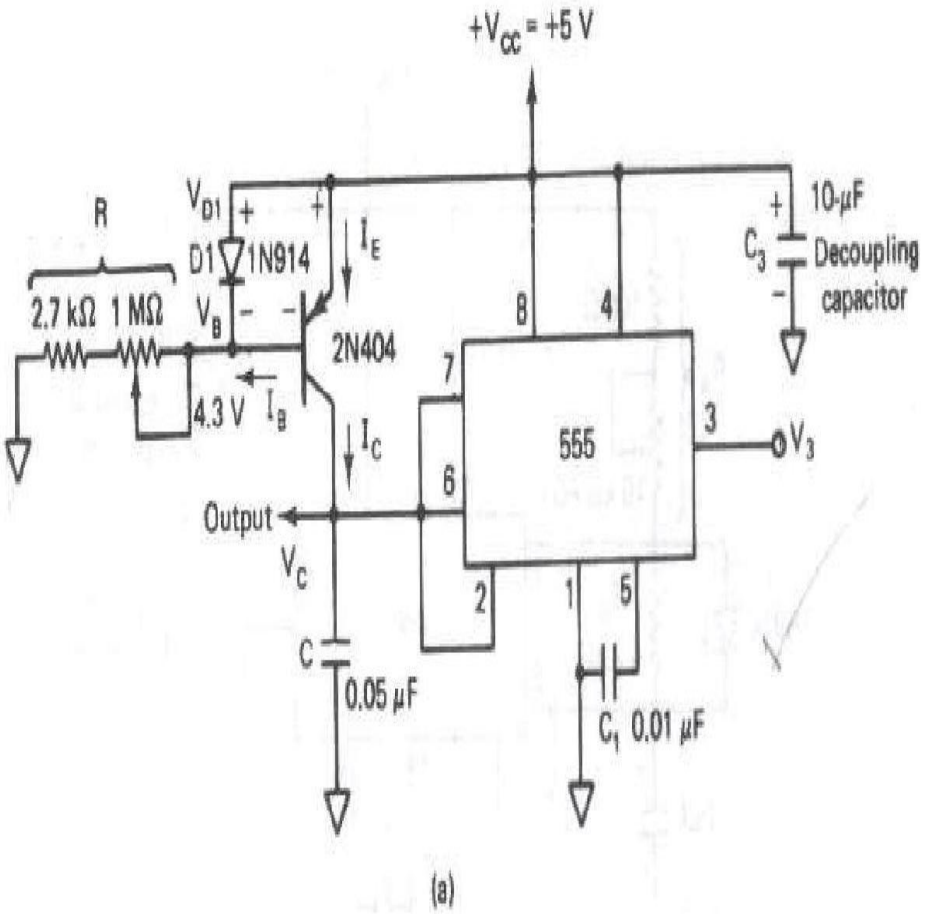
Free-running ramp generator: The astable multivibrator can be used as a free-running ramp generator when resistors RA and R3 are replaced by a current mirror. Figure 2.8(a) shows an astable multivibrator configured to perform this function. The current mirror starts charging capacitor C toward Vcc at a constant rate.

When voltage across C equals 2/3 Vcc, comparator 1 turns transistor Q on, and C rapidly discharges through transistor Q. However, when the discharge voltage across C is approximately equal to 1/3 Vcc, comparator 2 switches transistor Q off, and then capacitor C starts charging up again. Thus the charge— discharge cycle keeps repeating. The discharging time of the capacitor is relatively negligible compared to its charging time; hence, for all practical purposes, the time period of the ramp waveform is equal to the charging time and is approximately given by

$$T = \frac{V_{cc}C}{3I_C}$$

Where I = (Vcc — VBE)/R = constant current in amperes and C is in farads. Therefore, the free running frequency of the ramp generator is

$$f_o = \frac{3I_C}{V_{cc}C}$$



(a)

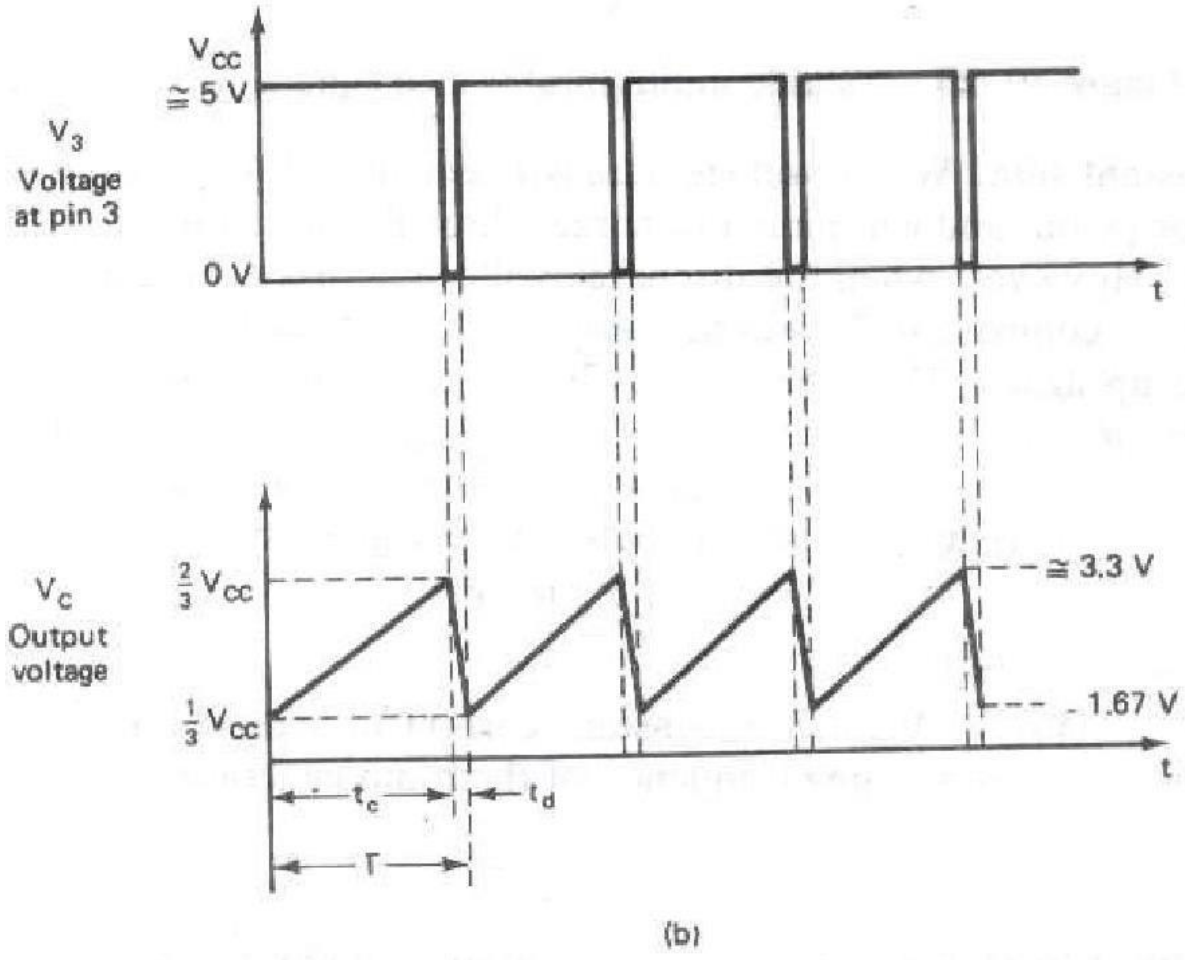


Fig 2.11: (a) Free Running ramp generator (b) Output waveform.

UNIT-III

PHASE LOCKED LOOPS, D/A AND A/D CONVERTERS, CMOS LOGIC

3.1 PHASE-LOCKED LOOPS

The phase-locked loop principle has been used in applications such as FM (frequency modulation) stereo decoders, motor speed controls, tracking filters, frequency synthesized transmitters and receivers, FM demodulators, frequency shift keying (FSK) decoders, and a generation of local oscillator frequencies in TV and in FM tuners.

Today the phase-locked loop is even available as a single package, typical examples of which include the Signetics SE/NE 560 series (the 560, 561, 562, 564, 565, and 567). However, for more economical operation, discrete ICs can be used to construct a phase-locked loop.

3.1.1 Bloch Schematic and Operating Principle

Figure 2.10 shows the phase-locked loop (PLL) in its basic form. As illustrated in this figure, the phase-locked loop consists of (1) a phase detector, (2) a low-pass filter, and, (3) a voltage controlled oscillator

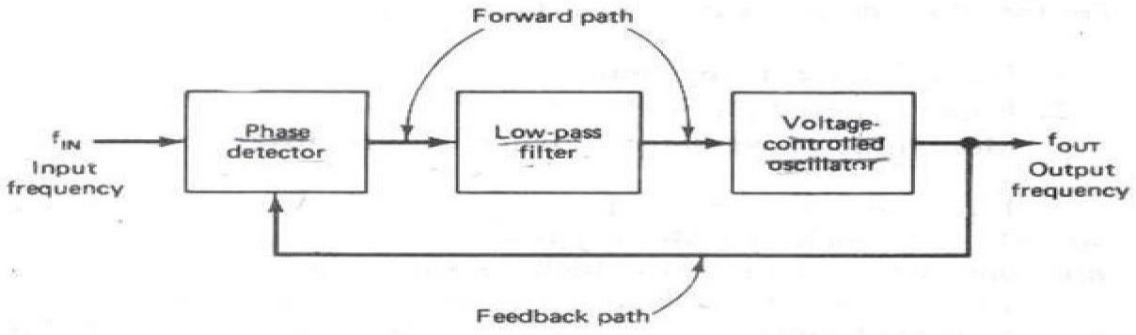


Fig 3.1: Block Diagram of Phase Locked Loop

The phase detectors or comparator compares the input frequency *f*IN with the feedback frequency *f*OUT.. The output voltage of the phase detector is a dc voltage and therefore, is often referred to as the error voltage. The output of the phase is then applied to the low-pass filter, which removes the high-frequency noise and produces a dc level.

This dc level, in turn, is the input to the voltage-controlled oscillator (VCO). The filter also helps in establishing the dynamic characteristics of the PLL circuit. The output frequency of the VCO is directly proportional to the input dc level. The VCO frequency is compared with the input frequencies and adjusted until it is equal to the input frequencies. In short, the phase-locked loop goes through three states: free- running, capture, and phase lock. Before the input is applied, the phase-locked loop is in the free-running state. Once the input frequency is applied, the VCO frequency starts to change and the phase-locked loop is said to be in the capture mode. The VCO frequency continues to change until it equals the input frequency, and the phase- locked loop is then in the phase-locked state. When phase locked, the loop tracks any change in the input frequency through its repetitive action. Before studying the specialized phase-locked-loop IC, we shall consider the discrete phase-locked loop, which may be assembled by combining a phase detector, a low-pass filter, and a voltage-controlled oscillator.

(a) Phase detector:

The phase detector compares the input frequency and the VCO frequency and generates a dc voltage that is proportional to the phase difference between the two frequencies. Depending on the analog or digital phase detector used, the PLL is either called an analog or digital type, respectively. Even though most of the monolithic PLL integrated circuits use analog phase detectors, the majority of discrete phase detectors in use are of the digital type mainly because of its simplicity.

A double-balanced mixer is a classic example of an analog phase detector. On the other hand,examples of digital phase detectors are these:

1. Exclusive-OR phase detector
2. Edge-triggered phase detector
3. Monolithic phase detector (such as type 4044)

The following fig 2.11 shows Exclusive-OR phase detector:

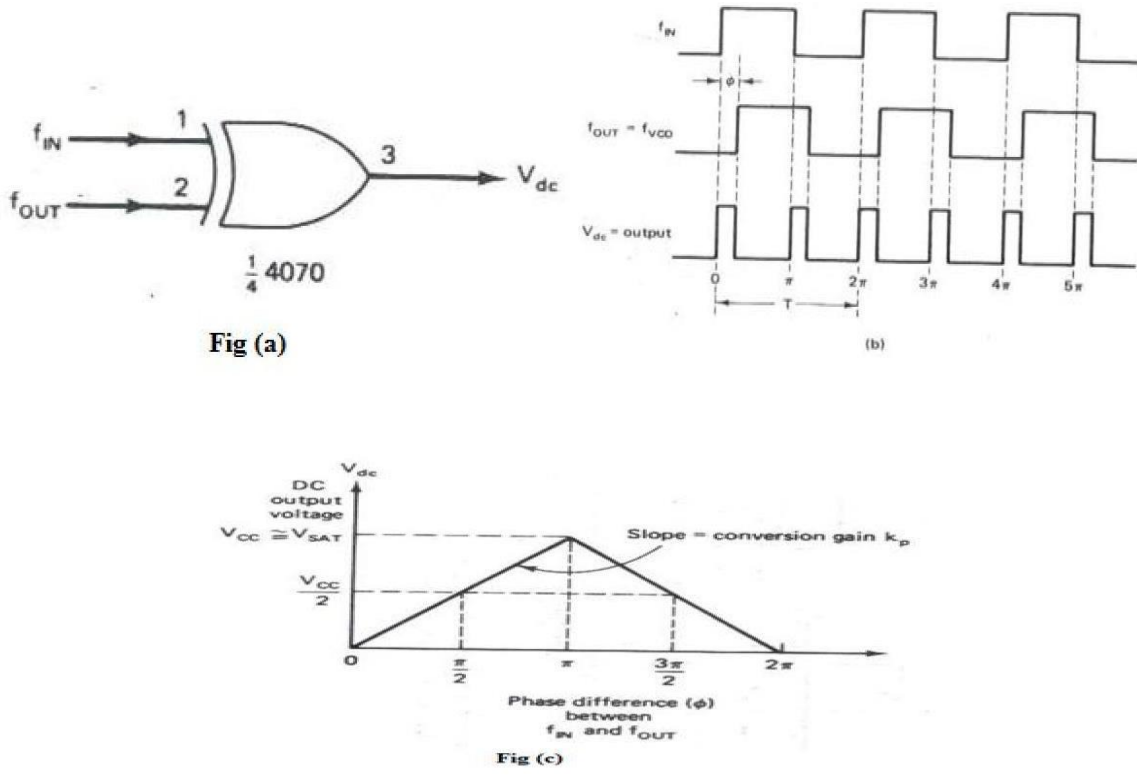


Fig 3.2 (a) Exclusive-OR phase detector: connection and logic diagram. (b) Input and output waveforms. (c) Average output voltage versus phase difference between *f*IN and *f*OUT curve.

(b) Low-pass filter.

The function of the low-pass filter is to remove the high-frequency components in the output of the phase detector and to remove high-frequency noise.

More important, the 1ow-pass filter controls the dynamic characteristics of the phase-locked loop. These characteristics include capture and lock ranges, bandwidth, and transient response. The lock range is defined as the range of frequencies over which the PLL system follows the changes in the input frequency *f*IN. An equivalent term for lock range is tracking range. On the other hand, the capture range is the frequency range in which the PLL acquires phase lock. Obviously, the capture range is always smaller than the lock range.

(c) Voltage-controlled oscillator:

A third section of the PLL is the voltage-controlled oscillator. The VCO generates an output frequency that is directly proportional to its input voltage. Typical example of VCO is Signetics NE/SE 566 VCO, which provides simultaneous square wave and triangular wave outputs as a function of input voltage. The block diagram of the VCO is shown in Fig 2.12. The frequency of oscillations is determined by three external R1 and capacitor C1 and the voltage VC applied to the control terminal 5.

The triangular wave is generated by alternatively charging the external capacitor C1 by one current source and then linearly discharging it by another. The charging and discharging levels are determined by Schmitt trigger action. The schmitt trigger also provides square wave output. Both the wave forms are buffered so that the output impedance of each is 50 ohms.

In this arrangement the R1C1 combination determines the free running frequency and the control voltage VC at pin 5 is set by voltage divider formed with R2 and R3. The initial voltage VC at pin 5 must be in the range

$$\frac{3}{4}(+V) \leq V_C \leq +V$$

Where +V is the total supply voltage.The modulating signal is ac coupled with the capacitor C and must be <3 VPP. The frequency of the output wave forms is approximated by

$$f_O \cong \frac{2(+V - V_C)}{R_1 C_1 (+V)}$$

where R1should be in the range 2KΩ < R1< 20KΩ. For affixed VC and constant C1, the frequency fO can be varied over a 10:1 frequency range by the choice of R1 between 2KΩ < R1< 20KΩ.

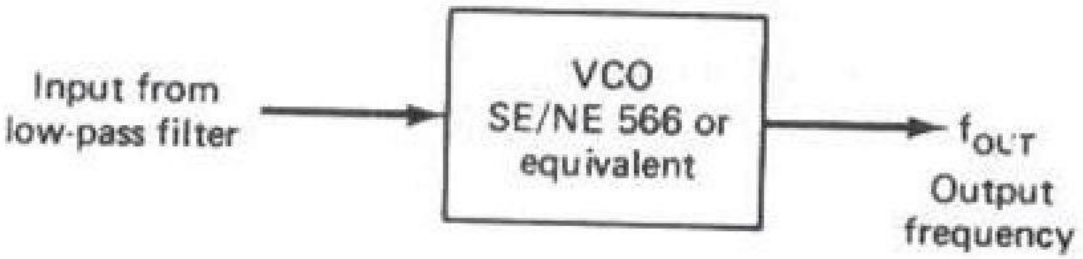


Fig 3.3: VCO Block Diagram

3.2 MONOLITHIC PHASE LOCK LOOPS IC 565:

Monolithic PLLs are introduced by signetics as SE/NE 560 series and by national semiconductors LM 560 series.

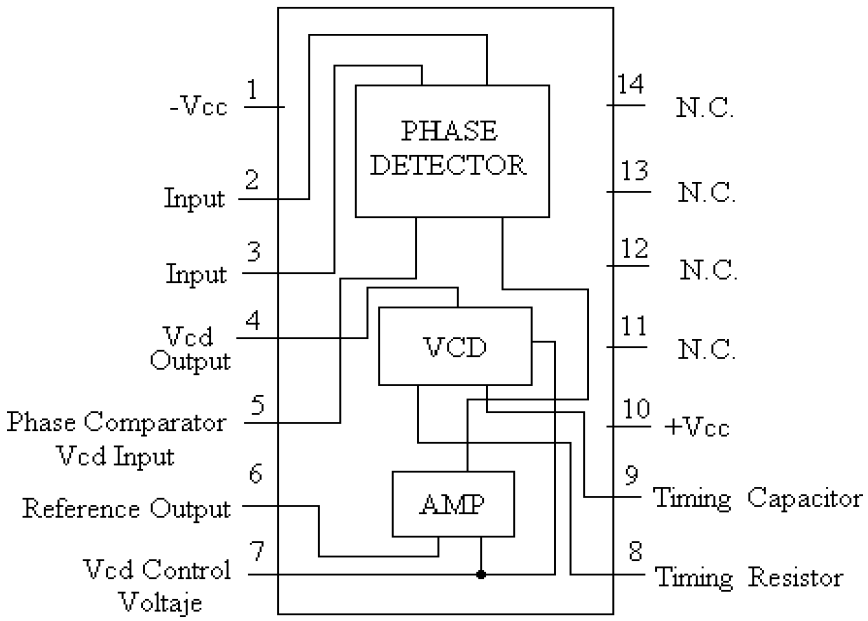


Fig 3.16: Pin configuration of IC 565

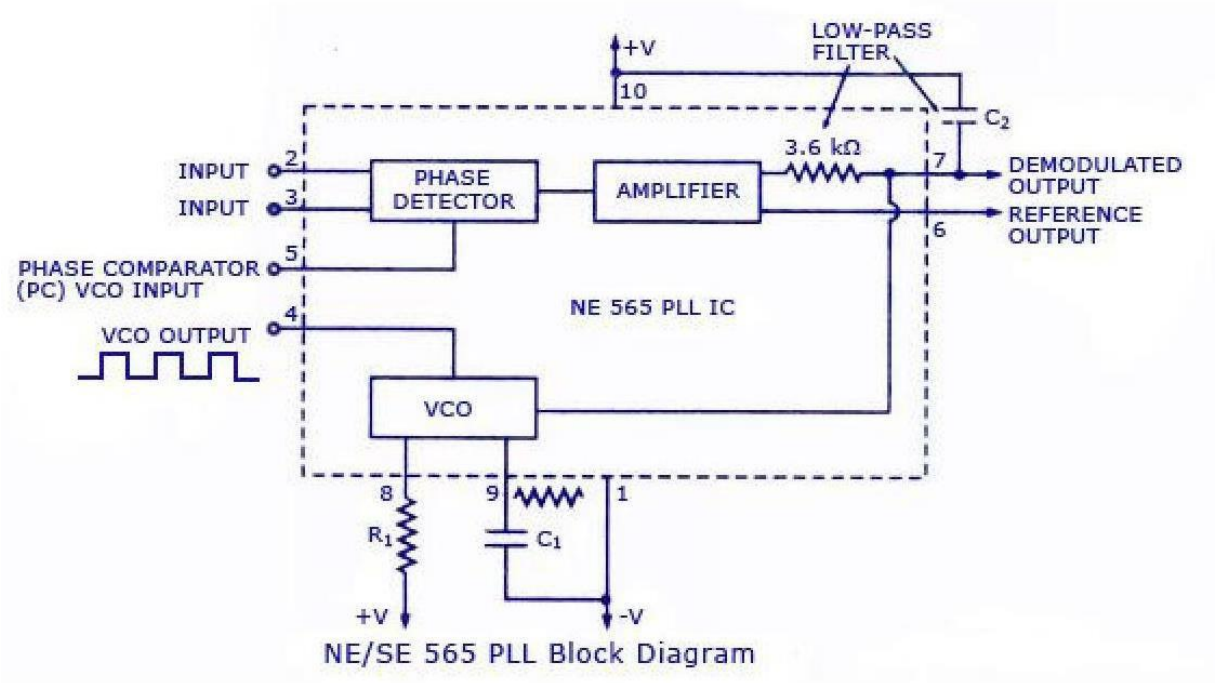


Fig 3.17: Block Diagram of IC 565

Fig 3.16 and 3.17 shows the pin diagram and block diagram of IC 565 PLL. It consists of phase detector, amplifier, low pass filter and VCO.As shown in the block diagram the phase locked feedback loop is not internally connected.
Therefore, it is necessary to connect output of VCO to the phase comparator input, externally. In frequency multiplication applications a digital frequency divider is inserted into the loop i.e., between pin 4 and pin 5. The centre frequency of the PLL is determined by the free-running frequency of the VCO and it is given by

$$f_o = \frac{1.2}{4R_1C_1}$$

Where R1 and C1 are an external resistor and capacitor connected to pins 8 and 9, respectively. The values of R1 and C1 are adjusted such that the free running frequency will be at the centre of the input frequency range. The values of R1 are restricted from 2 kΩ to 20 kΩ,but a capacitor can have any value. A capacitor C2 connected between pin 7 and the positive supply forms a first order low pass filter with an internal resistance of 3.6 kΩ. The value of filter capacitor C2 should be larger enough to eliminate possible demodulated output voltage at pin 7 in order to stabilize the VCO frequency.

The PLL can lock to and track an input signal over typically ±60% bandwidth w.r.t fo as the center frequency. The lock range fL and the capture range fC of the PLL are given by the following equations.

$$f_L = \pm\frac{8f_0}{V}$$

Where fo=free running frequency

V=(+V)-(-V)Volts

And

$$f_C = \pm\sqrt{\frac{f_L}{2\pi(3.6)10^3C_2}}$$

From above equation the lock range increases with an increase in input voltage but decrease with increase in supply voltage. The two inputs to the phase detector allows direct coupling of an input signal, provided that there is no dc voltage difference between the pins and the dc resistances seen from pins 2 and 3 are equal.

3.3 DATA CONVERTER INTEGRATED CIRCUITS

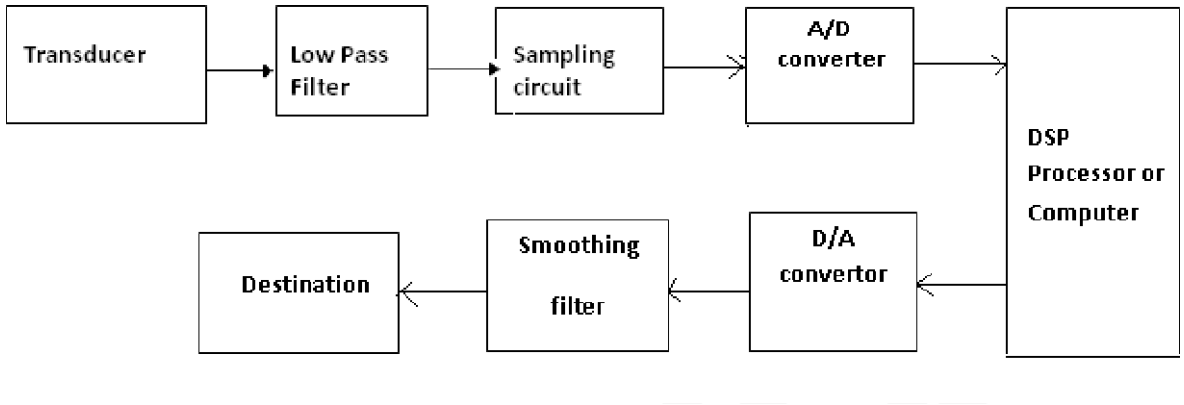


Fig 3.18: Application of A/D and D/A converters

Fig 3.15 shows the application of A/D and D/A converters. The transducer circuit will gives an analog signal. This signal is transmitted through the LPF circuit to avoid higher components, and then the signal is sampled at twice the frequency of the signal to avoid the overlapping. The output of the sampling circuit is applied to A/D converter where the samples are converted into binary data i.e. 0's and 1's. Like this the analog data converted into digital data.

The digital data is again reconverted back into analog by doing exact opposite operation of first half of the diagram. Then the output of the D/A convertor is transmitted through the smoothing filter to avoid the ripples.

3.4 BASIC DAC TECHNIQUES

The input of the block diagram is binary data i.e, 0 and 1,it contain 'n' number of input bits designated as d1,d2,d3,…..dn .this input is combined with the reference voltage called Vdd to give an analog output.

Where d1 is the MSB bit and dn is the LSB bit

$$Vo=Vdd(d1*2-1+d2*2-2+d3*2-3+..... +dn*2-n)$$

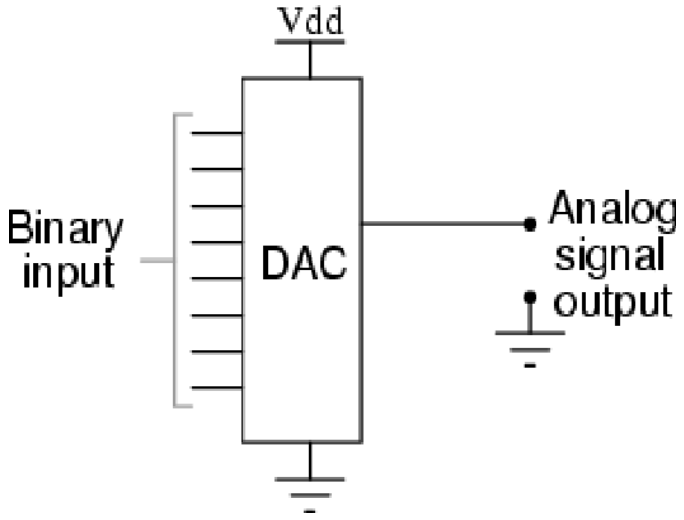


Fig 3.19: Basic DAC diagram

3.4.1 Weighted Resistor:

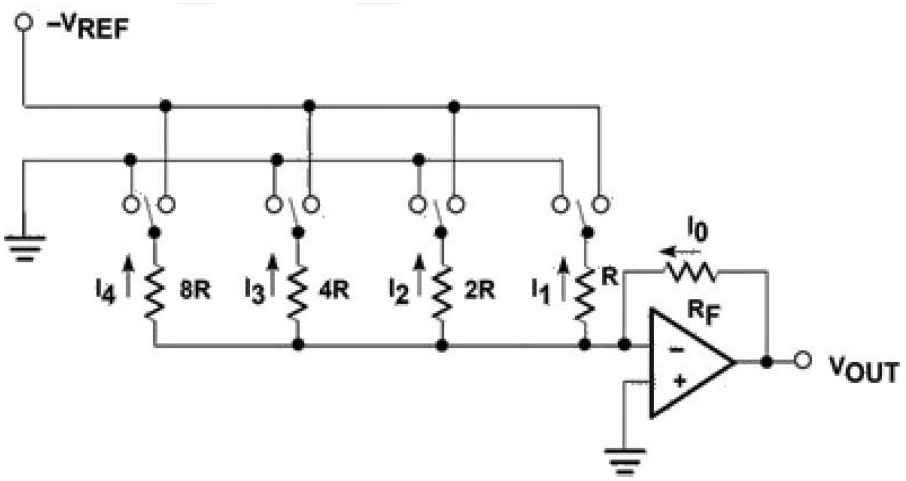


Fig: 2.20: simple 4-bit weighted resistor

Fig. 2.17 shows a simplest circuit of weighted resistor. It uses a summing inverting amplifier. It contains n- electronic switches (i.e. 4 switches) and these switches are controlled by binary input bits d1, d2, d3, d4. If the binary input bit is 1 then the switch is connected to reference voltage –VREF , if the binary input bit is 0 then the switch is connected to ground.

The output current equation is Io=I1+I2+ I3+I 4

Io= VREF (d1*2-1+d2*2-2+d3*2-3+d4*2-4)

The transfer characteristics are shown below (fig 2.13) for a 3-bit weighted resistor

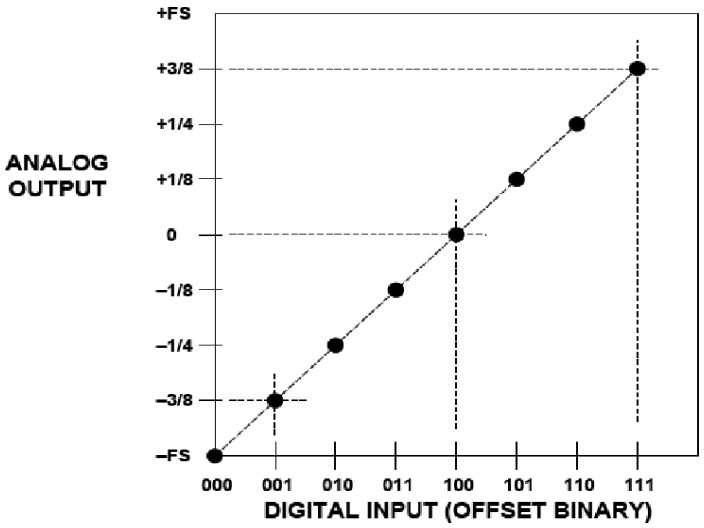


Fig 3.21: Transfer characteristics of 3-bit weighted resistor

Disadvantages of Weighted resistor D/A converter:

Wide range of resistor's are required in this circuit and it is very difficult to fabricate such a wide range of resistance values in monolithic IC. This difficulty can be eliminated using R-2R ladder network.

3.4.2 R-2R LADDER DAC

Wide range of resistors required in binary weighted resistor type DAC. This can be avoided by using R-2R ladder type DAC. The circuit of R-2R ladder network is shown in fig 3.19. The basic theory of the R-2R ladder network is that current flowing through any input resistor (2R) encounters two possible paths at the far end. The effective resistances of both paths are the same (also 2R), so the incoming current splits equally along both paths. The half-current that flows back towards lower orders of magnitude does not reach the op amp, and therefore has no effect on the output voltage. The half that takes the path towards the op amp along the ladder can affect the output. The inverting input of the op-amp is at virtual earth. Current flowing in the elements of the ladder network is therefore unaffected by switch positions.

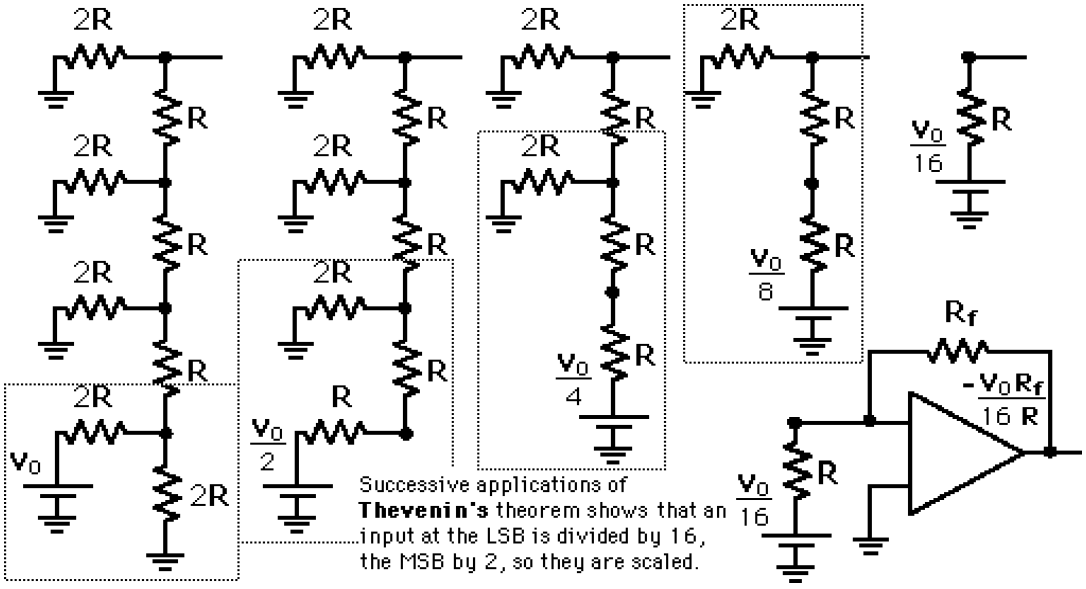


Fig 3.22: A 4-bit R-2R Ladder DAC

If we label the bits (or inputs) bit 1 to bit N the output voltage caused by connecting a particular bit to Vr with all other bits grounded is:

$$Vout = Vr/2N$$

where N is the bit number. For bit 1, Vout =Vr/2, for bit 2, Vout = Vr/4 etc.

Since an R/2R ladder is a linear circuit, we can apply the principle of superposition to calculate Vout. The expected output voltage is calculated by summing the effect of all bits connected to Vr. For example, if bits 1 and 3 are connected to Vr with all other inputs grounded, the output voltage is calculated by:

Vout = (Vr/2)+(Vr/8) which reduces to Vout = 5Vr/8.

An R/2R ladder of 4 bits would have a full-scale output voltage of 1/2 +1/4 + 1/8 + 1/16 = 15Vr/16 or 0.9375 volts (if Vr=1 volt) while a 10bit R/2R ladder would have a full-scale output voltage of 0.99902 (if Vr=1 volt).

3.4.3 INVERTED R-2R LADDER DAC

In weighted resistor and R-2R ladder DAC the current flowing through the resistor is always changed because of the changing input binary bits 0 and 1. More power dissipation causes heating, which in turn cerates non-linearity in DAC. This problem can be avoided by using INVERTED R-2R LADDER DAC (fig 2.20)

In this MSB and LSB is interchanged. Here each input binary word connects the corresponding switch either to ground or to the inverting input terminal of op-amp which is

also at virtual ground. When the input binary in logic 1 then it is connected to the virtual ground, when input binary is logic 0 then it is connected to the ground i.e. the current flowing through the resistor is constant.

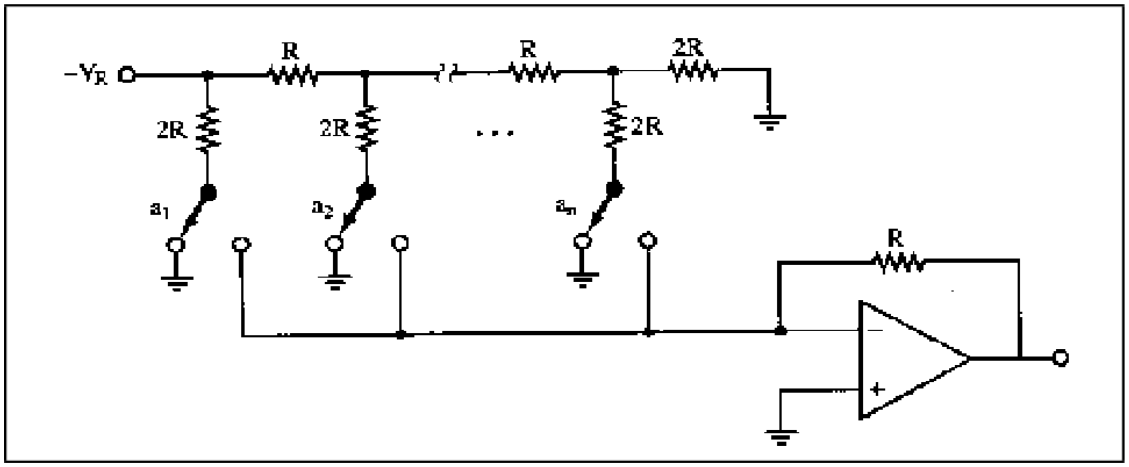


Fig 3.23: Inverted R-2R ladder

3.5 DIFFERENT TYPES OF ADC'S

It provides the function just opposite to that of a DAC. It accepts an analog input voltage Va and produces an output binary word d1, d2, d3…. dn. Where d1 is the most significant bit and dn is the least significant bit.

ADCs are broadly classified into two groups according to their conversion techniques

1) Direct type

2) Integrating type

Direct type ADCs compares a given analog signal with the internally generated equivalent signal. This group includes

i) Flash (Comparator) type converter

ii) Successive approximation type convertor

iii) Counter type

iv) Servo or Tracking type

Integrated type ADCs perform conversion in an indirect manner by first changing the analog input signal to linear function of time or frequency and then to a digital code.

3.5.1 FLASH (COMPARATOR) TYPE CONVERTER:

A direct-conversion ADC or flash ADC has a bank of comparators sampling the input signal in parallel, each firing for their decoded voltage range. The comparator bank feeds a logic circuit that generates a code for each voltage range. Direct conversion is very fast,

capable of gigahertz sampling rates, but usually has only 8 bits of resolution or fewer, since the number of comparators needed, 2N - 1, doubles with each additional bit, requiring a large, expensive circuit. ADCs of this type have a large die size, a high input capacitance, high power dissipation, and are prone to produce glitches at the output (by outputting an out-of-sequence code). Scaling to newer sub-micrometre technologies does not help as the device mismatch is the dominant design limitation. They are often used for video, wideband communications or other fast signals in optical storage.

A Flash ADC (also known as a direct conversion ADC) is a type of analog-to-digital converter that uses a linear voltage ladder with a comparator at each "rung" of the ladder to compare the input voltage to successive reference voltages. Often these reference ladders are constructed of many resistors; however modern implementations show that capacitive voltage division is also possible. The output of these comparators is generally fed into a digital encoder which converts the inputs into a binary value (the collected outputs from the comparators can be thought of as a unary value).

Also called the *parallel* A/D converter, this circuit is the simplest to understand. It is formed of a series of comparators, each one comparing the input signal to a unique reference voltage. The comparator outputs connect to the inputs of a priority encoder circuit, which then produces a binary output.

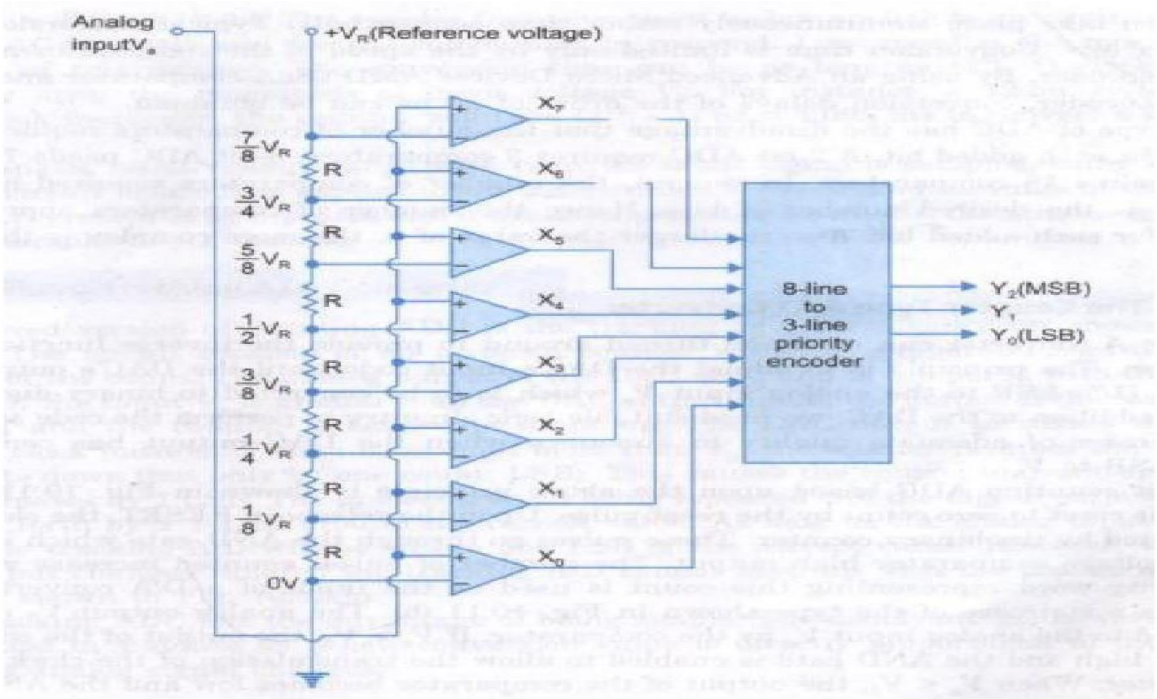


Fig 3.24: flash (parallel comparator) type ADC

VR is a stable reference voltage provided by a precision voltage regulator as part of the converter circuit, not shown in the schematic. As the analog input voltage exceeds the reference voltage at each comparator, the comparator outputs will sequentially saturate to a high state. The priority encoder generates a binary number based on the highest-order active input, ignoring all other active inputs.

3.5.2 COUNTER TYPE A/D CONVERTER

In the fig 3.22 the counter is reset to zero count by reset pulse. After releasing the reset pulse the clock pulses are counted by the binary counter. These pulses go through the AND gate which is enabled by the voltage comparator high output. The number of pulses counted increase with time.

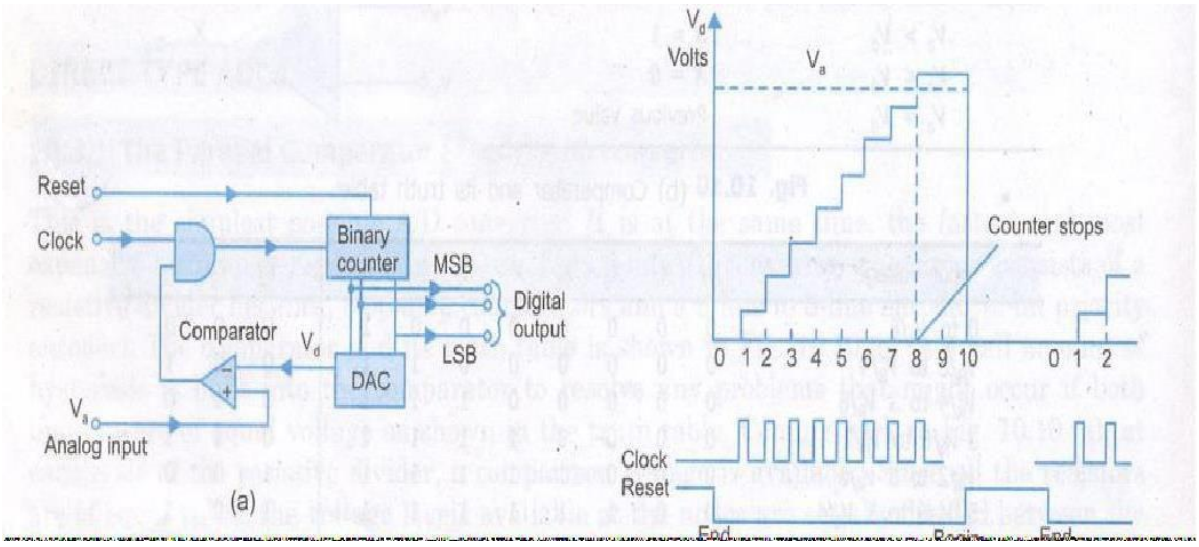


Fig 3.25: Countertype A/D converter

The binary word representing this count is used as the input of a D/A converter whose output is a stair case. The analog output Vd of DAC is compared to the analog input input Va by the comparator. If Va>Vd the output of the comparator becomes high and the AND gate is enabled to allow the transmission of the clock pulses to the counter. When Va<Vd the output of the comparator becomes low and the AND gate is disabled. This stops the counting we can get the digital data.

3.5.3 SERVO TRACKING A/D CONVERTER :

An improved version of counting ADC is the tracking or servo converter shown in fig 3.23. The circuit consists of an up/down counter with the comparator controlling the direction of the count

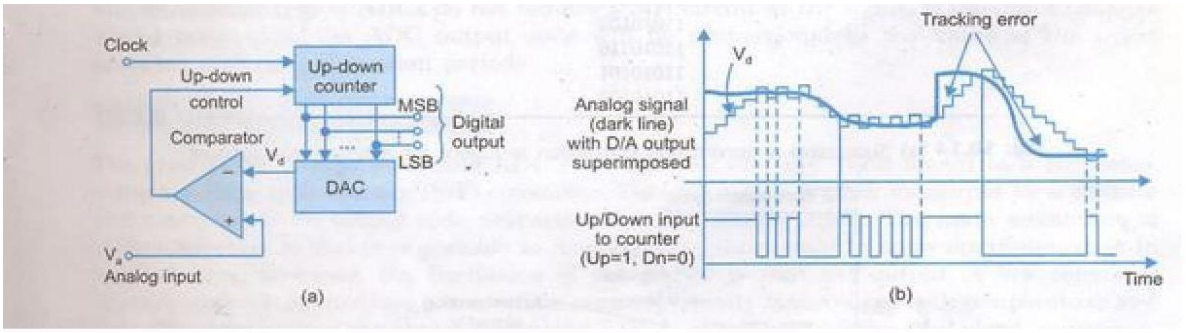


Fig: 3.26: (a) A tracking A/D converter (b) waveforms associated with a tracking A/D converter

The analog output of the DAC is Vd and is compared with the analog input Va. If the input Va is greater than the DAC output signal, the output of the comparator goes high and the counter is caused to count up. The DAC output increases with each incoming clock pulse when it becomes more than Va the counter reverses the direction and counts down.

3.5.4 SUCCESSIVE-APPROXIMATION ADC:

One method of addressing the digital ramp ADC's shortcomings is the so-called successive- approximation ADC. The only change in this design as shown in the fig 2.19 is a very special counter circuit known as a successive-approximation register.

Instead of counting up in binary sequence, this register counts by trying all values of bits starting with the most-significant bit and finishing at the least-significant bit. Throughout the count process, the register monitors the comparator's output to see if the binary count is less than or greater than the analog signal input, adjusting the bit values accordingly. The way the register counts is identical to the "trial-and-fit" method of decimal-to-binary conversion, whereby different values of bits are tried from MSB to LSB to get a binary number that equals the original decimal number. The advantage to this counting strategy is much faster results: the DAC output converges on the analog signal input in much larger steps than with the 0-to-full count sequence of a regular counter.

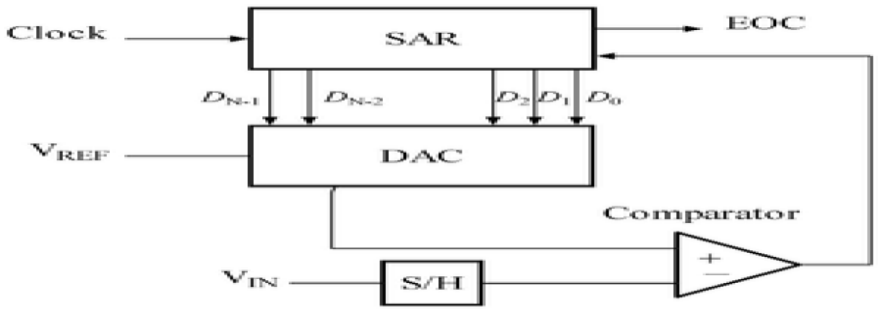


Fig: 3.27: Successive approximation ADC circuits

The successive approximation analog to digital converter circuit typically consists of four chief subs

1. A sample and hold circuit to acquire the input voltage (Vin).

2. An analog voltage comparator that compares Vin to the output of the internal DAC and outputs the result of the comparison to the successive approximation register (SAR).

3. A successive approximation register sub circuit designed to supply an approximate digital code of Vin to the internal DAC.

4. An internal reference DAC that supplies the comparator with an analog voltage equivalent of the digital code output of the SAR for comparison with Vin.

The successive approximation register is initialized so that the most significant bit (MSB) is equal to a digital 1. This code is fed into the DAC, which then supplies the analog equivalent of this digital code (Vref/2) into the comparator circuit for comparison with the sampled input voltage. If this analog voltage exceeds Vin the comparator causes the SAR to reset this bit; otherwise, the bit is left a 1. Then the next bit is set to 1 and the same test is done, continuing this binary search until every bit in the SAR has been tested. The resulting code is the digital approximation of the sampled input voltage and is finally output by the DAC at the end of the conversion (EOC).

Mathematically, let Vin = xVref, so x in [-1, 1] is the normalized input voltage. The objective is to approximately digitize x to an accuracy of 1/2n. The algorithm proceeds as follows:

1. Initial approximation x0 = 0.

2. ith approximation xi = xi-1 - s(xi-1 - x)/2i.

where, s(x) is the signum-function(sgn(x)) (+1 for $x \geq 0$, -1 for $x < 0$). It follows using mathematical induction that $|xn - x| \leq 1/2n$.

As shown in the above algorithm, a SAR ADC requires:

1. An input voltage source Vin.

2. A reference voltage source Vref to normalize the input.

3. A DAC to convert the ith approximation xi to a voltage.

4. A Comparator to perform the function s(xi - x) by comparing the DAC's voltage with the input voltage.

5. A Register to store the output of the comparator and apply xi-1 - s(xi-1 - x)/2i.

A successive-approximation ADC uses a comparator to reject ranges of voltages, eventually settling on a final voltage range. Successive approximation works by constantly comparing the input voltage to the output of an internal digital to analog converter (DAC, fed by the current value of the approximation) until the best approximation is achieved. At each step in this process, a binary value of the approximation is stored in a successive approximation register (SAR). The SAR uses a reference voltage (which is the largest signal the ADC is to convert) for comparisons.

For example, if the input voltage is 60 V and the reference voltage is 100 V, in the 1st clock cycle, 60 V is compared to 50 V (the reference, divided by two. This is the voltage at the output of the internal DAC when the input is a '1' followed by zeros), and the voltage from the comparator is positive (or '1') (because 60 V is greater than 50 V). At this point the first binary digit (MSB) is set to a '1'. In the 2nd clock cycle the input voltage is compared to 75 V (being halfway between 100 and 50 V: This is the output of the internal DAC when its input is '11' followed by zeros) because 60 V is less than 75 V, the comparator output is now negative (or '0'). The second binary digit is therefore set to a '0'. In the 3rd clock cycle, the input voltage is compared with 62.5 V (halfway between 50 V and 75 V: This is the output of the internal DAC when its input is '101' followed by zeros). The output of the comparator is negative or '0' (because 60 V is less than 62.5 V) so the third binary digit is set to a 0. The fourth clock cycle similarly results in the fourth digit being a '1' (60 V is greater than 56.25 V, the DAC output for '1001' followed by zeros). The result of this would be in the binary form 1001. This is also called *bit-weighting conversion*, and is similar to a binary search.

The analogue value is rounded to the nearest binary value below, meaning this converter type is mid-rise (see above). Because the approximations are successive (not simultaneous), the conversion takes one clock-cycle for each bit of resolution desired. The clock frequency must be equal to the sampling frequency multiplied by the number of bits of resolution desired. For example, to sample audio at 44.1 kHz with 32 bit resolution, a clock frequency of over 1.4 MHz would be required. ADCs of this type have good resolutions and quite wide ranges. They are more complex than some other designs.

3.5.5 DUAL-SLOPE ADC

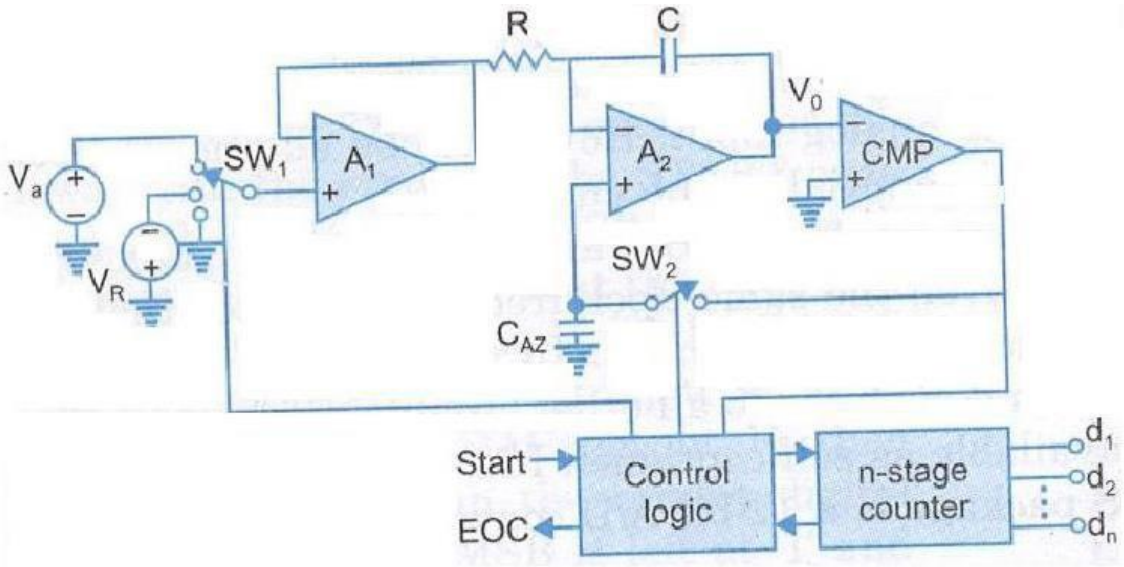


Fig 3.28: (a): Functional diagram of dual slope ADC

An integrating ADC (also **dual-slope** ADC) shown in fig 3.25 (a) applies the unknown input voltage to the input of an integrator and allows the voltage to ramp for a fixed time period (the run-up period). Then a known reference voltage of opposite polarity is applied to the integrator and is allowed to ramp until the integrator output returns to zero (the run-down period). The input voltage is computed as a function of the reference voltage, the constant run-up time period, and the measured run-down time period. The run-down time measurement is usually made in units of the converter's clock, so longer integration times allow for higher resolutions. Likewise, the speed of the converter can be improved by sacrificing resolution. Converters of this type (or variations on the concept) are used in most digital voltmeters for their linearity and flexibility.

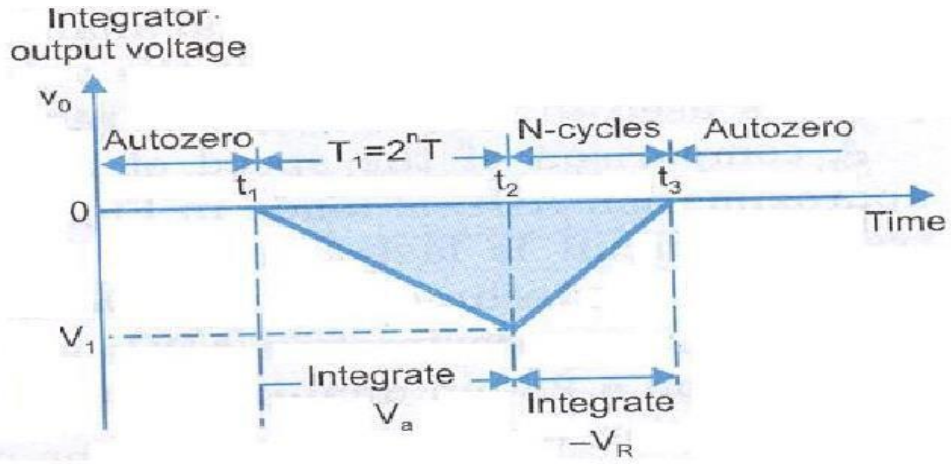


Fig 3.25 (b) o/p waveform of dual slope ADC

In operation the integrator is first zeroed (close SW2), then attached to the input (SW1 up) for a fixed time *M* counts of the clock (frequency 1/*t*). At the end of that time it is attached to the reference voltage (SW1 down) and the number of counts *N* which accumulate before the integrator reaches zero volts output and the comparator output changes are determined. The waveform of dual slope ADC is shown in fig 2.25 (b).

The equations of operation are therefore:

$$T_1 = t_2 - t_1 = \frac{2^n\ counts}{clock\ rate}$$

And

$$t_3 - t_2 = \frac{digitalcountN}{clockrate}$$

For an integrator,

$$\Delta V_O = \frac{-1}{RC} V(\Delta t)$$

The voltage Vo will be equal to V1 at the instant t2 and can be written as

$$V_1 = \frac{-1}{RC} V_a (t_2 - t_1)$$

The voltage V1 is also given by

$$V_1 = \frac{-1}{RC}(-V_R)(t_2 - t_3)$$

So,

$$V_a(t_2 - t_1) = (V_R)(t_3 - t_2)$$

Putting the values of $(t_2 - t_1) = 2^n$ and $(t_3 - t_2) = N$, we get

$$V_a(2^n) = (V_R)N$$

Or,

$$V_a = (V_R)\left(\frac{N}{2^n}\right)$$

3.6 SPECIFICATIONS FOR DAC/ADC

1. RESOLUTION: The Resolution of a converter is the smallest change in voltage which may be produced at the output of the converter.

Resolution (in volts) =(VFS)/(2n-1) = 1 LSB increment

Ex: An 8-bit D/A converter have 28-1=255 equal intervals. Hence the smallest change in output voltage is (1/255) of the full-scale output range.

An 8-bit DAC is said to have: 8 bit resolution

: a resolution of 0.392 of full scale

: a resolution of 1 part in 255

Similarly the resolution of an A/D converter is defined as the smallest change in analog input for a one bit change at the output.

Ex: the input range of 8-bit A/D converter is divided into 255 intervals. So the resolution for a 10V input range is 39.22 mV = (10V/255)

2. LINEARITY: The linearity of an A/D or D/A converter is an important measure of its accuracy and tells us how close the converter output is to its ideal characteristics.

3. GLITCHES (PARTICULARLY DAC): In transition from one digital input to the next, like 0111 to 1000, it may effectively go through 1111 or 0000, which produces ―unexpected‖ voltage briefly. If can cause problems elsewhere.

4. ACCURACY: Absolute accuracy is the maximum deviation between the actual converter output and the ideal converter output.

5. MONOTONIC: A monotonic DAC is the one whose analog output increases for an increase in digital input. It is essential in control applications. If a DAC has to be monotonic, the error should de lessthan ±(1/2) LSB at each output level.

6. SETTLING TIME: The most important dynamic parameter is the settling time. It represents the time it takes for the output to settle within a specified band ± (1/2) LSB of its final value following a code change at the input. It depends upon the switching time of the logic circuitry due to internal parasitic capacitances and inductances. Its ranges from 100ns to 10μs.

7. STABILITY: The performance of converter changes with temperature, age and power supply variations. So the stability is required.

(CMOS LOGIC)

Logic Signals and Gates

Digital logic hides the pitfalls of the analog world by mapping the infinite set of real values for a physical quantity into two subsets corresponding to just two possible numbers or *logic values*—0 and 1. As a result, digital logic circuits can be analyzed and designed functionally, using switching algebra, tables, and other abstract means to describe the operation of well-behaved 0s and 1s in a circuit.

A logic value, 0 or 1, is often called a *binary digit*, or *bit*. If an application requires more than two discrete values, additional bits may be used, with a set of *n* bits representing 2*n* different values. With most phenomena, there is an undefined region between the 0 and 1 states (e.g., voltage = 1.8 V, dim light, capacitor slightly charged, etc.). This undefined region is needed so that the 0 and 1 states can be unambiguously defined and reliably detected. Noise can more easily corrupt results if the boundaries separating the 0 and 1 states are too close. When discussing electronic logic circuits such as CMOS and TTL, digital designers often use the words –LOW‖ and –HIGH‖ in place of –0‖ and –1‖ to remind them that they are dealing with real circuits, not abstract quantities:

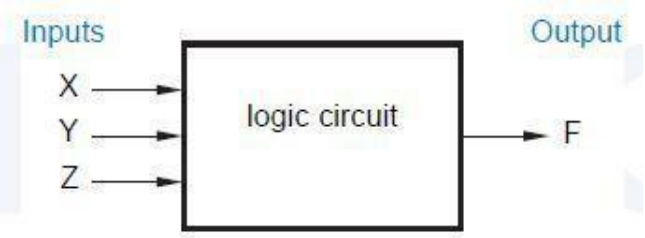
LOW A signal in the range of algebraically lower voltages, which is interpreted as a logic 0.

HIGH A signal in the range of algebraically higher voltages, which is interpreted as a logic 1.

Note that the assignments of 0 and 1 to LOW and HIGH are somewhat arbitrary. Assigning 0 to LOW and 1 to HIGH seems most natural, and is called *positive logic*. The opposite assignment, 1 to LOW and 0 to HIGH, is not often used, and
is called *negative logic*.

Because a wide range of physical values represent the same binary value, digital logic is highly immune to component and power supply variations and noise. Furthermore, *buffer amplifier* circuits can be used to regenerate –weak‖ values into –strong‖ ones, so that digital signals can be transmitted over arbitrary distances without loss of information. For example, a buffer amplifier for CMOS logic converts any HIGH input voltage into an output very close to 5.0 V, and any LOW input voltage into an output very close to 0.0 V. A logic circuit can be represented with a minimum amount of detail simply as a –black box‖ with a certain number of inputs and outputs. For example, Figure shows a logic circuit with three inputs and one output. However, this representation does not describe how the circuit responds to input signals.

From the point of view of electronic circuit design, it takes a lot of information to describe the precise electrical behavior of a circuit. However, since the inputs of a digital logic circuit can be viewed as taking on only discrete 0 and 1 values, the circuit's –logical‖ operation can be described with a table that ignores electrical behavior and lists only discrete 0 and 1 values.



A logic circuit whose outputs depend only on its current inputs is called a *combinational circuit*. Its operation is fully described by a *truth table* that lists all combinations of input values and the output value(s) produced by each one.

Table is the truth table for a logic circuit with three inputs X, Y, and Z and a single output F.

X	Y	Z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

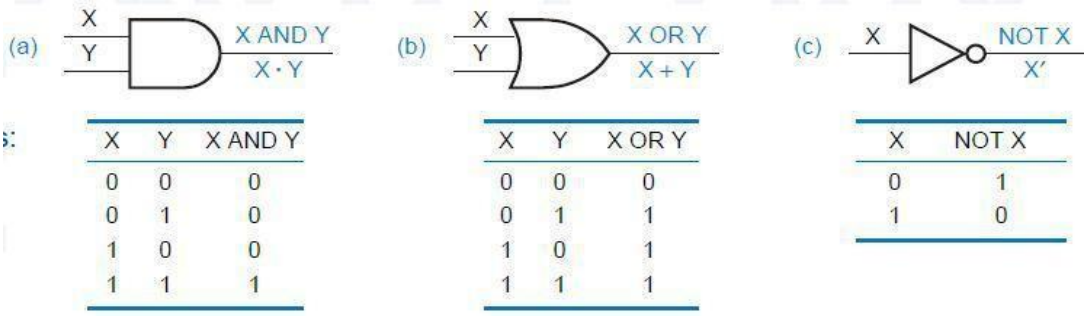
A circuit with memory, whose outputs depend on the current input *and* the sequence of past inputs, is called a *sequential circuit*. The behavior of such a circuit may be described by a *state table* that specifies its output and next state as functions of its current state and input.

The gates' functions are easily defined in words:

• An AND *gate* produces a 1 output if and only if all of its inputs are 1.

• An OR *gate* produces a 1 if and only if one or more of its inputs are 1.

• A NOT *gate,* usually called an *inverter*, produces an output value that is the opposite of its input value.



X	Y	X AND Y
0	0	0
0	1	0
1	0	0
1	1	1

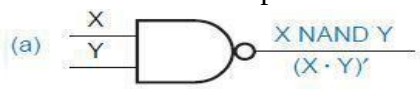
X	Y	X OR Y
0	0	0
0	1	1
1	0	1
1	1	1

X	NOT X
0	1
1	0

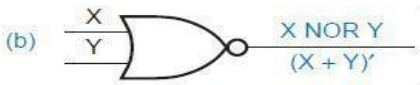
The circle on the inverter symbol's output is called an *inversion bubble*, and is used in this and other gate symbols to denote ‑inverting‖ behavior. Notice that in the definitions of AND and OR functions, we only had to state the input conditions for which the output is 1, because there is only one possibility when the output is not 1—it must be 0.

Two more logic functions areobtained by combining NOT with an AND or OR function in a single gate. Figure 3-3 shows the truth tables and symbols for these gates; Their functions are also easily described in words:

• A NAND *gate* produces the opposite of an AND gate's output, a 0 if and only if all of its inputs are 1.
• A NOR *gate* produces the opposite of an OR gate's output, a 0 if and only if one or more of its inputs are 1.



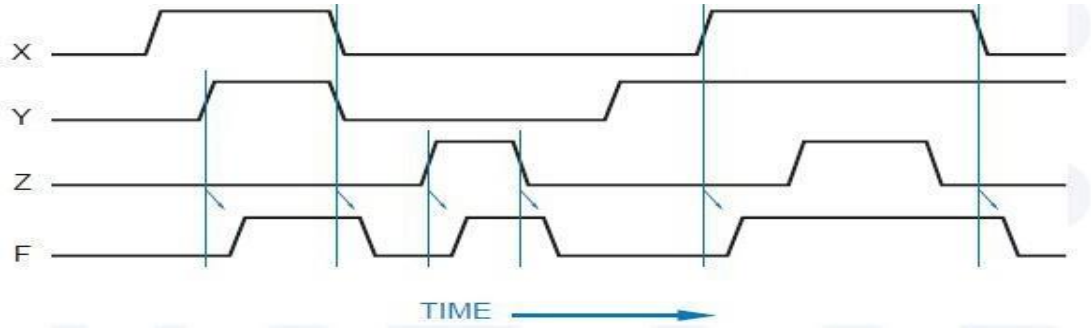
X	Y	X NAND Y
0	0	1
0	1	1
1	0	1
1	1	0



X	Y	X NOR Y
0	0	1
0	1	0
1	0	0
1	1	0

As with AND and OR gates, the symbols and truth tables for NAND and NOR may be extended to gates with any number of inputs.

Real logic circuits also function in another analog dimension—time. For example, Figure is a *timing diagram* that shows how the circuit might respond to a time-varying pattern of input signals. The timing diagram shows that the logic signals do not change between 0 and 1 instantaneously, and also that there is a lag between an input change and the corresponding output change.



Logic Families

There are many, many ways to design an electronic logic circuit.

1. The first electrically controlled logic circuits, developed at Bell Laboratories in 1930s, were based on relays.
2. In the mid-1940s, the first electronic digital computer, the Eniac, used logic circuits based on vacuum tubes. The Eniac had about 18,000 tubes and a similar number of logic gates, not a lot by today's standards of microprocessor chips with tens of millions of transistors. However, the Eniac could hurt you a lot more than a chip could if it fell on you—it was 100 feet long, 10 feet high, 3 feet deep, and consumed 140,000 watts of power!

3. The inventions of the *semiconductor diode* and the *bipolar junction transistor* allowed the development of smaller, faster, and more capable computers in the late 1950s.
4. In the 1960s, the invention of the *integrated circuit (IC)* allowed multiple diodes, transistors, and other components to be fabricated on a single chip, and computers got still better.

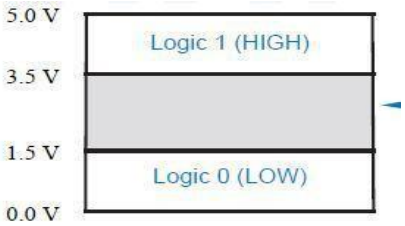
A *logic family:* is a collection of different integrated-circuit chips that have similar input, output, and internal circuit characteristics, but that perform different logic functions. Chips from the same family can be interconnected to perform any desired logic function.

CMOS Logic

The functional behavior of a CMOS logic circuit is fairly easy to understand, even if your knowledge of analog electronics is not particularly deep. The basic (and typically only) building blocks in CMOS logic circuits are MOS transistors, described shortly. Before introducing MOS transistors and CMOS logic circuits, we must talk about logic levels.

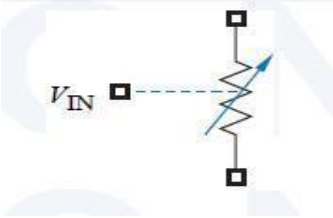
CMOS Logic Levels

Abstract logic elements process binary digits, 0 and 1. However, real logic circuits process electrical signals such as voltage levels. In any logic circuit, there is a range of voltages (or other circuit conditions) that is interpreted as a logic 0, and another, nonoverlapping range that is interpreted as a logic 1. A typical CMOS logic circuit operates from a 5-volt power supply. Such a circuit may interpret any voltage in the range 0–1.5 V as a logic 0, and in the range 3.5–5.0 V as a logic 1. Thus, the definitions of LOW and HIGH for 5-volt CMOS logic are as shown in Figure. Voltages in the intermediate range are not expected to occur except during signal transitions, and yield undefined logic values (i.e., a circuit may interpret them as either 0 or 1). CMOS circuits using other power supply voltages, such as 3.3 or 2.7 volts, partition the voltage range similarly.



MOS Transistors

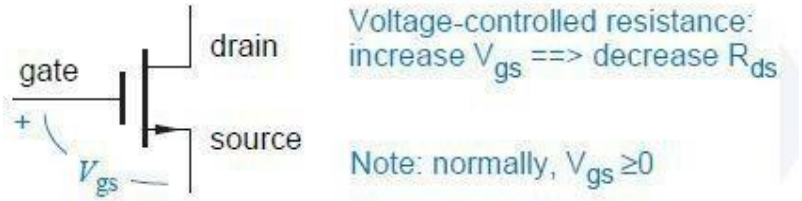
A MOS transistor can be modeled as a 3-terminal device that acts like a voltagecontrolled resistance. As suggested by Figure, an input voltage applied to one terminal controls the resistance between the remaining two terminals. In digital logic applications, a MOS transistor is operated so its resistance is always either very high (and the transistor is ―off‖) or very low (and the transistor is ―on‖).



There are two types of MOS transistors, *n*-channel and *p*-channel; the names refer to the type of semiconductor material used for the resistance-controlled terminals.

n-channel MOS (NMOS) transistor:

The circuit symbol for an *n-channel MOS (NMOS) transistor* is shown in Figure . The terminals are called *gate*, *source*, and *drain*. (Note that the –gate‖ of a MOS transistor has nothing to do with a –logic gate.‖) As you might guess from the orientation of the circuit symbol, the drain is normally at a higher voltage than the source. The voltage from gate to source (*V*gs) in an NMOS transistor is normally zero or positive. If *V*gs = 0, then the resistance from drain to source (*R*ds) is very high, on the order of a megohm (106 ohms) or more. As we increase *V*gs (i.e., increase the voltage on the gate), *R*ds decreases to a very low value, 10 ohms or less in some devices.



p-channel MOS (PMOS) transistor:

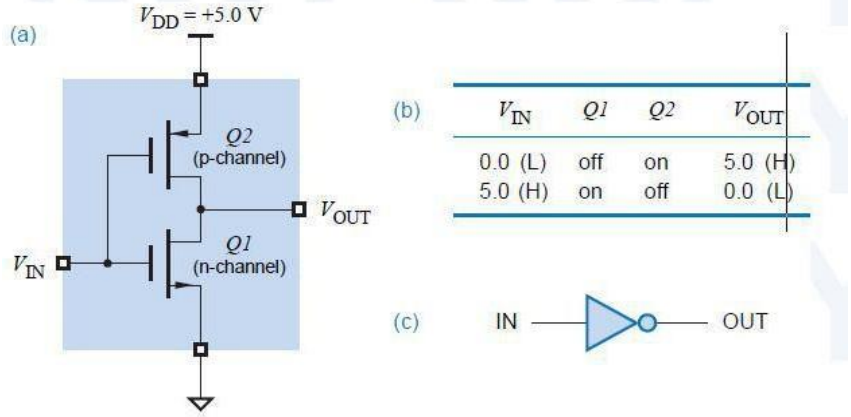
The circuit symbol for a *p-channel MOS (PMOS) transistor* is shown in Figure. Operation is analogous to that of an NMOS transistor, except that the source is normally at a higher voltage than the drain, and *V*gs is normally zero or negative. If *V*gs is zero, then the resistance from source to drain (*R*ds) is very high. As we algebraically decrease *V*gs (i.e., *decrease* the voltage on the gate), *R*ds decreases to a very low value.

The gate of a MOS transistor has a very high impedance. That is, the gate is separated from the source and the drain by an insulating material with a very high resistance. However, the gate voltage creates an electric field that enhances or retards the flow of current between source and drain. This is the –field effect‖ in the –MOSFET‖ name.

Basic CMOS Inverter Circuit

NMOS and PMOS transistors are used together in a complementary way to form *CMOS logic*. The simplest CMOS circuit, a logic inverter, requires only one of each type of transistor, connected as shown in Figure. The power supply voltage, *V*DD, typically may be in the range 2–6 V, and is most often set at 5.0 V for compatibility with TTL circuits. Ideally, the functional behavior of the CMOS inverter circuit can be characterized by just two cases tabulated in Figure:

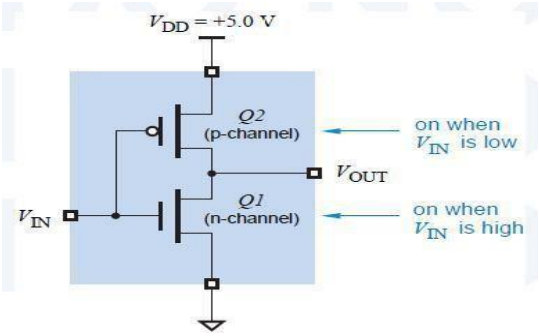
1. *V*IN is 0.0 V. In this case, the bottom, *n*-channel transistor *Q1* is off, since its *V*gs is 0, but the top, *p*-channel transistor *Q2* is on, since its *V*gs is a large negative value (5.0 V). Therefore, *Q2* presents only a small resistance between the power supply terminal (*V*DD, 5.0 V) and the output terminal (*V*OUT), and the output voltage is 5.0 V.
2. *V*IN is 5.0 V. Here, *Q1* is on, since its *V*gs is a large positive value (+5.0 V), but *Q2* is off, since its *V*gs is 0. Thus, *Q1* presents a small resistance between the output terminal and ground, and the output voltage is 0 V.



With the foregoing functional behavior, the circuit clearly behaves as a logical inverter, since a 0-volt input produces a 5-volt output, and vice versa Another way to visualize CMOS operation uses switches. As shown in Figure, the *n*-channel (bottom) transistor is modeled by a normally-open switch, and the *p*-channel (top) transistor by a normally-closed switch. Applying a HIGH voltage changes each switch to the opposite of its normal state, as shownin (b).

The switch model gives rise to a way of drawing CMOS circuits that makes their logical behavior more readily apparent. As shown in Figure , different symbols are used for the *p*-and *n*- channel transistors to reflect their logical behavior. The *n*-channel transistor (*Q1*) is switched

—on,‖ and current flows between source and drain, when a HIGH voltage is applied to its gate; this seems natural enough. The *p*-channel transistor (*Q2*) has the opposite behavior. It is –on‖ when a LOW voltage is applied; the inversion bubble on its gate indicates this inverting behavior.



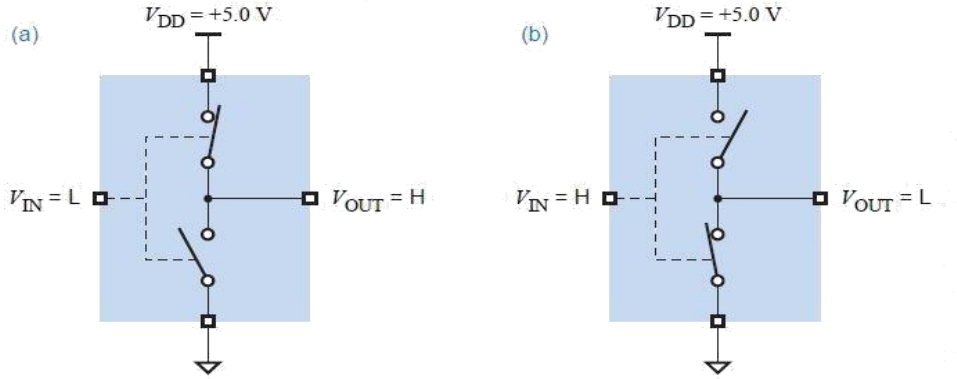
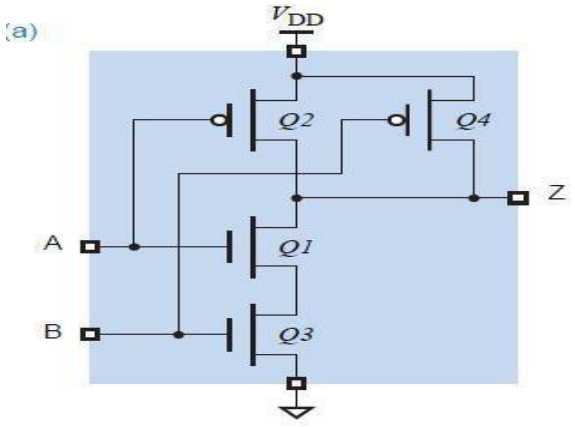


Figure 3-11
Switch model for CMOS inverter: (a LOW input; (b) HIG input.

CMOS NAND and NOR Gates

Both NAND and NOR gates can be constructed using CMOS. A *k*-input gate uses *k* *p*-channel and *k* *n*-channel transistors. Figure shows a 2-input CMOS NAND gate. If either input is LOW, the output Z has a low-impedance connection to *V*DD through the corresponding —on‖ *p*-channel transistor, and the path to ground is blocked by the corresponding –off‖ *n*-channel transistor. If both inputs are HIGH, the path to *V*DD is blocked, and Z has a low-impedance connection to ground. Figure shows the switch model

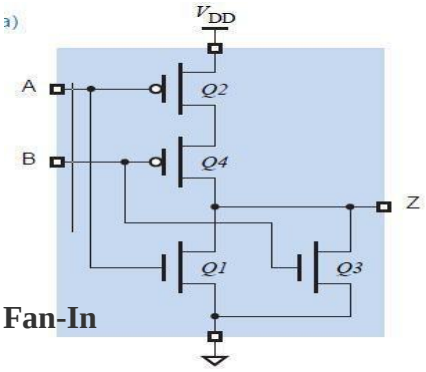


(b)

A	B	Q1	Q2	Q3	Q4	Z
L	L	off	on	off	on	H
L	H	off	on	on	off	H
H	L	on	off	off	on	H
H	H	on	off	on	off	L

for the NAND gate's operation.

Figure shows a CMOS NOR gate. If both inputs are LOW, the output Z has a low-impedance connection to *V*DD through the ―on‖ *p*-channel transistors, and the path to ground is blocked by the ―off‖ *n*-channel transistors. If either input is HIGH, the path to *V*DD is blocked, and Z has a low-impedance connection to ground.



(b)

A	B	Q1	Q2	Q3	Q4	Z
L	L	off	on	off	on	H
L	H	off	on	on	off	L
H	L	on	off	off	on	L
H	H	on	off	on	off	L

(c)

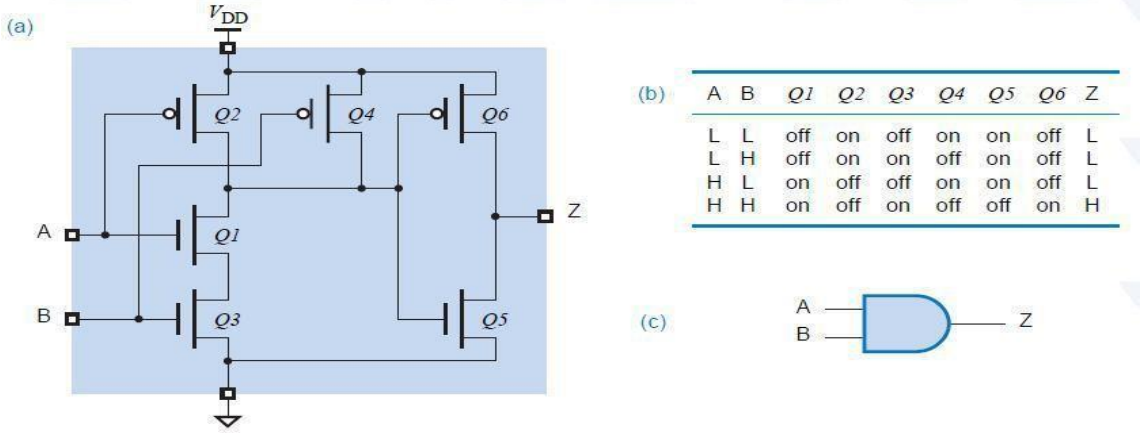
Figure 3-15
CMOS 2-input NOR gate:
(a) circuit diagram;
(b) function table;
(c) logic symbol.

The number of inputs that a gate can have in a particular logic family is called the logic family's *fan-in*. As the number of inputs is increased, CMOS gate designers may compensate by increasing the size of the series transistors to reduce their resistance and the corresponding switching delay. However, at some point this becomes inefficient or impractical. Gates with a large number of inputs can be made faster and smaller by cascading gates with fewer inputs.

Noninverting Gates

In CMOS, and in most other logic families, the simplest gates are inverters, and the next simplest are NAND gates and NOR gates. A logical inversion comes –for free,‖ and it typically is not possible to design a noninverting gate with a smaller number of transistors than an inverting one. CMOS noninverting buffers and AND and OR gates are obtained by connecting an inverter to the output of the corresponding inverting gate.

Figure 3-19 CMOS 2-input AND gate: (a) circuit diagram; (b) function table; (c) logic symbol.

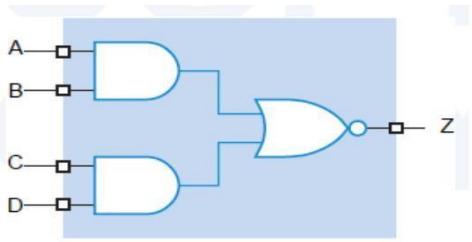


A	B	Q1	Q2	Q3	Q4	Q5	Q6	Z
L	L	off	on	off	on	on	off	L
L	H	off	on	on	off	on	off	L
H	L	on	off	off	on	on	off	L
H	H	on	off	on	off	off	on	H

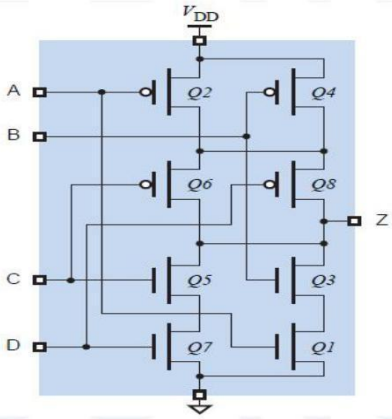
CMOS AND-OR-INVERT and OR-AND-INVERT Gates

CMOS circuits can perform two levels of logic with just a single –level‖ of transistors. For example, the circuit in Figure is a two-wide, two-input CMOS AND-OR-INVERT (AOI) *gate*. The function table for this circuit is shown in (b) and a logic diagram for this function using AND and NOR gates is shown in Figure 3-21. Transistors can be added to or removed from this circuit to obtain an AOI function with a different number of ANDs or a different number of inputs per AND.

The contents of each of the *Q1–Q8* columns in Figure 0(b) depends only on the input signal connected to the corresponding transistor's gate. The last column is constructed by examining each input combination and determining whether Z is connected to *V*DD or ground by ―on‖ transistors for that input combination. Note that Z is never connected to *both V*DD and ground for any input combination; in such a case the output would be a non-logic value some where between LOW and HIGH, and the output structure would consume excessive power due to the low-impedance connection between *V*DD and ground.



(a)

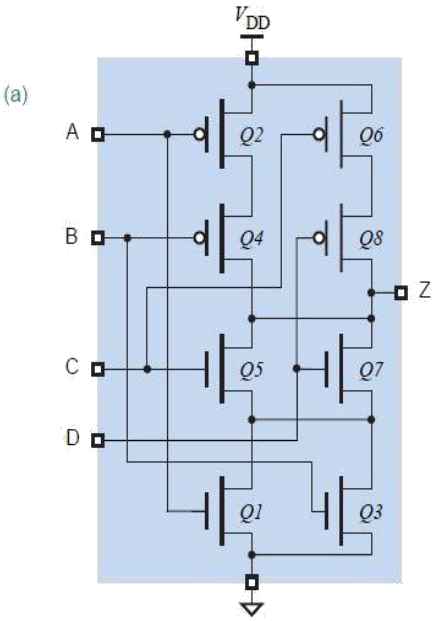


(b)

A	B	C	D	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Z
L	L	L	L	off	on	off	on	off	on	off	on	H
L	L	L	H	off	on	off	on	off	on	on	off	H
L	L	H	L	off	on	off	on	on	off	off	on	H
L	L	H	H	off	on	off	on	on	off	on	off	L
L	H	L	L	off	on	on	off	off	on	off	on	H
L	H	L	H	off	on	on	off	off	on	on	off	H
L	H	H	L	off	on	on	off	on	off	off	on	H
L	H	H	H	off	on	on	off	on	off	on	off	L
H	L	L	L	on	off	off	on	off	on	off	on	H
H	L	L	H	on	off	off	on	off	on	on	off	H
H	L	H	L	on	off	off	on	on	off	off	on	H
H	L	H	H	on	off	off	on	on	off	on	off	L
H	H	L	L	on	off	on	off	off	on	off	on	L
H	H	L	H	on	off	on	off	off	on	on	off	L
H	H	H	L	on	off	on	off	on	off	off	on	L
H	H	H	H	on	off	on	off	on	off	on	off	L

A circuit can also be designed to perform an OR-AND-INVERT function. For example, Figure is a two-wide, two-input CMOS OR-AND-INVERT (OAI) *gate*. The function table for this circuit is shown in (b); the values in each column are determined just as we did for the CMOS AOI gate.

(a)



(b)

A	B	C	D	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Z
L	L	L	L	off	on	off	on	off	on	off	on	H
L	L	L	H	off	on	off	on	off	on	on	off	H
L	L	H	L	off	on	off	on	on	off	off	on	H
L	L	H	H	off	on	off	on	on	off	on	off	H
L	H	L	L	off	on	on	off	off	on	off	on	H
L	H	L	H	off	on	on	off	off	on	on	off	L
L	H	H	L	off	on	on	off	on	off	off	on	L
L	H	H	H	off	on	on	off	on	off	on	off	L
H	L	L	L	on	off	off	on	off	on	off	on	H
H	L	L	H	on	off	off	on	off	on	on	off	L
H	L	H	L	on	off	off	on	on	off	off	on	L
H	L	H	H	on	off	off	on	on	off	on	off	L
H	H	L	L	on	off	on	off	off	on	off	on	H
H	H	L	H	on	off	on	off	off	on	on	off	L
H	H	H	L	on	off	on	off	on	off	off	on	L
H	H	H	H	on	off	on	off	on	off	on	off	L

(BIPOLAR LOGIC AND INTERFACING)

Diode Logic

Diode action can be exploited to perform logical operations. Within the 5-volt range, signal voltages are partitioned into two ranges, LOW and HIGH, with a 1-volt noise margin between. A voltage in the LOW range is considered to be a logic 0, and a voltage in the HIGH range is a logic 1. With these definitions, a *diode* AND *gate* can be constructed as shown in Figure 3-64(a). In this circuit, suppose that both inputs X and Y are connected to HIGH voltage sources, say 4 V, so that *VX* and *VY* are both 4 V as in (b). Then both diodes are forward biased, and the output voltage *VZ* is one diode-drop above 4 V, or about 4.6 V. A small amount of current, determined by the value of *R*, flows from the 5-V supply through the two diodes and into the 4-V sources. The colored arrows in the figure show the path of this current flow.

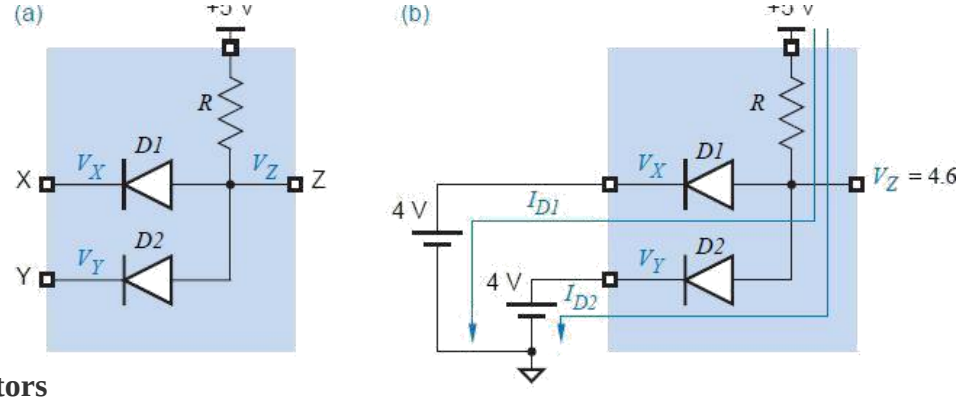


Figure 3-64
Diode AND gate: (a) electrical circuit; (b) both inputs HIGH; (c) one input HIGH, one LOW; (d) function table; (e) truth table.

Bipolar Junction Transistors

A *bipolar junction transistor* is a three-terminal device that, in most logic circuits, acts like a current-controlled switch. If we put a small current into one of the terminals, called the *base*, then the switch is –on‖—current may flow between the other two terminals, called the *emitter* and the *collector*. If no current is put into the base, then the switch is –off‖—no current flows between the emitter and the collector.

Transistor Logic Inverter

Figure 3-69 shows that we can make a logic inverter from an *npn* transistor in the common-emitter configuration. When the input voltage is LOW, the output voltage is HIGH, and vice versa. In digital switching applications, bipolar transistors are often operated so they are always either cut off or saturated. That is, digital circuits such as the

$$VCE = VCC - IcR2$$
$$VCC = - IbR2$$
$$= VCC - VIN - 0.6\ R2 / R1$$

inverter in Figure are designed so that their transistors are always (well, almost always) in one of the states depicted in Figure 3-70. When the input voltage *VIN* is LOW, it is low enough that *Ib* is zero and the transistor is cut off; the collector-emitter junction looks like an open

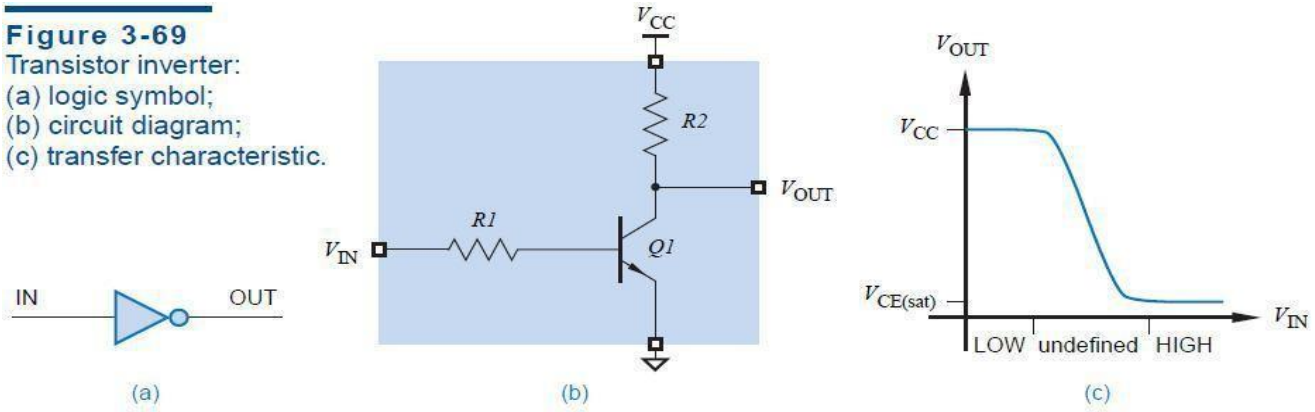


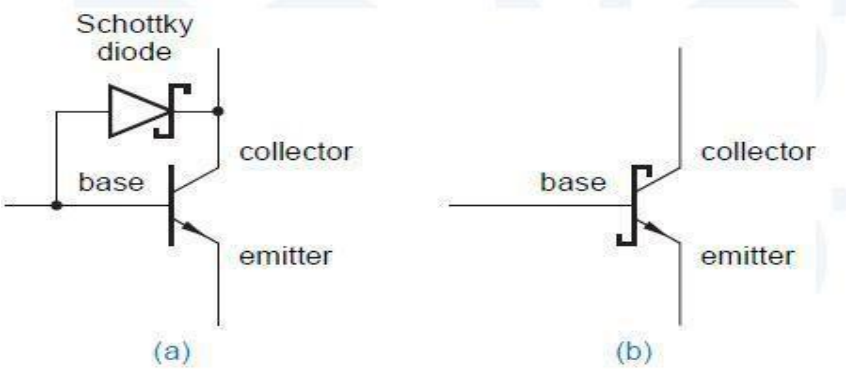
Figure 3-69
Transistor inverter:
(a) logic symbol;
(b) circuit diagram;
(c) transfer characteristic.

circuit. When *V*IN is HIGH, it is high enough (and *R1* is low enough and is high enough) that the transistor will be saturated for any reasonable value of *R2*; the collector-emitter junction looks almost like a short circuit. Input voltages in the undefined region between LOW and HIGH are not allowed, except during transitions.

Schottky Transistors

When the input of a saturated transistor is changed, the output does not change immediately; it takes extra time, called *storage time*, to come out of saturation. In fact, storage time accounts for a significant portion of the propagation delay in the original TTL logic family. Storage time can be eliminated and propagation delay can be reduced by ensuring that transistors do not saturate in normal operation. Contemporary TTL logic families do this by placing a *Schottky diode* between the base and collector of each transistor that might saturate, as shown in Figure The resulting transistors, which do not saturate, are called *Schottky-clamped transistors* or *Schottky transistors* for short.

When forward biased, a Schottky diode's voltage drop is much less than a standard diode's, 0.25 V vs. 0.6 V. In a standard saturated transistor, the base-to-collector voltage is 0.4 V, as shown in Figure .



Transistor-Transistor Logic

The most commonly used bipolar logic family is transistor-transistor logic. Actually, there are many different TTL families, with a range of speed, power consumption, and other characteristics. The circuit examples in this section are based on a representative TTL family,

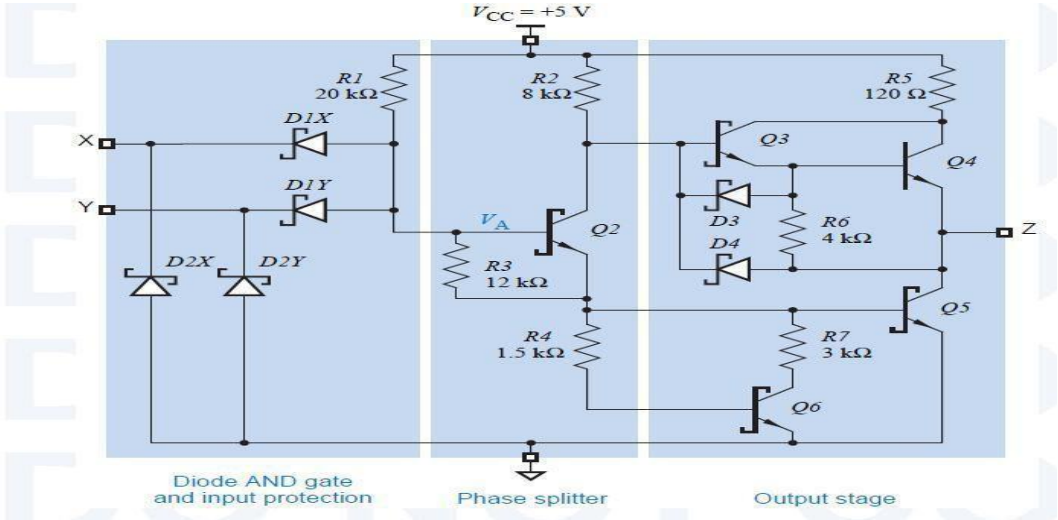
Low-power Schottky (LS or LS-TTL). TTL families use basically the same logic levels as the TTL-compatible CMOS families in previous sections. We'll use the following definitions of LOW and HIGH in our discussions of TTL circuit behavior: LOW 0–0.8 volts. HIGH 2.0–5.0 volts.

Basic TTL NAND **Gate**

The circuit diagram for a two-input LS-TTL NAND gate, part number 74LS00, is shown in Figure 3-75. The NAND function is obtained by combining a diode AND gate with an inverting buffer amplifier. The circuit's operation is best understood by dividing it into the three parts that are shown in the figure and discussed in the next three paragraphs:

- Diode AND gate and input protection.
- Phase splitter.
- Output stage.

Diodes *D1X* and *D1Y* and resistor *R1* in Figure form a *diode* AND *gate*. *Clamp diodes D2X* and *D2Y* do nothing in normal operation, but limit undesirable negative excursions on the inputs to a single diode drop. Such negative excursions may occur on HIGH-to-LOW input transitions as a result of transmission-line effects. Transistor *Q2* and the surrounding resistors form a *phase splitter* that controls the output stage. Depending on whether the diode AND gate produces a –low‖ or a –high‖ voltage at *VA*, *Q2* is either cut off or turned on.



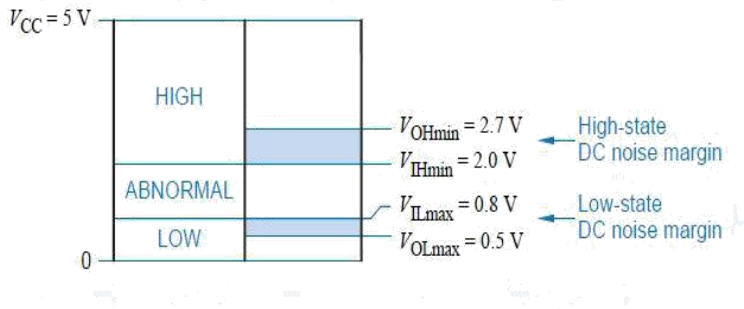
The *output stage* has two transistors, *Q4* and *Q5*, only one of which is on at any time. The TTL output stage is sometimes called a *totem-pole* or *push-pull output*. Similar to the *p*-channel and *n*- channel transistors in CMOS, *Q4* and *Q5* provide active pull-up and pull-down to the HIGH and LOW states, respectively.

Logic Levels and Noise Margins

At the beginning of this section, we indicated that we would consider TTL signals between 0 and 0.8 V to be LOW, and signals between 2.0 and 5.0 V to be HIGH. Actually, we can be more precise by defining TTL input and output levels in the same way as we did for CMOS:

*V*OHmin The minimum output voltage in the HIGH state, 2.7 V for most TTL families.

*V*IHmin The minimum input voltage guaranteed to be recognized as a HIGH, 2.0 V for all TTL families.



CURRENT SPIKES

Current spikes can show up as noise on the power-supply and ground connections in TTL and CMOS circuits, especially when multiple outputs are switched simultaneously. For this reason, reliable circuits require decoupling capacitors between *V*CC and ground, distributed throughout the circuit so that there is a capacitor within an inch or so of each chip. Decoupling capacitors supply the instantaneous current needed during transitions.

sinking current *sourcing current*

Fanout

As we defined it previously in Section 3.5.4, *fanout* is a measure of the number of gate inputs that are connected to (and driven by) a single gate output. As we showed in that section, the DC fanout of CMOS outputs driving CMOS inputs is virtually unlimited, because CMOS inputs require almost no current in either state, HIGH or LOW. This is not the case with TTL inputs. As a result, there are very definite limits on the fanout of TTL or CMOS outputs driving TTL inputs, as you'll learn in the paragraphs that follow.

As in CMOS, the *current flow* in a TTL input or output lead is defined to be positive if the current actually flows *into* the lead, and negative if current flows *out* of the lead. As a result,

when an output is connected to one or more inputs, the algebraic sum of all the input and output currents is 0. The amount of current required by a TTL input depends on whether the input is HIGH or LOW.

Unused Inputs

Unused inputs of TTL gates can be handled in the same way as we described for CMOS gates in Section 3.5.6 on page 107. That is, unused inputs may be tied to used ones, or unused inputs may be pulled HIGH or LOW as is appropriate for the logic function. The resistance value of a pull-up or pull-down resistor is more critical with TTL gates than CMOS gates, because TTL inputs draw significantly more current, especially in the LOW state. If the resistance is too large, the voltage drop across the resistor may result in a gate input voltage beyond the normal LOW or HIGH range.

TTL Families

TTL families have evolved over the years in response to the demands of digital designers for better performance. As a result, three TTL families have come and gone, and today's designers have five surviving families from which to choose. All of the TTL families are compatible in that they use the same power supply voltage and logic levels, but each family has its own advantages in terms of speed, power consumption, and cost.

Early TTL Families

The original TTL family of logic gates was introduced by Sylvania in 1963. It was popularized by Texas Instruments, whose "7400-series" part numbers for gates and other TTL components quickly became an industry standard. As in 7400-series CMOS, devices in a given TTL family have part numbers of the form 74FAM*nn*, where "FAM" is an alphabetic family mnemonic and *nn* is a numeric function designator. Devices in different families with the same value of *nn* perform the same function. In the original TTL family, "FAM" is null and the family is called *74-series TTL*.

Resistor values in the original TTL circuit were changed to obtain two more TTL families with different performance characteristics. The *74H (Highspeed TTL)* family used lower resistor values to reduce propagation delay at the expense of increased power consumption. The *74L (Low-power TTL)* family used higher resistor values to reduce power consumption at the expense of propagation delay.

The availability of three TTL families allowed digital designers in the 1970s to make a choice between high speed and low power consumption for their circuits. However, like many people in the 1970s, they wanted to "have it all, now." The development of Schottky transistors provided this opportunity, and made 74, 74H, and 74L TTL obsolete. The characteristics of better performing, contemporary TTL families are discussed in the rest of this section.

Schottky TTL Families

Historically, the first family to make use of Schottky transistors was *74S (Schottky TTL)*. With Schottky transistors and low resistor values, this family has much higher speed, but higher power consumption, than the original 74-series TTL. Perhaps the most widely used and certainly the least expensive TTL family is *74LS (Low-power Schottky TTL)*, introduced shortly after 74S. By combining Schottky transistors with higher resistor values, 74LS TTL matches the speed of 74-series TTL but has about one-fifth of its power consumption. Thus, 74LS is a preferred logic family for new TTL designs. Subsequent IC processing and circuit innovations gave rise to two more Schottky logic families. The *74AS (Advanced Schottky TTL)* family offers speeds approximately twice as fast as 74S with approximately the same power

74-series TTL
74H (High-speed TTL)
74L (Low-power TTL)
74S (Schottky TTL)

74LS (Low-power Schottky TTL)
74AS (Advanced Schottky TTL)

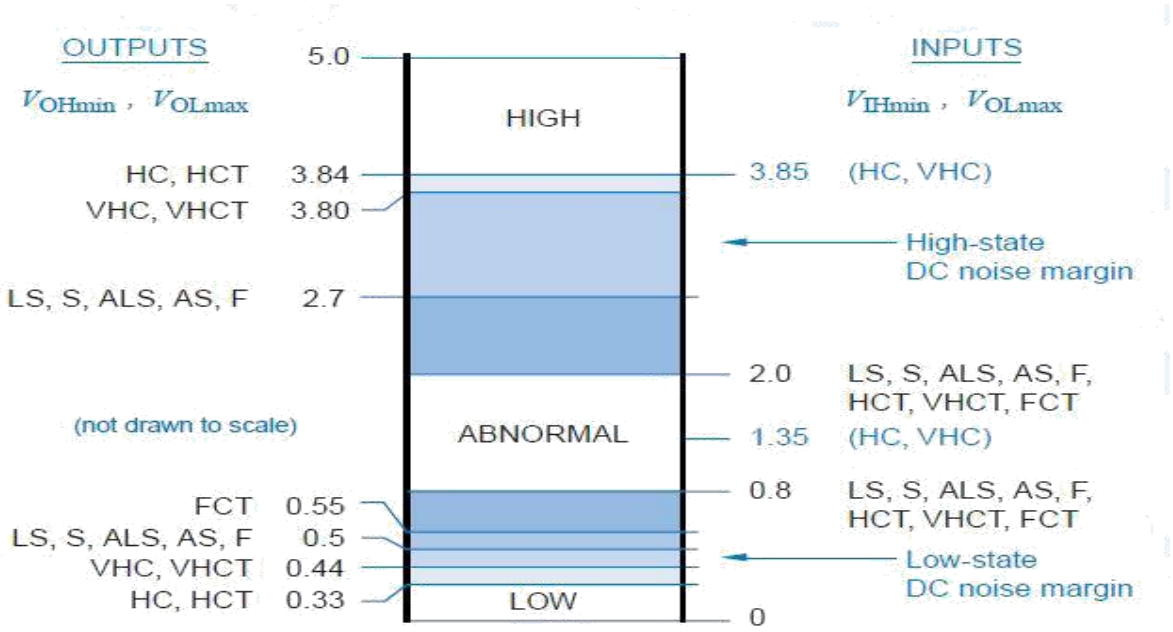
Characteristics of TTL Families

The important characteristics of contemporary TTL families are summarized in Table. The first two rows of the table list the propagation delay (in nanoseconds) and the power consumption (in milliwatts) of a typical 2-input NAND gate in each family. One figure of merit of a logic family is its *speed-power product* listed in the third row of the table. As discussed previously, this is simply the product of the propagation delay and power consumption of a typical gate. The speed-power product measures a sort of efficiency—how much energy a logic gate uses to switch its output.

		Family				
Description	**Symbol**	**74S**	**74LS**	**74AS**	**74ALS**	**74F**
Maximum propagation delay (ns)		3	9	1.7	4	3
Power consumption per gate (mW)		19	2	8	1.2	4
Speed-power product (pJ)		57	18	13.6	4.8	12
LOW-level input voltage (V)	V_{ILmax}	0.8	0.8	0.8	0.8	0.8
LOW-level output voltage (V)	V_{OLmax}	0.5	0.5	0.5	0.5	0.5
HIGH-level input voltage (V)	V_{IHmin}	2.0	2.0	2.0	2.0	2.0
HIGH-level output voltage (V)	V_{OHmin}	2.7	2.7	2.7	2.7	2.7
LOW-level input current (mA)	I_{ILmax}	−2.0	−0.4	−0.5	−0.2	−0.6
LOW-level output current (mA)	I_{OLmax}	20	8	20	8	20
HIGH-level input current (μA)	I_{IHmax}	50	20	20	20	20
HIGH-level output current (μA)	I_{OHmax}	−1000	−400	−2000	−400	−1000

CMOS/TTL Interfacing

A digital designer selects a –default‖ logic family to use in a system, based on general requirements of speed, power, cost, and so on. However, the designer may select devices from other families in some cases because of availability or other special requirements. (For example, not all 74LS part numbers are available in 74HCT, and vice versa.) Thus, it's important for a designer to understand the implications of connecting TTL outputs to CMOS inputs, and vice versa. There are several factors to consider in TTL/CMOS interfacing, and the first is noise margin. The LOW-state DC noise margin depends on *V*OLmax of the driving output and *V*ILmax of the driven input, and equals *V*ILmax *V*OLmax. Similarly, the HIGH-state DC noise margin equals *V*OHmin *V*IHmin.



Low-Voltage CMOS Logic and Interfacing

Two important factors have led the IC industry to move towards lower powersupply voltages in CMOS devices:

• In most applications, CMOS output voltages swing from rail to rail, so the *V* in the *CV*2*f* equation is the power-supply voltage. Cutting power-supply voltage reduces dynamic power dissipation more than proportionally.

• As the industry moves towards ever-smaller transistor geometries, the oxide insulation between a CMOS transistor's gate and its source and drain is getting ever thinner, and thus incapable of insulating voltage potentials as –high‖ as 5 V. As a result, JEDEC, an IC industry standards group, selected 3.3V ± 0.3V, 2.5V ± 0.2V, and 1.8V± 0.15V as the next –standard‖ logic power-supply voltages. JEDEC standards specify the input and output logic voltage levels for devices operating with these power-supply voltages.

Emitter-Coupled Logic

The key to reducing propagation delay in a bipolar logic family is to prevent a gate's transistors from saturating. In Section 3.9.5, we learned how Schottky diodes prevent saturation in TTL gates. However, it is also possible to prevent saturation by using a radically different circuit structure, called *current-mode logic (CML)* or *emitter-coupled logic (ECL)*. Unlike the other logic families in this chapter, CML does not produce a large voltage swing between the LOW and HIGH levels. Instead, it has a small voltage swing, less than a volt, and it internally switches current between two possible paths, depending on the output state. The first CML logic family was introduced by General Electric in 1961.

The concept was soon refined by Motorola and others to produce the still popular 10K and 100K *emitter-coupled logic (ECL)* families. These families are

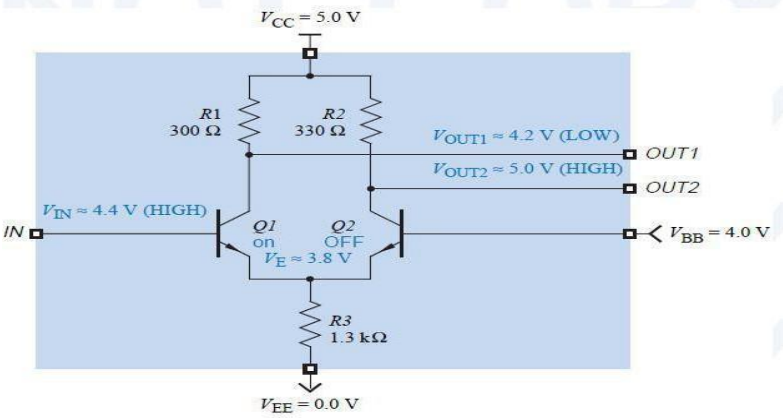
level translator

level shifter

current-mode logic(CML)

emitter-coupled

logic(ECL)



UNIT-IV

THE VHDL HARDWARE DESCRIPTION LANGUAGE

What is VHDL?

"VHDL" stands for "VHSIC Hardware Description Language." VHSIC, in turn, stands for "Very High Speed Integrated Circuit, which was a joint program between the US Department of Defense and IEEE in the mid-1980s to research on high-performance IC technology.

VHDL was standardized by the IEEE in 1987 (VHDL-87) and extended in 1993 (VHDL-93).

Features:

Designs may be decomposed hierarchically.

Each design element has both

1. a well-defined interface (for connecting it to other elements) and
2. a precise behavioral specification (for simulating it).

Behavioral specifications can use either an algorithm or an actual hardware structure to define an element's operation.

Concurrency, timing, and clocking can all be modeled.

VHDL handles asynchronous as well as synchronous sequential-circuit structures.

The logical operation and timing behavior of a design can be simulated.

VHDL synthesis tools are programs that can create logic-circuit structures directly from VHDL behavioral descriptions. Using VHDL, you can design, simulate, and synthesize anything from a simple combinational circuit to a complete microprocessor system on a chip.

Design Flow: Steps in the design flow:

1. Hierarchical / block diagram. Figuring out the basic approach and building blocks at the block diagram level. Large logic designs are usually hierarchical, and VHDL gives you a good framework for defining modules and their interfaces and filling in the details later.

2. Coding. Actual writing of VHDL code for modules, their interfaces, and their internal details.

3. Compilation. Analyses your code for syntax errors and checks it for compatibility with other modules on which it relies. Compilation also creates the internal information that is needed for simulation.

4. Simulation. A VHDL simulator allows you to define and apply inputs to your design, and to observe its outputs. Simulation is part of a larger step called verification. A functional verification is performed to verify that the circuit's logical operation works as desired independent of timing considerations and gate delays.

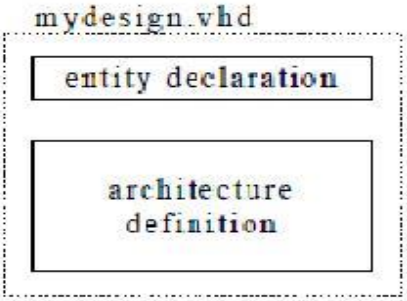
5. Synthesis. Converting the VHDL description into a set of primitives or components that can be assembled in the target technology. For example, with PLDs or CPLDs, the synthesis tool may generate two-level sum-of-products equations. With ASICs, it may generate a net list that specifies how the gates should be interconnected.

6. Fitting / Placement & Routing. Maps the synthesized components onto physical devices.

7. Timing verification. At this stage, the actual circuit delays due to wire lengths, electrical loading, and other factors are known, so precise timing simulation can be performed. Study the circuit's operation including estimated delays, and we verify that the setup, hold, and other timing requirements for sequential devices like flip-flops are met.

Program Structure:

A key idea in VHDL is to define the interface of a hardware module while hiding its internal details. A VHDL entity is simply a declaration of a module's inputs and outputs, i.e. its external interface signals or ports. A VHDL architecture is a detailed description of the module's internal structure or behavior. You can think of the entity as a "wrapper" for the architecture, hiding the details of what's inside while providing the "hooks" for other modules to use it. VHDL actually allows you to define multiple architectures for a single entity, and it provides a configuration management facility that allows you to specify which one to use during a particular synthesis run.



In the text file of a VHDL program, the entity declaration and architecture definition are separated.

Example – VHDL program for an "inhibit" gate:

```
entity Inhibit is -- also known as 'BUT-NOT'
        port (X, Y: in BIT; -- as in 'X but not Y'
                Z: out BIT);
end Inhibit;
architecture Inhibit_arch of Inhibit is
        begin
                Z <= '1' when X='1' and Y='0' else '0';
end Inhibit_arch;
```

Keywords: *entity, port, is, in, out, end, architecture, begin, when, else,* and *not*.

Comments: begin with two hyphens (--) and end at the end of a line.

Identifiers: begin with a letter and contain letters, digits, and underscores. (An underscore may not follow another underscore or be the last character in an identifier.) Keywords and

identifiers are not case sensitive.

A basic entity declaration has the syntax as shown below:

entity *entity-name* is
    port (*signal-names* : *mode signal-type*;
        signal-names : *mode signal-type*;
        …
        signal-names : *mode signal-type*);
    end *entity-names*;

In addition to the keywords, an entity declaration has the following elements:

entity-name: A user-selected identifier to name the entity.

signal-names: A comma-separated list of one or more user-selected identifiers to name external-interface signals.

mode: One of four reserved words, specifying the signal direction:

in – The signal is an input to the entity.

out – The signal is an output of the entity. Note that the value of such a signal cannot be "read" inside the entity's architecture, only by other entities that use it.

buffer – The signal is an output of the entity, and its value can also be read inside the entity's architecture.

inout – The signal can be used as an input or an output of the entity. This mode is typically used for threestate input/output pins on PLDs.

signal-type: A built-in or user-defined signal type. See below.

A basic architecture definition has the syntax as shown below:

architecture *architecture-name* of *entity-name* is
type declarations
signal declarations
constant declarations
function definitions
procedure definitions
component declarations
begin
concurrent-statement

…

concurrent-statement

end *architecture-name*;

The *architecture-name* is a user-selected identifier, usually related to the entity name.

An architecture's external interface signals (ports) are inherited from the port-declaration part of its corresponding entity declaration. An architecture may also include signals and other declarations that are local to that architecture. Declarations common to multiple entities can be made in a separate "package" used by all entities. The declarations can appear in any order.

signal declaration:

signal *signal-names* : *signal-type*;

Types and constants:

All signals, variables, and constants must have an associated "type." The type specifies the set or range of values that the object can take on. Some predefined types are: bit, bit_vector, boolen, character, integer, real, and string.

A *user-defined enumerated types* have the following syntax:

type *type-name* is (*value-list*);

-- *value-list* is a comma-separated list of all possible values of the type.

-- subtypes of a type. Values must be a contiguous range of values of the base type, from *start* to *end*.

subtype sub*type-name* is *type-name* range *start* to *end*; -- ascending order
subtype sub*type-name* is *type-name* range *start* downto *end*; -- descending order
constant *constant-name*: *type-name* := *value*;

Example (enumerated type):

type traffic_light_state is (reset, stop, wait, go); type
std_logic is ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-');

subtype fourval_logic is std_logic range 'X' to 'Z'; subtype
bitnum is integer range 31 downto 0; constant bus_size:

integer := 32; -- width of component constant MSB: integer

:= bus_size-1; -- bit number of MSB constant Z: character

:= 'Z';-- synonym for Hi-Z value array types have the

following syntax:

type *type-name* is array (*start* to *end*) of *element-type*; type

type-name is array (*start* downto *end*) of *element-type*; type

type-name is array (*range-type*) of *element-type*;

type *type-name* is array (*range-type* range *start* to *end*) of *element-type*; type

type-name is array (*range-type* range *start* downto *end*) of *element-type*;

Example (array type):

type monthly_count is array (1 to 12) of integer;

type byte is array (7 downto 0) of std_logic;

constant word_len: integer := 32;

type word is array (word_len – 1 downto 0) of std_logic; -- note: a range value can be a simple expression.

constant num_regs: integer := 8;

type reg_file is array (1 to num_regs) of word; -- a two dimensional array

type statecount is array (traffic_light_state) of integer; -- an enumerated type can be the range.

Array elements are considered to be ordered from left to right, in the same direction as index range. Thus, the leftmost elements of arrays of types: monthly_count is 1, byte is 7, word is 31, reg_file is 1, and statecount is reset.

Array elements are accessed using parentheses. If M, B, *W*, R, and S are signals or variables of the five array types above, then

M(11), B(5), *W*(word_len – 5), R(0, 0), S(reset).

Array literals can be specified using different formats:

B := ('1', '1', '1', '1', '1', '1', '1', '1');

B := "11111111";

W := (0=>'0', 8=>'0', 16=>'0', 24=>'0', others => '1');

W := "11111101111111101111111011111110";

It is also possible to refer to a contiguous subset or *slice* of an array. eg.

M(6 to 9), B(3 downto 0), W(15 downto 8), S(stop to go).

You can combine arrays or array elements using the *concatenation operator* & which joins arrays and elements in the order written, from left to right. eg. '0' & '1' & "1Z" = "011Z"

B(6 downto 0) & B(7) yields a 1-bit left circular shift of the 8-bit array B.

The most important array type is:

type STD_LOGIC_VECTOR is array (natural range <>) of STD_LOGIC; -- example of an *unconstrained array type* – the range of the array is unspecified

Functions and procedures:

A *function* accepts a number of arguments and returns a result.

function *function-name* (
    signal-names : *signal-type*;
    signal-names : *signal-type*;
    …
    signal-names : *signal-type*
) return *return-type* is
    type declarations
    constant declarations
    variable declarations
    function definitions
    procedure definitions
begin
    sequential-statement
    …
    sequential -statement
end *function-name*;

A *procedure* does not return a result. Their arguments can be specified with type out or inout to "return" results.

A function can be used in the place of an expression.

A procedure can be used in the place of a statement.

Example – an "inhibit" function:

```
architecture Inhibit_archf of Inhibit is
function ButNot (A, B: bit) return bit is
begin
        if B = '0' then return A;
        else return '0';
        end if;
end ButNot;
begin
        Z <= ButNot(X, Y);
end Inhibit_archf;
```

Can have *overloaded* operators. The compiler picks the definition that matches the operand types in each use of the operator.

Libraries and Packages:

A *library* is a place where the VHDL compiler stores information about a particular design project, including intermediate files that are used in the analysis, simulation, and synthesis of the design. For a given VHDL design, the compiler automatically creates and uses a library named "work" under the current design directory.

Use the *library clause* at the beginning of the design file to use a standard library.

```
library ieee;
```

The clause "library work;" is included implicitly at the beginning of every VHDL design file. A library name in a design gives it access to any previously analyzed entities and architectures stored in the library, but it does not give access to type definitions.

A *package* is a file containing definitions of objects that can be used in other programs. The kind of objects that can be put into a package include signal, type, constant, function, procedure, and component declarations. Signals that are defined in a package are global" signals, available to any entity that uses the package. A design can use a package with the statement

use ieee.std_logic_1164.all;

Here, “ieee” is the name of the library. Use the file named “std_logic_1164” within this library. The “all” tells the compiler to use all of the definitions in this file.

The package syntax:

package *package-name* is

-- public section: visible in any design file that uses the package
type declarations
signal declarations
constant declarations
component declarations
function declarations -- lists only the function name, arguments, and type.
procedure declarations

end *package-name*;

package body *package-name* is

-- private section: local to the package
type declarations
constant declarations
function definitions -- the complete function definition
procedure definitions

end *package-name*;

The VHDL Design Elements

Structural design elements:

The body of an architecture is a series of concurrent statements. Each concurrent statement executes simultaneously with the other concurrent statements in the same architecture body. e.g. if the last statement updates a signal that is used by the first statement, then the simulator will go back to that first statement and update its results according to the signal that just changed. The simulator will keep propagating changes and updating results until the simulated circuit stabilizes.

Component statement (a concurrent statement):

label: *component-name* port map (*signal1, signal2, …, signal_n*);

label: *component-name* port map(*port1=>signal1*, *port2=>signal2*, …, *port_n=>signal_n*);

Component-name is the name of a previously defined entity that is to be used, or *instantiated*, within the current architecture body. Each instance must be named by a unique *label*. The *port map* introduces a list that associates ports of the named entity with signals in the current architecture. Before being instantiated in an architecture's definition, a component must be declared in a *component declaration* in an architecture's definition. It is essentially the same as the port-declaration part of the corresponding entity declaration. The components used in architecture may be ones that were previously defined as part of a design, or they may be part of a library.

A component declaration:

component *component-name*
port (*signal-names* : *mode signal-type*;
signal-names : *mode signal-type*;
…
signal-names : *mode signal-type*);
end component;

A VHDL architecture that uses components is often called a *structural description* or *structural design*, because it defines the precise interconnection structure of signals and entities that realize the entity. A pure structural description is equivalent to a schematic or a net list for the circuit. In some applications, it is necessary to create multiple copies of a particular structure within an architecture. e.g. an *n*-bit "ripple adder" can be created by cascading *n* "full adders."

A generate statement allows you to create such repetitive structures using a kind of "for loop."

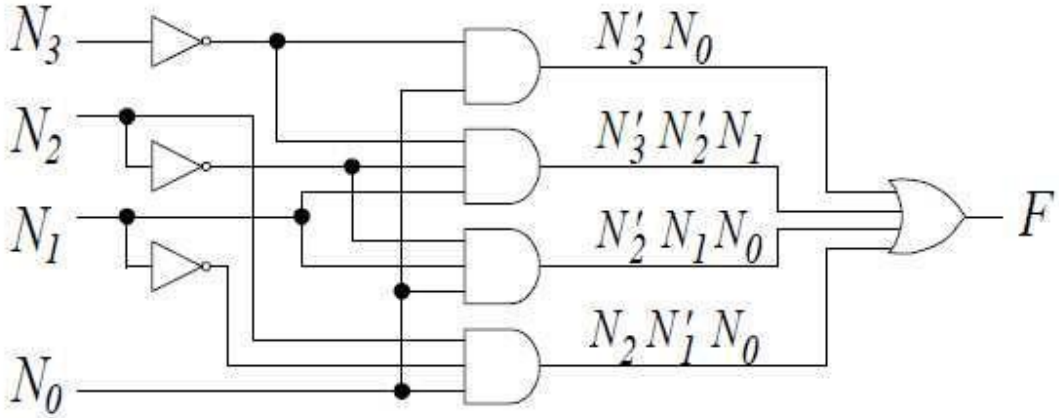
label: for *identifier* in *range* generate
concurrent-statement
end generate;

The value of a constant must be known at the time that a VHDL program is compiled. In many applications it is useful to design and compile an entity and its architecture while

leaving some of its parameters, such as bus width, unspecified. A generic statement lets you do this.

entity *entity-name* is
generic (*constant-names* : *constant-type*;
    constant-names : *constant-type*;
    ...
    constant-names : *constant-type*);
port (*signal-names* : *mode signal-type*;
    ...
    signal-names : *mode signal-type*);
end component;

Example – Given the schematic diagram for a prime-number detector, we can implement a



structural VHDL program for the prime-number detector.

```
library IEEE;
use IEEE.std_logic_1164.all;
entity prime is
port ( N: in STD_LOGIC_VECTOR (3 downto 0);
        F: out STD_LOGIC );
end prime;
architecture prime1_arch of prime is
        signal N3_L, N2_L, N1_L: STD_LOGIC;
        signal N3L_N0, N3L_N2L_N1, N2L_N1_N0, N2_N1L_N0: STD_LOGIC;
        component INV port (I: in STD_LOGIC; O: out STD_LOGIC);
                end component;
        component AND2 port (I0,I1: in STD_LOGIC; O: out STD_LOGIC);
                end component;
        component AND3 port (I0,I1,I2: in STD_LOGIC; O: out STD_LOGIC);
                end component;
```

```
        component OR4 port (I0,I1,I2,I3: in STD_LOGIC; O: out STD_LOGIC);
                end component;
begin
        U1: INV port map (N(3), N3_L);
        U2: INV port map (N(2), N2_L);
        U3: INV port map (N(1), N1_L);
        U4: AND2 port map (N3_L, N(0), N3L_N0);
        U5: AND3 port map (N3_L, N2_L, N(1), N3L_N2L_N1);
        U6: AND3 port map (N2_L, N(1), N(0), N2L_N1_N0);
        U7: AND3 port map (N(2), N1_L, N(0), N2_N1L_N0);
        U8: OR4 port map (N3L_N0, N3L_N2L_N1, N2L_N1_N0, N2_N1L_N0, F); end
prim1_arch;
```

Example (generate) – an 8-bit inverter generated from one component:

```
library IEEE;
use IEEE.std_logic_1164.all;
entity inv8 is
port ( X: in STD_LOGIC_VECTOR (1 to 8);
        Y: out STD_LOGIC_VECTOR (1 to 8) );
end inv8;
architecture inv8_arch of inv8 is
        component INV port (I: in STD_LOGIC; O: out STD_LOGIC);
                end component;
begin
        g1: for b in 1 to 8 generate
        U1: INV port map (X(b), Y(b));
        end generate;
end inv8_arch;
```

Example (generic, generate) – an arbitrary-width bus inverter:

```
library IEEE;
use IEEE.std_logic_1164.all;
entity businv is
generic ( WIDTH: positive);
port ( X: in STD_LOGIC_VECTOR (WIDTH-1 downto 0);
          Y: out STD_LOGIC_VECTOR (WIDTH-1 downto 0) );
```

```
end businv;
architecture businv_arch of businv is
        component INV port (I: in STD_LOGIC; O: out STD_LOGIC);
                end component;
begin
        g1: for b in WIDTH-1 downto 0 generate
        U1: INV port map (X(b), Y(b));
        end generate;
end businv_arch;
```

Dataflow design elements:

Several additional concurrent statements allow VHDL to describe a circuit in terms of the flow of data and operations on it within the circuit. This style is called a *dataflow description* or *dataflow design*.

concurrent signal-assignment statement:

signal-name <= *expression*;

The type for expression must be identical or a sub-type of signal-name. In the case

of arrays, both the element type and the length must match; however, the index range and

direction need not match.

Example (concurrent signal-assignment) – Dataflow VHDL architecture for the prime-number detector:

```
library IEEE;
use IEEE.std_logic_1164.all;
entity prime is
port ( N: in STD_LOGIC_VECTOR (3 downto 0);
        F: out
STD_LOGIC ); end
prime;
architecture prime2_arch of prime is
```

```
signal N3L_N0, N3L_N2L_N1, N2L_N1_N0, N2_N1L_N0:
STD_LOGIC; begin
        N3L_N0 <= not N(3) and N(0);
        N3L_N2L_N1 <= not N(3) and not N(2) and
        N(1); N2L_N1_N0 <= not N(2) and N(1) and
        N(0); N2_N1L_N0 <= N(2) and not N(1) and
        N(0);
        F <= N3L_N0 or N3L_N2L_N1 or N2L_N1_N0 or
N2_N1L_N0; end prime2_arch;
```

conditional signal-assignment statement:

signal-name <= *expression* when *boolean-expression*
else *expression* when *boolean-expression* else
…
expression when *boolean-expression* else
expression;

Boolean operators: and, or, not.

Relational operators: =, /= (not equal), >, >=, <, <=.
The combined set of conditions in a single statement should cover all possible input combinations.

Example (conditional signal-assignment) – Dataflow VHDL architecture for the prime-number detector:

```
library IEEE;
use IEEE.std_logic_1164.all;
entity prime is
port ( N: in STD_LOGIC_VECTOR (3 downto 0);
        F: out
STD_LOGIC ); end
prime;
architecture prime3_arch of prime is
        signal N3L_N0, N3L_N2L_N1, N2L_N1_N0, N2_N1L_N0: STD_LOGIC;
begin
        N3L_N0 <= '1' when N(3)='0' and N(0)='1' else '0';
        N3L_N2L_N1 <= '1' when N(3)='0' and N(2)='0' and N(1)='1' else
        '0'; N2L_N1_N0 <= '1' when N(2)='0' and N(1)='1' and N(0)='1'
```

```
        else '0'; N2_N1L_N0 <= '1' when N(2)='1' and N(1)='0' and
        N(0)='1' else '0'; F <= N3L_N0 or N3L_N2L_N1 or N2L_N1_N0 or
        N2_N1L_N0;
end prime3_arch;
```

selected signal-assignment statement:

with *expression* select

signal-name <= *signal-value* when *choices*,

signal-value when *choices*,

…

signal-value when *choices*,

The *choices* may be a single value of *expression* or a list of values separated by vertical bars (|).

The *choices* for the entire statement must be mutually exclusive and all inclusive.

Example (concurrent signal-assignment) – Dataflow VHDL architecture for the prime-number detector:

```
architecture prime4_arch of prime is
begin
        with N select
                F <= '1' when "0001",
                        '1' when "0010",
                        '1' when "0011" | "0101" | "0111",
                        '1' when "1011" | "1101",
                        '0' when others;
end prime4_arch;
```

Example – A more behavioral description of the prime-number detector:

```
architecture prime5_arch of prime is
begin
        with CONV_INTEGER(N) select -- convert STD_LOGIC_VECTOR to INTEGER

                F <= '1' when 1 | 2 | 3 | 5 | 7 | 11 | 13,
                        '0' when others;
end prime5_arch;
```

Behavioral design elements:

VHDL's key behavioral element is the "process." A *process* is a collection of sequential statements that executes in parallel with other concurrent statements and other processes. process statement:

process (*signal-name, signal-name, …, signal-name*)
    type declarations
    constant declarations
    variable declarations
    function definitions
    procedure definitions
begin
    sequential-statement
    …
    sequential -statement
end process;

A process statement can be used anywhere that a concurrent statement can be used.

A process is either *running* or *suspended*.

The list of signals in the process definition, called the *sensitivity list*, determines when the process runs. A process initially is suspended. When any signal in its sensitivity list changes value, the process resumes execution, starting with its first sequential statement and continuing until the end. If any signal in the sensitivity list change value as a result of running the process, it runs again. This continues until none of the signals change value.

In simulation, all of this happens in zero simulated time. The sensitivity list is optional; a process without a sensitivity list starts running at time zero in simulation. One application of such a process is to generate input waveforms in a test bench. A process may not declare signals. It can only declare variables.

A VHDL *variable* is similar to signals, except that they usually don't have physical significance in a circuit. They are used only in functions, procedures, and processes to

keep track of the state within a process and is not visible outside of the process.

variable declaration:

variable *variable-names* : *variable-type*;

Example – A process-based dataflow VHDL architecture for the prime-number detector:

```
library IEEE;
use IEEE.std_logic_1164.all;
entity prime is
port ( N: in STD_LOGIC_VECTOR (3 downto 0);
        F: out
STD_LOGIC ); end
prime;
architecture prime6_arch of
prime is begin
process(N)
        variable N3L_N0, N3L_N2L_N1, N2L_N1_N0, N2_N1L_N0: STD_LOGIC;
begin
        N3L_N0 := not N(3) and N(0);
        N3L_N2L_N1 := not N(3) and not N(2) and
        N(1); N2L_N1_N0 := not N(2) and N(1) and
        N(0); N2_N1L_N0 := N(2) and not N(1) and
        N(0);
        F <= N3L_N0 or N3L_N2L_N1 or N2L_N1_N0 or
N2_N1L_N0; end process;
end prime6_arch;
```

VHDL has several kinds of sequential statements.

sequential signal-assignment statement – same as the concurrent signal-assignment statement. It is sequential because it occurs in the body of a process rather than an architecture. Since we cannot have signal definitions in a process, the *signal-name* must be one from the sensitivity list.

signal-name <= *expression*;

variable-assignment statement –

variable-name := *expression*; -- notice the different assignment operator for variables.

if statement:

if *boolean-expression* then *sequential-statement*

end if;

if *boolean-expression* then *sequential-statement* else *sequential-statement*

end if;

if *boolean-expression* then *sequential-statement*

elsif *boolean-expression* then *sequential-statement* -- note the spelling for the keyword elsif …

elsif *boolean-expression* then *sequential-statement*

else *sequential-statement*

end if;

case statement:

case *expression* is

when *choices* => *sequential-statements* -- one or more sequential statements can be used …

when *choices* => *sequential-statements*

when others => *sequential-statements* -- optional; to denote all values that have not yet been covered.

end case;

A *case* statement is usually more readable and may yield a better synthesized circuit.

The *choices* must be mutually exclusive and include all possible values of *expression*'s

type.

loop statements:

for *identifier* in *range* loop -- for loop

    sequential-statement

    …

    sequential-statement

end loop;

loop -- infinite loop

    sequential-statement

    …

    sequential-statement

end loop;

exit; -- transfers control to the statement immediately following the loop end
-- causes any remaining statements in the loop to be bypassed and begins the next

iteration of the loop.

The *identifier* is declared implicitly by its appearance in the *for loop* and has the same type

as *range*. This variable may be used within the loop's sequential statements, and it steps

through all of the values in *range*, from left to right, one per iteration.

Example – Prime-number detector using an if statement:

```
library IEEE;
use IEEE.std_logic_1164.all;
entity prime is
port ( N: in STD_LOGIC_VECTOR (3 downto 0);
        F: out
STD_LOGIC ); end
prime;
architecture prime7_arch of
prime is begin
   process(N)
    variable NI:
    integer;
  begin
              NI :=
              CONV_INTEGER(N); if
              NI=1 or NI=2 then F <=
              '1';
              elsif NI=3 or NI=5 or NI=7 or NI=11 or NI=13 then F <=
              '1'; else F <= '0';
               en
         d if; end
         process;
end prime7_arch;
```

Example – Prime-number detector using an case statement:

```
library IEEE;
```

```
use IEEE.std_logic_1164.all;
entity prime is
port ( N: in STD_LOGIC_VECTOR (3 downto 0);
        F: out STD_LOGIC );
end prime;
architecture prime8_arch of prime is
begin
  process(N)
   begin
    case CONV_INTEGER(N) is
       when 1 => F <= '1';
                when 2 => F <= '1';
                when 3 | 5 | 7 | 11 | 13 => F <= '1';
                when others => F <= '0';
       end case;
 end process;
end prime8_arch;
```

Example – A truly behavioral description of a prime-number detector:

```
library IEEE;
use IEEE.std_logic_1164.all;
entity prime9 is
        port ( N: in STD_LOGIC_VECTOR (15 downto 0);
                 F: out STD_LOGIC );
end prime9;
architecture prime9_arch of prime9 is
begin
  process(N)
    variable NI: integer;
         variable prime: boolean;
        begin
                NI := CONV_INTEGER(N);
                prime := true;
                if NI=1 or NI=2 then null; -- take case of boundary cases
                else
                for i in 2 to 253 loop
                if NI mod i = 0 then
                prime := false;
```

```
                exit;
                end if;
                end loop;
                end if;
                if prime then F <= '1';
                else F <= '0';
        end if;
  end process;
end prime9 _arch;
```

UNIT-V
COMBINATIONAL LOGIC DESIGN

Decoders:

A decoder is a multiple-input, multiple-output logic circuit that converts coded inputs into coded outputs, where the input and output codes are different. The input code generally has fewer bits than the output code, and there is a one-to one mapping from input code words into output code words. In a one-to-one mapping, each input code word produces a different output code word.

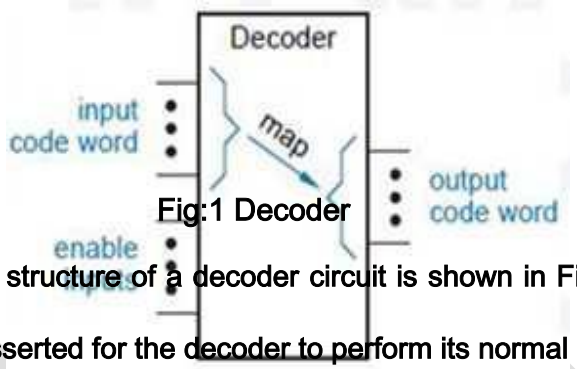


Fig:1 Decoder

The general structure of a decoder circuit is shown in Figure 1. The enable inputs, if present, must be asserted for the decoder to perform its normal mapping function. Otherwise, the decoder maps all input code words into a single, "disabled," output code word. The most commonly used output code is a 1-out-of-m code, which contains m bits, where one bit is asserted at any time. Thus, in a 1-out-of-4 code with active-high outputs, the code words are 0001, 0010, 0100, and 1000. With active-low outputs, the code words are 1110, 1101, 1011, and 0111.

Binary Decoders

The most common decoder circuit is an n-to-2n decoder or binary decoder. Such a decoder has an n-bit binary input code and a 1-out-of-2n output code. A binary decoder is used when you need to activate exactly one of 2n outputs based on an n-bit input value

Inputs			*Outputs*			
EN	I1	I0	Y3	Y2	Y1	Y0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

Table 1: 2 to 4 decoder

Table 1 is the truth table of a 2-to-4 decoder. The input code word 1,I0 represents an integer in the range 0-3. The output code word Y3,Y2,Y1,Y0 has Yi equal to 1 if and only if the input code word is the binary representation of i and the enable input EN is 1. If EN

is 0, then all of the outputs are 0. A gate-level circuit for the 2-to-4 decoder is shown in Figure 2 Each AND gate decodes one combination of the input code word I1,I0.

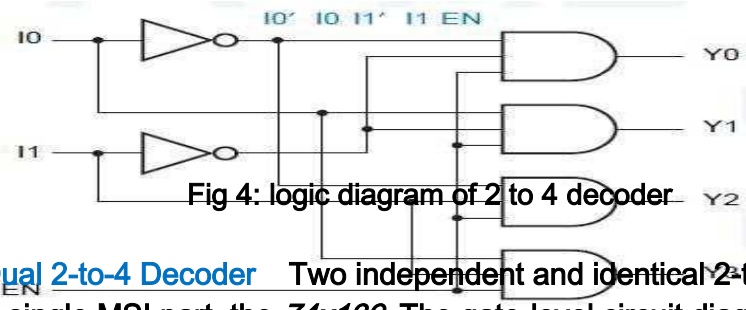


Fig 4: logic diagram of 2 to 4 decoder

The 74x139 Dual 2-to-4 Decoder Two independent and identical 2-to-4 decoders are contained in a single MSI part, the *74x139*. The gate-level circuit diagram for this IC is shown in Figure.

1. The outputs and the enable input of the '139 are active-low.
2. Most MSI decoders were originally designed with active-low outputs, since TTL inverting gates are generally faster than non inverting ones.

3. '139 has extra inverters on its select inputs. Without these inverters, each select input would present three AC or DC loads instead of one, consuming much more of the fanout budget of the device that drives it.

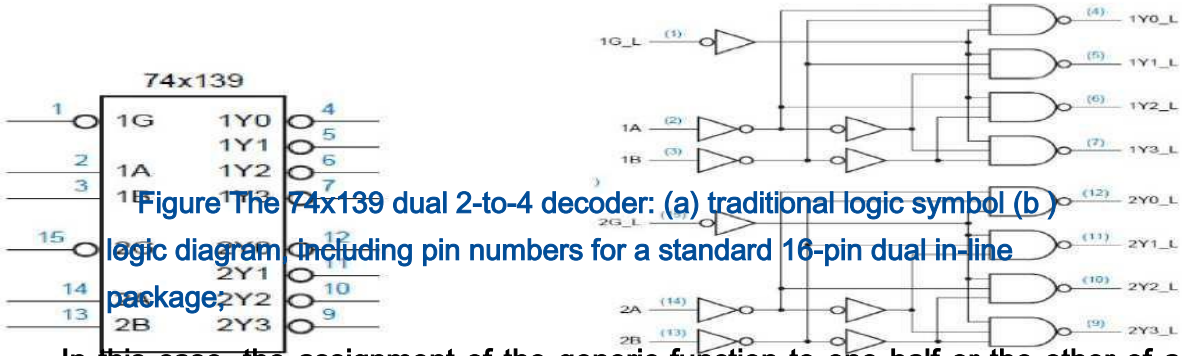


Figure The 74x139 dual 2-to-4 decoder: (a) traditional logic symbol (b) logic diagram, including pin numbers for a standard 16-pin dual in-line package;

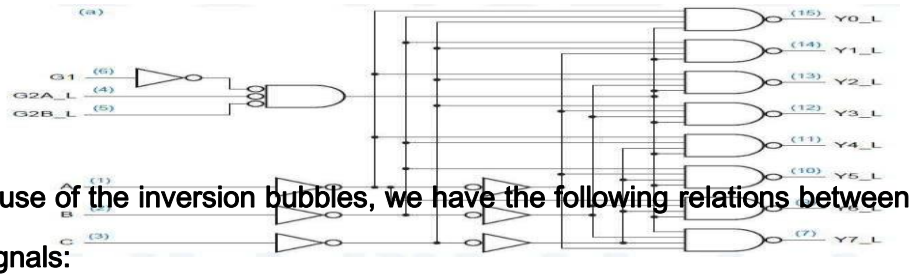
In this case, the assignment of the generic function to one half or the other of a particular '139 package can be deferred until the schematic is completed Table is the truth table for a 74x139-type decoder.

The 74x138 3-to-8 Decoder

The *74x138* is a commercially available MSI 3-to-8 decoder whose gate-level circuit diagram and symbol are shown in Figure 7; its truth table is given in Table. Like the 74x139, the 74x138 has active-low outputs, and it has three enable inputs (G1, /G2A, /G2B), all of

(b) 74x138 Inputs Outputs

which must be asserted for the selected output to be asserted.



However, because of the inversion bubbles, we have the following relations between internal and external signals:

$$G2A = G2A\_L'$$
$$G2B = G2B\_L'$$

Therefore, if we're interested, we can write the following equation for the external output signal

$$Y5 = Y5\_L'$$

Y5_L in terms of external input signals:

$$Y5\_L = Y5' = (G1 \cdot G2A\_L' \cdot G2B\_L' \cdot C \cdot B' \cdot A)'$$

On the surface, this equation doesn't resemble what you might expect for a decoder, since it is a logical sum rather than a product. However, if you practice bubble-to-bubble logic design, you don't have to worry about this; you just give the output signal an active-low name and remember that it's active low when you connect it to other inputs.

Cascading Binary Decoders

Multiple binary decoders can be used to decode larger code words. Figure 5-38 shows how two 3-to-8 decoders can be combined to make a 4-to-16 decoder. The availability of both active-high and active-low enable inputs on the 74x138 makes it possible to enable one or the other directly based on the state of the most significant input bit. The top decoder (U1) is enabled when N3 is 0, and the bottom one (U2) is enabled when N3 is

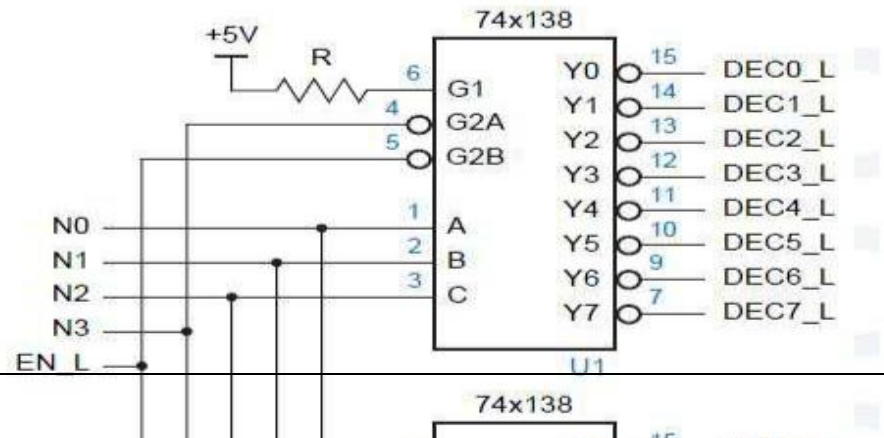
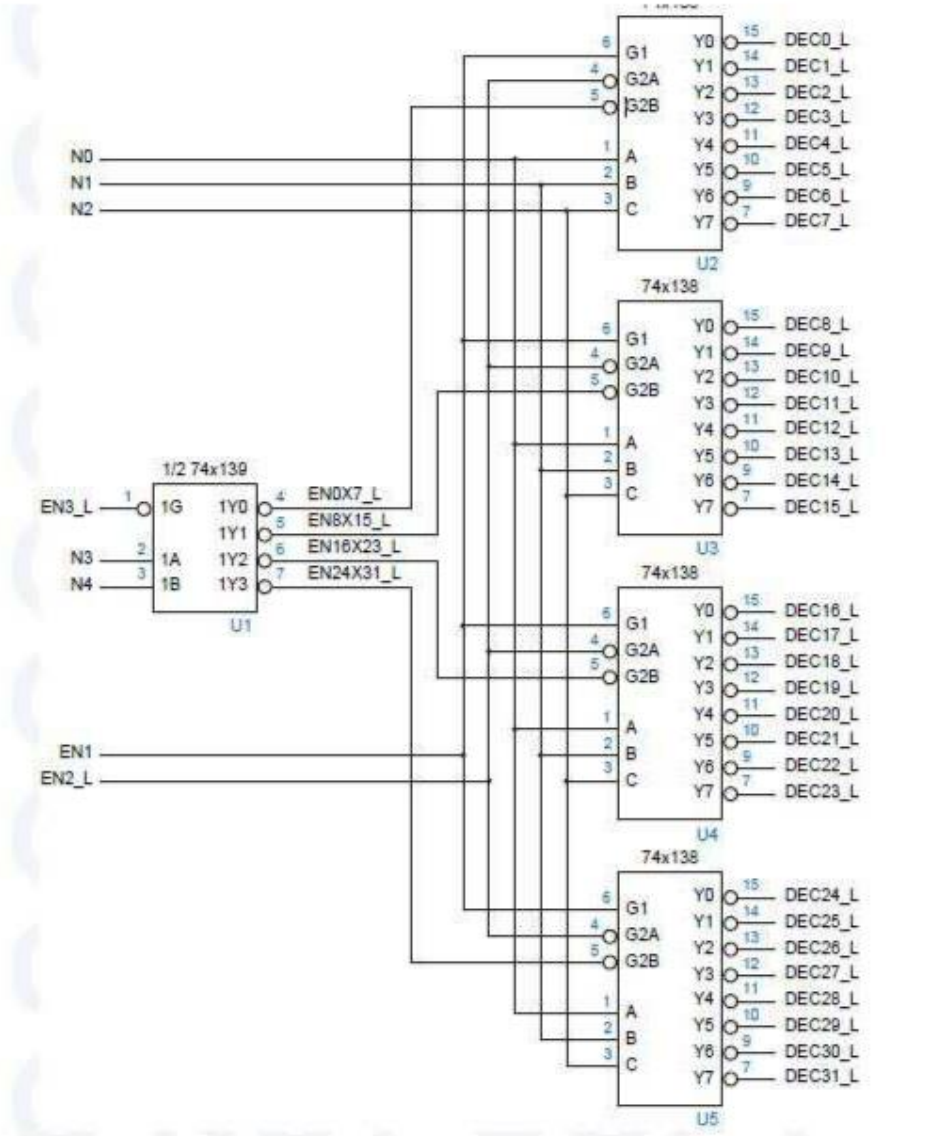
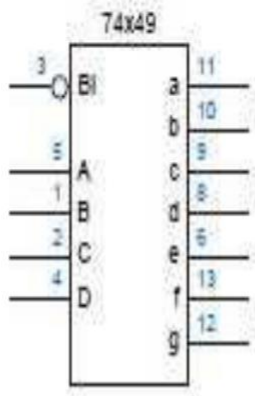


Figure Design of a 4-to-16 decoder using 74x138s.



Look at your wrist and you'll probably see a *seven-segment display*. This type of display, which normally uses light-emitting diodes (LEDs) or liquid-crystal display (LCD) elements, is used in watches, calculators, and instruments to display decimal data. A digit is displayed by illuminating a subset of the seven line segments shown in Figure (a). A *seven-segment decoder* has 4-bit BCD as its input code and the "seven segment code,"



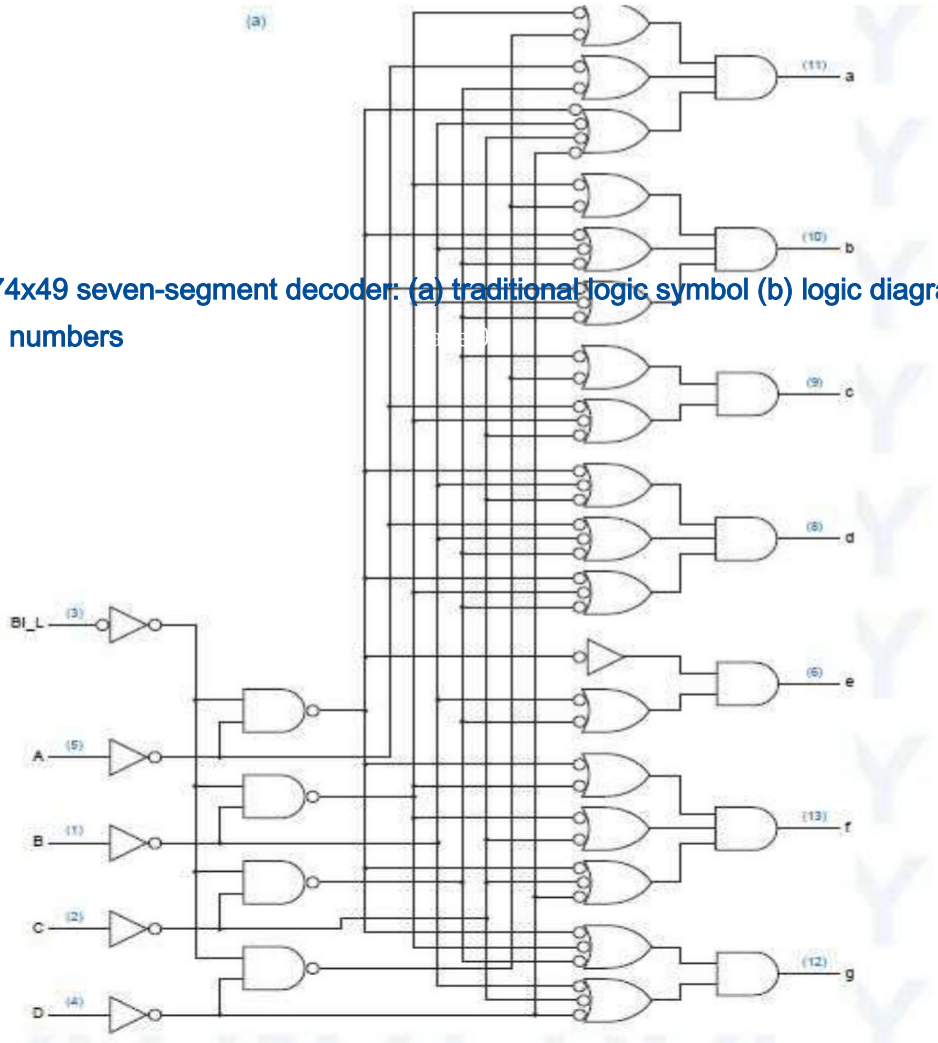


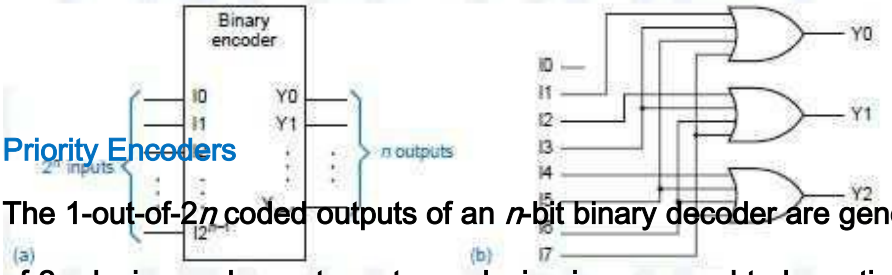
Figure The 74x49 seven-segment decoder: (a) traditional logic symbol (b) logic diagram, including pin numbers

Encoders

A decoder's output code normally has more bits than its input code. If the device's output code has *fewer* bits than the input code, the device is usually called an *encoder*. Probably the simplest encoder to build is a 2*n*-to-*n* or *binary encoder*. As shown in Figure(a), it has just the opposite function as a binary *de*coder— its input code is the 1-out-of-2*n* code and its output code is *n*-bit binary. The equations for an 8-to-3 encoder with inputs I0–I7 and outputs Y0–Y2 are given below:

$$Y0 = I1 + I3 + I5 + I7$$
$$Y1 = I2 + I3 + I6 + I7$$
$$Y2 = I4 + I5 + I6 + I7$$

The corresponding logic circuit is shown in (b). In general, a 2*n*-to-*n* encoder can be built from *n* 2*n* 1-input OR gates. Bit *i* of the input code is connected to OR gate *j* if bit *j* in the binary representation of *i* is 1.



Priority Encoders

The 1-out-of-2*n* coded outputs of an *n*-bit binary decoder are generally used to control a set of 2*n* devices, where at most one device is supposed to be active at any time. Conversely, consider a system with 2*n* *inputs*, each of which indicates a request for service. This structure is often found in microprocessor input/output subsystems, where the inputs might be interrupt requests. In this situation, it may seem natural to use a binary encoder. to observe the inputs and indicate which one is requesting service at any time. However, this encoder works properly only if the inputs are guaranteed to be asserted at most one at a time. If multiple requests can be made simultaneously, the encoder gives undesirable results. For example, suppose that inputs I2 and I4 of the 8-to-3 encoder are both 1; then the output is 110, the binary encoding of 6.

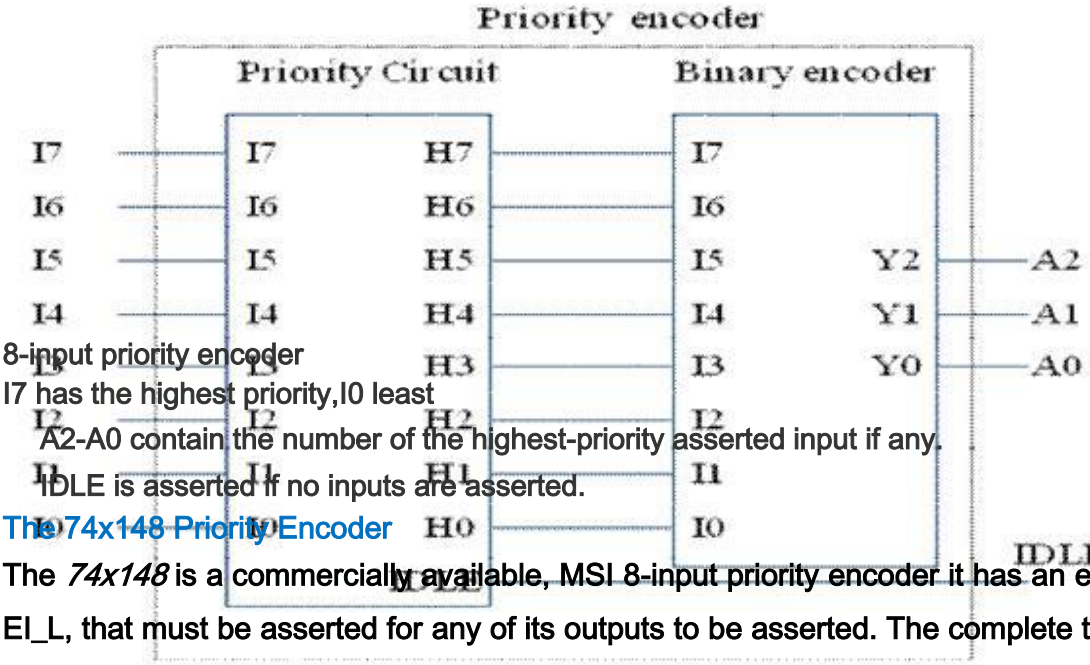
Y0=I1+I3+I5+I7
Y1=I2+I3+I6+I7
Y2=I4+I5+I6+I7

Either 2 or 4, not 6, would be a useful output in the preceding example, but how can the encoding device decide which? The solution is to assign *priority* to the input lines, so that

when multiple requests are asserted, the encoding device produces the number of the highest-priority requestor. Such a device is called a *priority encoder*.

Input I7 has the highest priority. Outputs A2–A0 contain the number of the highest-priority asserted input, if any. The IDLE output is asserted if no inputs are asserted. In order to write logic equations for the priority encoder's outputs, we first define eight intermediate variables H0–H7, such that Hi is 1 if and only if Ii is the highest priority 1 input: Using these signals, the equations for the A2–A0 outputs are similar to the ones for a simple binary encoder:

H7=I7 (Highest Priority)
H6=I6.I7'
H5=I5.I6'.I7'
H4=I4.I5'.I6'.I7'
H3=I3.I4'.I5'.I6'.I7'
H2=I2.I3'.I4'.I5'.I6'.I7'
H1=I1. I2'.I3'.I4'.I5'.I6'.I7'
H0=I0.I1'. I2'.I3'.I4'.I5'.I6'.I7'
IDLE= I0'.I1'. I2'.I3'.I4'.I5'.I6'.I7'
- Encoder
A0=Y0=H1+H3+H5+H7
A1=Y1=H2+H3+H6+H7
A2=Y2=H4+H5+H6+H7



8-input priority encoder
I7 has the highest priority,I0 least
A2-A0 contain the number of the highest-priority asserted input if any.
IDLE is asserted if no inputs are asserted.

The 74x148 Priority Encoder

The *74x148* is a commercially available, MSI 8-input priority encoder it has an enable input, EI_L, that must be asserted for any of its outputs to be asserted. The complete truth table is given in Table. Instead of an IDLE output, the '148 has a GS_L output that is asserted when the device is enabled and one or more of the request inputs is asserted. The manufacturer

calls this “Group Select,” but it’s easier to remember as “Got Something.” The EO_L signal is an enable *output* designed to be connected to the EI_L input of another ’148 that handles lower-priority requests. EO is asserted if EI_L is asserted but no request input is asserted; thus, a lower-priority ’148 may be enabled

Inputs									Outputs				
/EI	/I0	/I1	/I2	/I3	/I4	/I5	/I6	/I7	/A2	/A1	/A0	/GS	/EO
1	x	x	x	x	x	x	x	x	1	1	1	1	1
0	x	x	x	x	x	x	x	0	0	0	0	0	1
0	x	x	x	x	x	x	0	1	0	0	1	0	1
0	x	x	x	x	x	0	1	1	0	1	0	0	1
0	x	x	x	x	0	1	1	1	0	1	1	0	1
0	x	x	x	0	1	1	1	1	1	0	0	0	1
0	x	0	1	1	1	1	1	1	1	0	1	0	1
0	0	1	1	1	1	1	1	1	1	1	0	0	1
0	1	1	1	1	1	1	1	1	1	1	1	1	0

Figure shows how four 74x148s can be connected in this way to accept 32 request inputs and produce a 5-bit output, RA4–RA0, indicating the highest-priority requestor. Since the A2–A0 outputs of at most one ’148 will be enabled at any time, the outputs of the individual ’148s can be ORed to produce RA2–RA0. Likewise, the individual GS_L outputs can be combined in a 4-to-2 encoder to produce RA4 and RA3. The RGS output is asserted if any GS output is asserted.

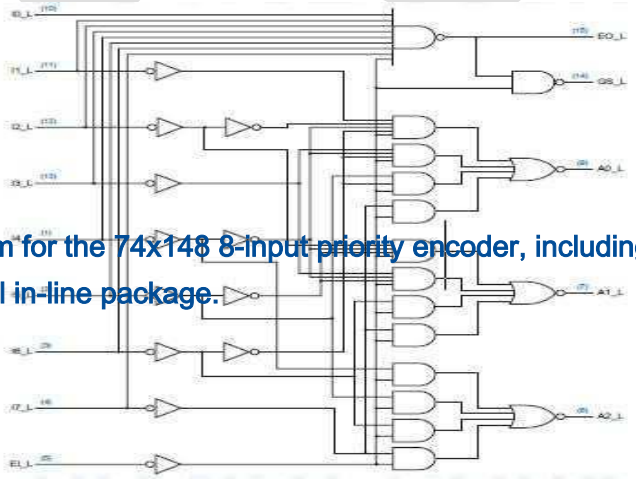


Figure Logic diagram for the 74x148 8-input priority encoder, including pin numbers for a standard 16-pin dual in-line package.

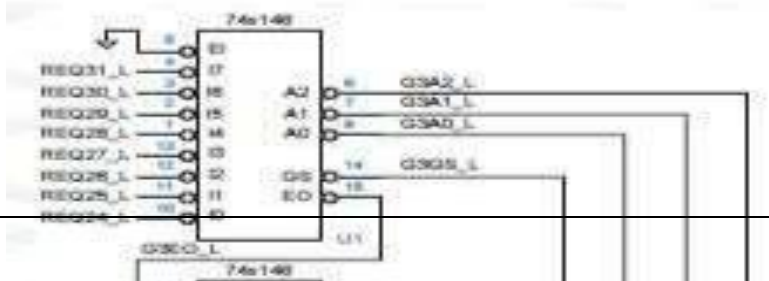
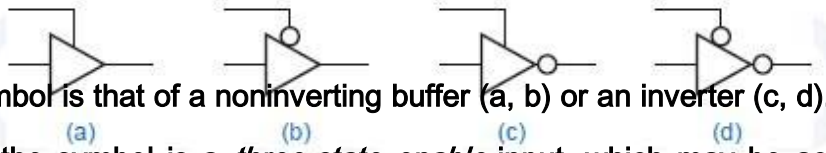


Figure Four 74x148s cascaded to handle 32 requests.

Three-State Devices:

Three-State Buffers

The most basic three-state device is a *three-state buffer*, often called a *three-state driver*. The logic symbols for four physically different three-state buffers are shown in Figure



The basic symbol is that of a noninverting buffer (a, b) or an inverter (c, d). The extra signal at the top of the symbol is a *three-state enable* input, which may be active high (a,c) or active low (b, d). When the enable input is asserted, the device behaves like an ordinary buffer or inverter. When the enable input is negated, the device output "floats"; that is, it goes to a high impedance (Hi-Z), disconnected state and functionally behaves as if it weren't even there.

Both enable inputs, G1_L and G2_L, must be asserted to enable the device's three-state outputs. The little rectangular symbols inside the buffer symbols indicate *hysteresis*, an electrical characteristic of the inputs that improves noise immunity. The 74x541 inputs typically have 0.4 volts of hysteresis.

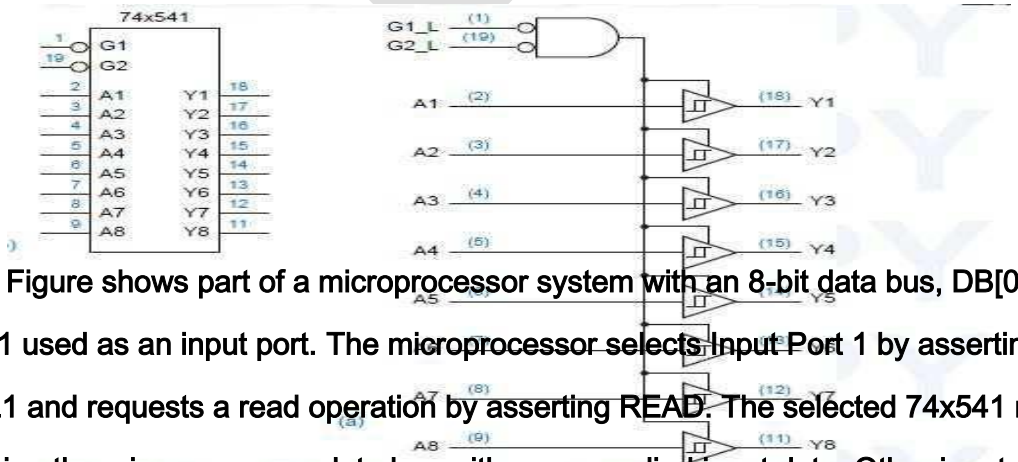
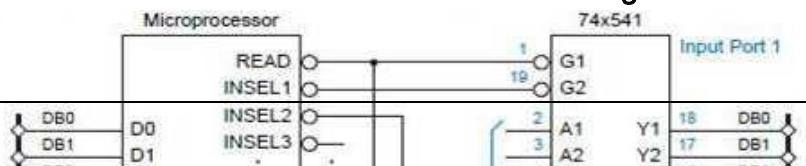
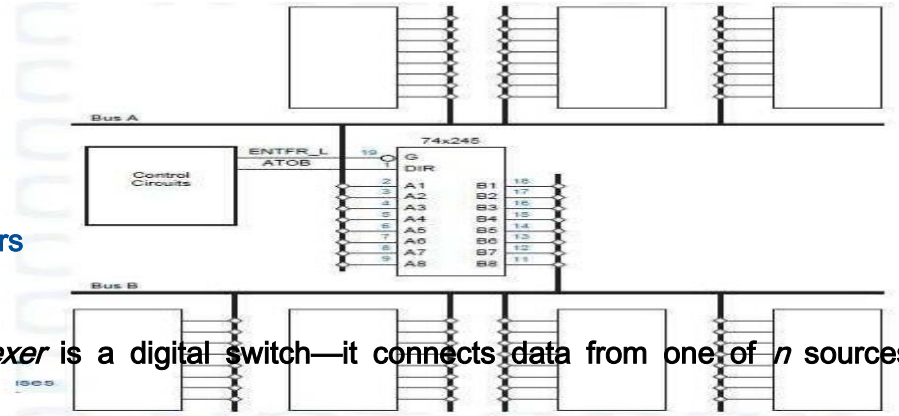


Figure shows part of a microprocessor system with an 8-bit data bus, DB[0–7], and a 74x541 used as an input port. The microprocessor selects Input Port 1 by asserting INSEL1 and requests a read operation by asserting READ. The selected 74x541 responds by driving the microprocessor data bus with user supplied input data. Other input ports may be selected when a different INSEL line is asserted along with READ.



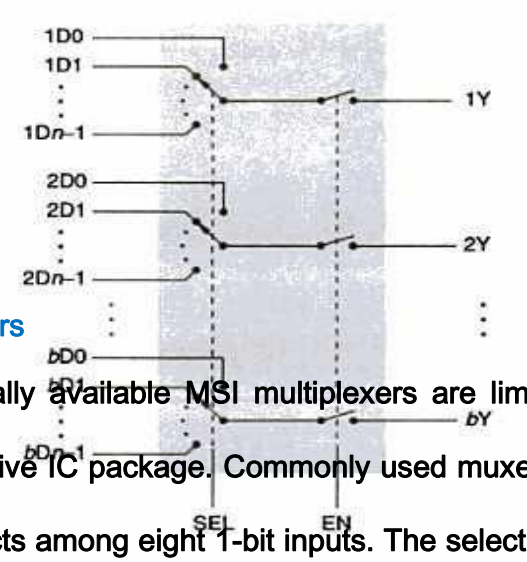
A bus transceiver is typically used between two *bidirectional buses*, as shown in Figure. Three different modes of operation are possible, depending on the state of G_L and DIR, as shown in Table. As usual, it is the designer's responsibility to ensure that neither bus is ever driven simultaneously by two devices. However, independent transfers where both buses are driven at the same time may occur when the transceiver is disabled, as indicated in the last row of the table.



Multiplexers

A *multiplexer* is a digital switch—it connects data from one of *n* sources to its output. Figure(a) shows the inputs and outputs of an *n*-input, *b*-bit multiplexer. There are *n* sources of data, each of which is *b* bits wide. A multiplexer is often called a *mux* for short. A multiplexer can use addressing bits to select one of several input bits to be the output. A selector chooses a single data input and passes it to the MUX output. It has one output selected at a time. Figure shows a switch circuit that is roughly equivalent to the multiplexer. However, unlike a mechanical switch, a multiplexer is a unidirectional device: information flows only from inputs (on the left) to outputs (on the right. Multiplexers are obviously useful devices in any application in which data must be switched from multiple sources to a destination. A common application in computers is the multiplexer between the processor's registers and its arithmetic logic unit (ALU). For example, consider a 16-bit processor in

which each instruction has a 3-bit field that specifies one of eight registers to use. This 3-bit field is connected to the select inputs of an 8-input, 16-bit multiplexer. The multiplexer's data inputs are connected to the eight registers, and its data outputs are connected to the ALU to execute the instruction using the selected register.



Standard MSI Multiplexers

The sizes of commercially available MSI multiplexers are limited by the number of pins available in an inexpensive IC package. Commonly used muxes come in 16-pin packages. Shown in fig which selects among eight 1-bit inputs. The select inputs are named C, B, and A, where C is most significant numerically. The enable input EN_L is active low; both active-high (Y) and active-low (Y_L) versions of the output are provided.

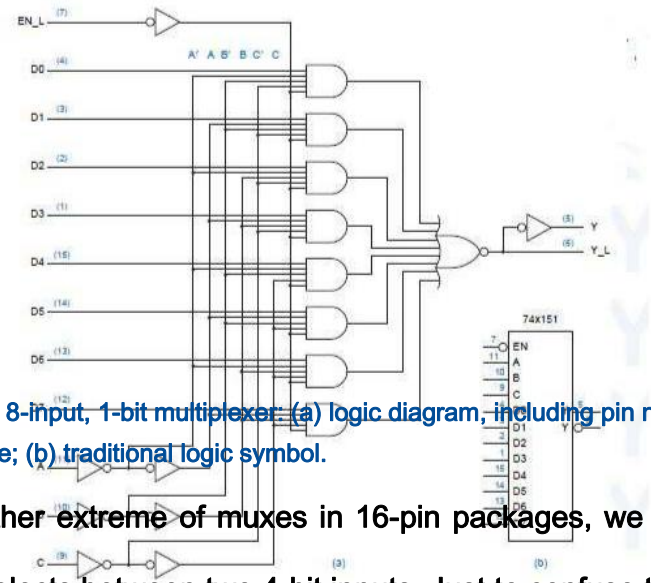
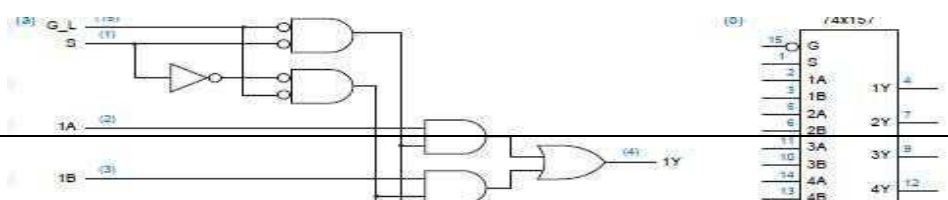
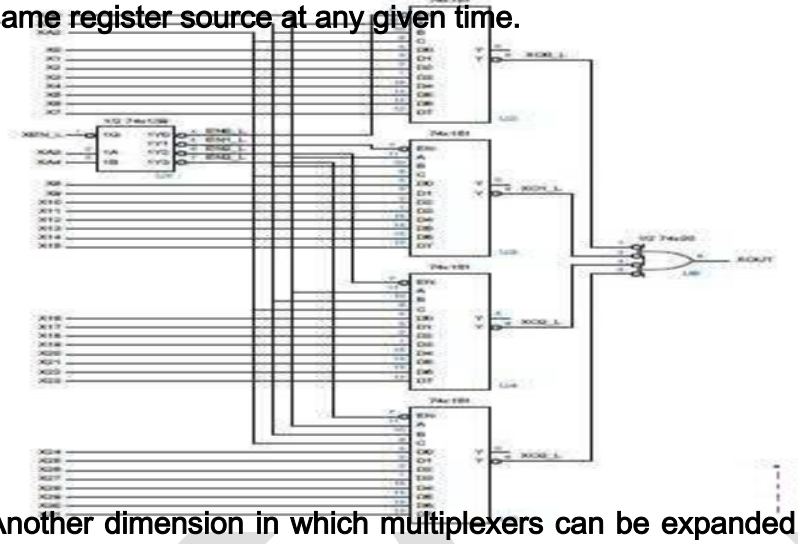


Figure The 74x151 8-input, 1-bit multiplexer: (a) logic diagram, including pin numbers for a standard 16-pin dual in-line package; (b) traditional logic symbol.

At the other extreme of muxes in 16-pin packages, we have the *74x157*, shown in Figure, which selects between two 4-bit inputs. Just to confuse things, the manufacturer has named the select input S and the active-low enable input G_L. Also note that the data sources are named A and B.



Expanding Multiplexers Seldom does the size of an MSI multiplexer match the characteristics of the problem at hand. For example, we suggested earlier that an 8-input, 16-bit multiplexer might be used in thedesign of a computer processor. This function could be performed by 16 74x151 8-input, 1- bit multiplexers or equivalent ASIC cells, each handling one bit of all the inputs and the output. The processor's 3-bit register-select field would be connected to the A, B, and C inputs of all 16 muxes, so they would all select the same register source at any given time.



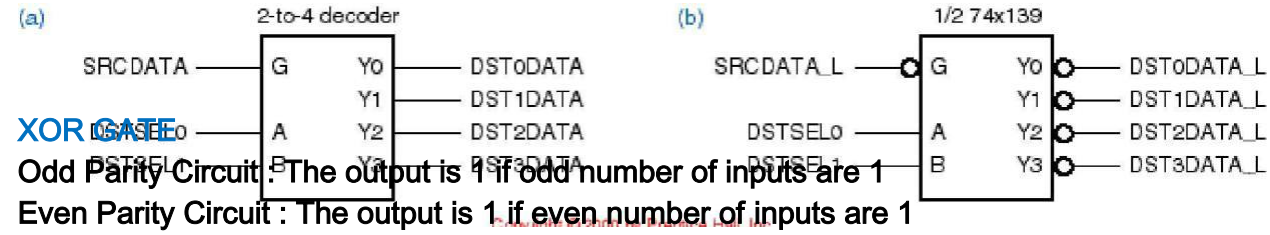
Another dimension in which multiplexers can be expanded is the number of data sources. For example, suppose we needed a 32-input, 1-bit multiplexer. Figure shows one way to build it. Five select bits are required. A 2-to-4 decoder (one-half of a 74x139) decodes the two highorder select bits to enable one of four 74x151 8-input multiplexers. Since only one '151 is enabled at a time, the '151 outputs can simply be ORed to obtain the final output. The 32-to-1 multiplexer can also be built using 74x251s. The circuit is identical to Figure, except that the output NAND gate is eliminated. Instead, the Y (and, if desired, Y_L) outputs of the four '251s are simply tied together. The '139 decoder ensures that at most one of the '251s has its threestate outputs enabled at any time. If the '139 is disabled (XEN_L is negated), then all of the '251s are disabled, and the XOUT and XOUT_L outputs are undefined. However, if desired, resistors may be connected from each of these signals to 5 volts to pull the output HIGH in this case.

Multiplexers, Demultiplexers, and Buses

A multiplexer can be used to select one of *n* sources of data to transmit on a bus. At the far

end of the bus, a *demultiplexer* can be used to route the bus data to one of *m* destinations. Such an application, using a 1-bit bus. In fact, block diagrams for logic circuits often depict multiplexers and demultiplexers , to suggest visually how a selected one of multiple data sources gets directed onto a bus and routed to a selected one of multiple destinations. The function of a demultiplexer is just the inverse of a multiplexer's. For example, a 1-bit, *n*output demultiplexer has one data input and *s* inputs to select one of *n* 2*s* data outputs. In normal operation, all outputs except the selected one are 0; the selected output equals the data input. This definition may be generalized for a *b*-bit, *n*-output demultiplexer; such a device has *b* data inputs, and its *s* select inputs choose one of *n* 2*s* sets of *b* data outputs. A binary decoder with an enable input can be used as a demultiplexer, as shown in Figure. The decoder's enable input is connected to the data line, and its select inputs determine which of its output lines is driven with the data bit. The remaining output lines are negated. Thus, the 74x139 can be used as a 2-bit, 4-output demultiplexer with active-low data inputs and outputs, and the 74x138 can be used as a 1-bit, 8-output demultiplexer. In fact, the manufacturer's catalog typically lists these ICs as "decoders/demultiplexers."

A Mux is used to select one of n sources of data to transmit on a bus. A demultiplexer can be used to route the bus data to one of m destinations. Just the inverse of a mux. A binary decoder with an enable input can be used as a Demux. E.g. 74x139 can be used as a 2-bit, 4- output Demux.

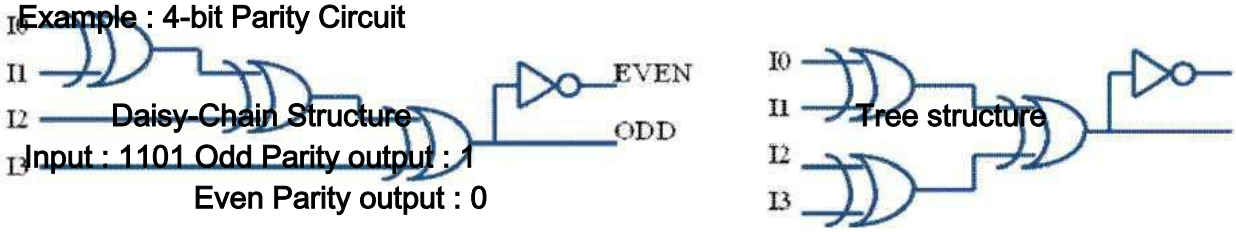


XOR GATE

Odd Parity Circuit : The output is 1 if odd number of inputs are 1

Even Parity Circuit : The output is 1 if even number of inputs are 1

Example : 4-bit Parity Circuit



Daisy-Chain Structure

Input : 1101 Odd Parity output : 1

Even Parity output : 0

Tree structure

Four XOR gates are provided in a single 14-pin SSI IC, the *74x86* shown in Figure. New SSI logic families do not offer XNOR gates, although they are readily available in FPGA and ASIC libraries and as primitives in HDLs.

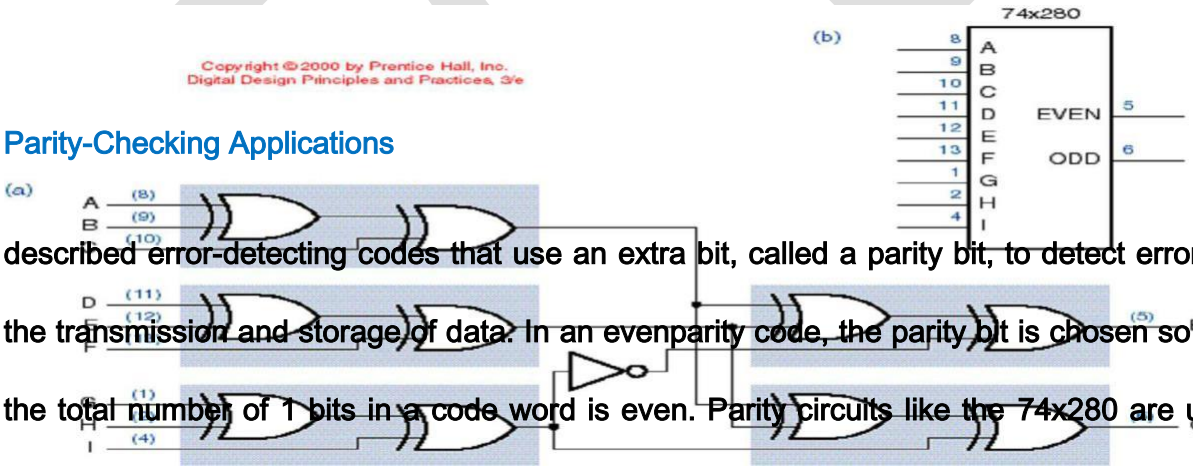
Parity Circuits

N XOR gates may be cascaded to form a circuit with *n* 1 inputs and a single output. This is called an *odd-parity circuit*, because its output is 1 if an odd number of its inputs are 1. If the output of either circuit is inverted, we get an *even-parity circuit*, whose output is 1 if an even number of its inputs are 1.

The 74x280 9-Bit Parity Generator

Rather than build a multibit parity circuit with discrete XOR gates, it is more economical to put all of the XORs in a single MSI package with just the primary inputs and outputs

available at the external pins. The *74x280* 9-bit parity generator, shown in Figure is such a device. It has nine inputs and two outputs that indicate whether an even or odd number of inputs are 1.



Parity-Checking Applications

described error-detecting codes that use an extra bit, called a parity bit, to detect errors in the transmission and storage of data. In an evenparity code, the parity bit is chosen so that the total number of 1 bits in a code word is even. Parity circuits like the 74x280 are used both to generate the correct value of the parity bit when a code word is stored or transmitted, and to check the parity bit when a code word is retrieved or received. Figure shows how a parity circuit might be used to detect errors in the memory of a microprocessor system. The memory stores 8-bit bytes, plus a parity bit for each byte. The microprocessor uses a bidirectional bus D[0:7] to transfer data to and from the memory. Two control lines, RD and WR, are used to indicate whether a read or write operation is desired, and an ERROR signal is asserted to indicate parity errors during read operations. Complete details

of the memory chips, such as addressing inputs, are not shown.

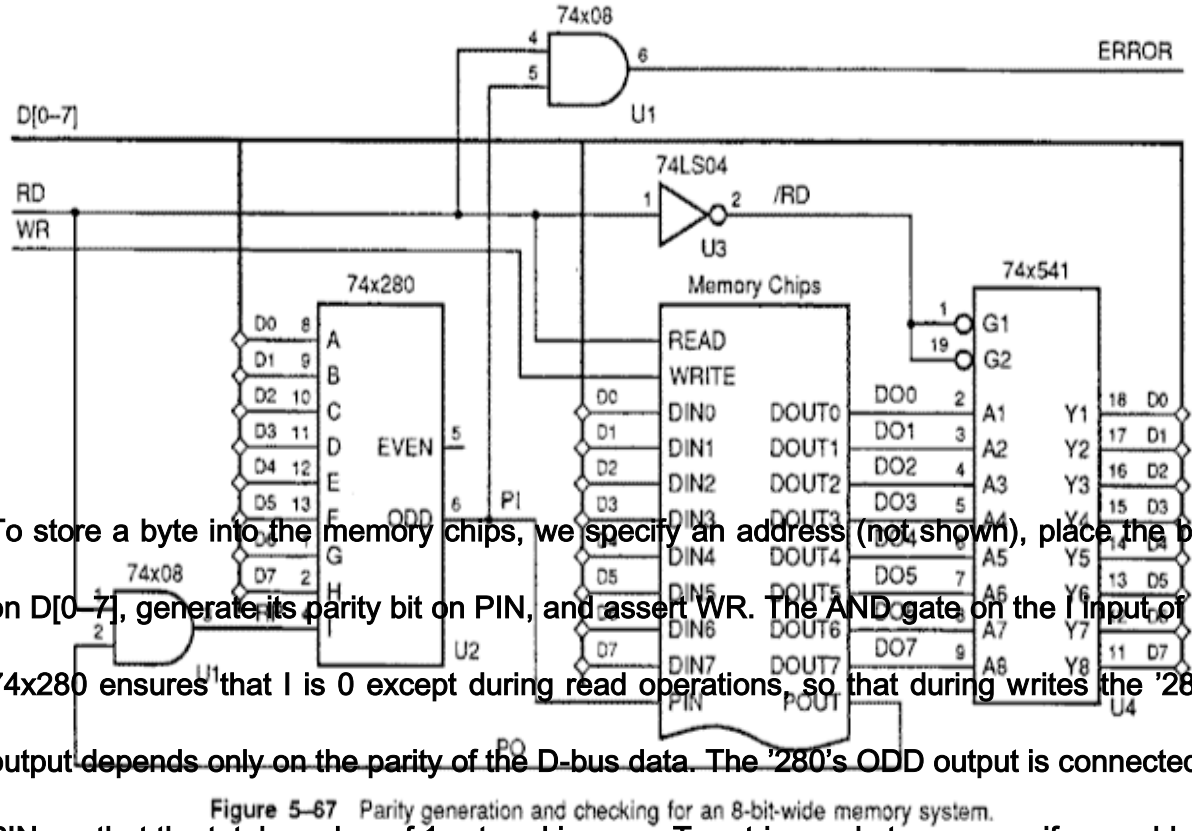


Figure 5–67 Parity generation and checking for an 8-bit-wide memory system.

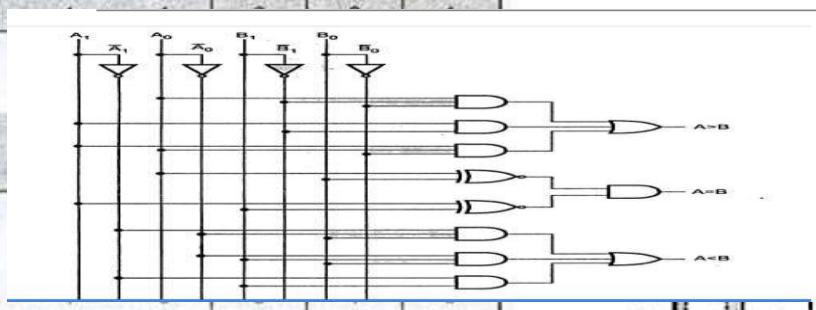
To store a byte into the memory chips, we specify an address (not shown), place the byte on D[0–7], generate its parity bit on PIN, and assert WR. The AND gate on the I input of the 74x280 ensures that I is 0 except during read operations, so that during writes the '280's output depends only on the parity of the D-bus data. The '280's ODD output is connected to PIN, so that the total number of 1s stored is even. To retrieve a byte, we specify an address (not shown) and assert RD; the byte value appears on DOUT[0–7] and its parity appears on POUT. A 74x541 drives the byte onto the D bus, and the '280 checks its parity. If the parity of the 9-bit word DOUT[0–7],POUT is odd during a read, the ERROR signal is asserted. Parity circuits are also used with error-correcting codes such as the Hamming codes. A 7-bitword, possibly containing an error, is presented on DU[1–7]. Three 74x280s compute the parity of the three bit-groups defined by the parity-check matrix. The outputs of the '280s form the syndrome, which is the number of the erroneous input bit, if any. A 74x138 is used to decode the syndrome. If the syndrome is zero, the NOERROR_L signal is asserted (this signal also could be named ERROR). Otherwise, the erroneous 74x280 ordingly.

Comparators

Comparing two binary words for equality is a commonly used operation in computer systems and device interfaces. we showed a system structure in which devices are enabled

by comparing a “device select” word with a predetermined “device ID.” A circuit that compares two binary words and indicates whether they are equal is called a *comparator*. Some comparators interpret their input words as signed or unsigned numbers and also indicate an arithmetic relationship (greater or less than) between the words. These devices are often called *magnitude comparators*.

Inputs				Outputs		
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	0	0	1
0	0	1	0	0	0	1
0	0					
0	1					
0	1					
0	1					
0	1					
1	0					
1	0					
1	0					
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0



	00	01	11	10
11	1	1	0	1
10	1	1	0	0

$A>B = A_0\overline{B}_1\overline{B}_0 + A_1\overline{B}_1 + A_1A_0\overline{B}_0$

A = B

A_1A_0 \ B_1B_0	00	01	11	10

A < B

A_1A_0 \ B_1B_0	00	01	11	10

Inputs				Outputs		
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	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	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	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

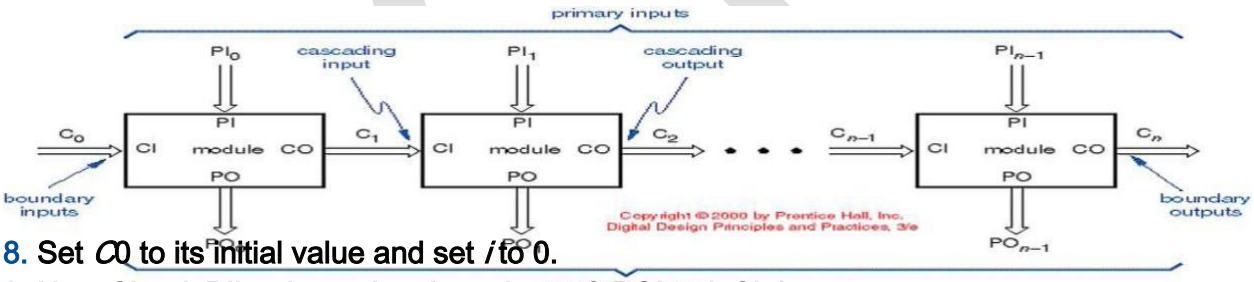
A > B

A_1A_0 \ B_1B_0	00	01	11	10
00	0	0	0	0
01	1	0	0	0
11	1	1	0	1
10	1	1	0	0

$A>B = A_0\overline{B}_1\overline{B}_0 + A_1\overline{B}_1 + A_1A_0\overline{B}_0$

Iterative Circuits

An *iterative circuit* is a special type of combinational circuit, with the structure shown in Figure The circuit contains *n* identical modules, each of which has both *primary inputs and outputs* and *cascading inputs and outputs*. The leftmost cascading inputs are called fixed logic values in most iterative circuits. The *boundary outputs* and usually provide important *boundary inputs* and are connected to rightmost cascading outputs are called information. Iterative circuits are well suited to problems that can be solved by a simple iterative algorithm:



8. Set *C*0 to its initial value and set *i* to 0.
9. Use *Ci* and *Pli* to determine the values of *POi* and *Ci*+1.
10. Increment *i*.
11. If *i* < *n*, go to step 2.

AN ITERATIVE COMPARATOR

The *n*-bit comparators in the preceding subsection might be called *parallel comparators* because they look at each pair of input bits simultaneously and deliver the 1-bit comparison results in parallel to an *n*-input OR or AND function. It is also possible to design an "iterative comparator" that looks at its bits one at a time using a small, fixed amount of logic per bit.

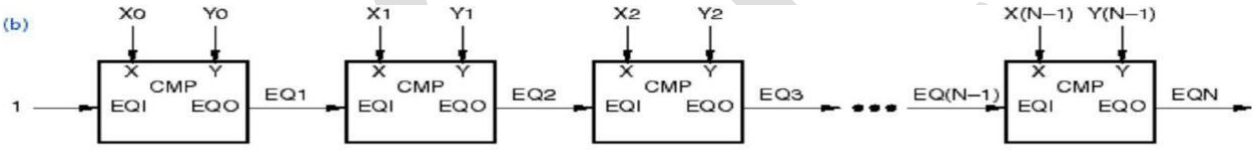
An Iterative Comparator Circuit

Two *n*-bit values *X* and *Y* can be compared one bit at a time using a single bit EQ*i* at each step to keep track of whether all of the bit-pairs have been equal so far:

-- Set EQ0 to 1 and set *i* to 0.
-- If EQ*i* is 1 and X*i* and Y*i* are equal, set EQ*i* 1 to 1. Else set EQ*i* 1 to 0.

-- Increment *i*.
-- If *i* < *n*, go to step 2.

Figure shows a corresponding iterative circuit. Note that this circuit has no primary outputs; the boundary output is all that interests us. Other iterative circuits, such as the ripple adder have primary outputs of interest. Given a choice between the iterative comparator circuit in this subsection and one of the parallel comparators shown previously, you would probably prefer the parallel comparator. The iterative comparator saves little if any cost, and it's very slow because the cascading signals need time to "ripple" from the leftmost to the rightmost module. Iterative circuits that process more than one bit



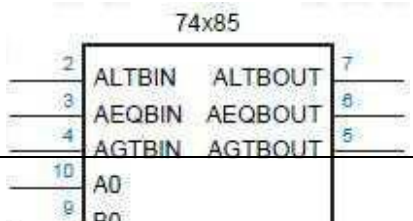
Standard MSI Comparators

Comparator applications are common enough that several MSI comparators have been developed commercially. The *74x85* is a 4-bit comparator with the logic symbol shown in Figure. It provides a greater-than output (AGTBOUT) and a less-than output (ALTBOUT) as well as an equal output (AEQBOUT). The '85 also has *cascading inputs* (AGTBIN, ALTBIN, AEQBIN) for combining multiple '85s to create comparators for more than four bits. Both the cascading inputs and the outputs are arranged in a 1-out-of-3 code, since in normal operation exactly one input and one output should be asserted. The cascading inputs are defined so the outputs of an '85 that compares less-significant bits are connected to the inputs of an '85 that compares more.

4 bit comparator

3 outputs : A=B, A<B, A>B

3 Cascading inputs



Functional Output equations :

(A>B OUT)= (A>B)+(A=B).(A>B IN)

(A<B OUT)= (A<B)+(A=B).(A<B IN)

(A=B OUT)= (A=B).(A=B IN)

Cascading inputs initial values:

(A=B IN) =1

(A>B IN) =0

(A<B IN) =0

significant bits, as shown in Figure for a 12-bit comparator. This is an iterative circuit according to the definition Each ’85 develops its cascading outputs roughly according to the following pseudo-logic equations: The parenthesized subexpressions above are not normal logic expressions, but indicate an arithmetic comparison that occurs between the A3–A0 and B3–B0 inputs. In other words, AGTBOUT is asserted if A > B or if A B and AGTBIN is asserted (if the higher-order bits are equal, we have to look at the lower-order bits for the answer).

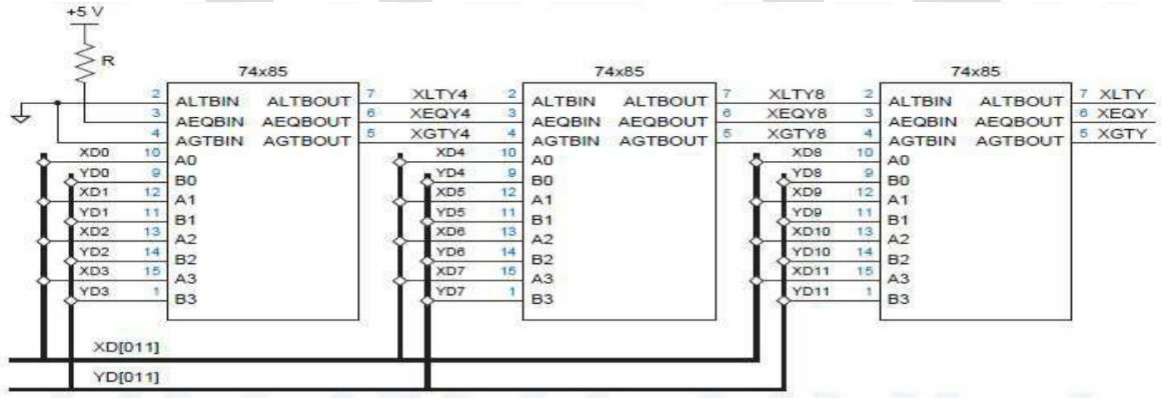
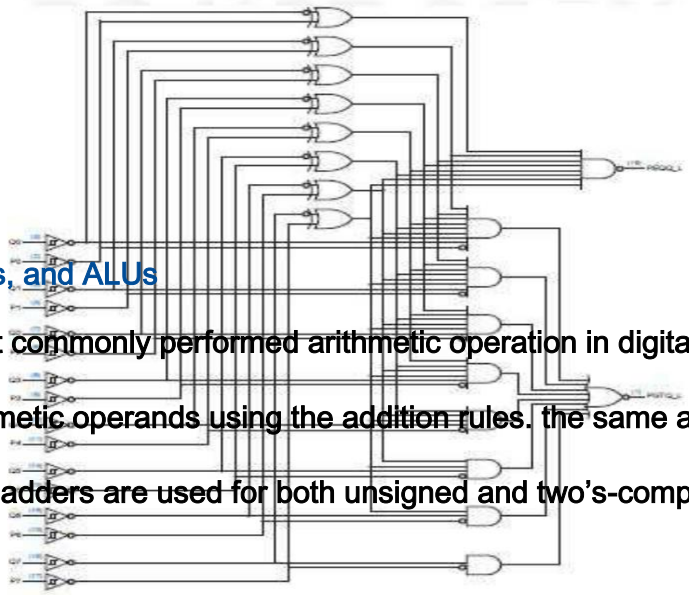


Figure A 12-bit comparator using 74x85s.



Adders, Subtractors, and ALUs

Addition is the most commonly performed arithmetic operation in digital systems. An *adder* combines two arithmetic operands using the addition rules. the same addition rules and therefore the same adders are used for both unsigned and two’s-complement numbers. An

adder can perform subtraction as the addition of the minuend and the complemented (negated) subtrahend, but you can also build *subtractor Half Adder: adds two 1-bit operands*

Half Adders and Full Adders

The simplest adder, called a *half adder*, adds two 1-bit operands X and Y, producing a 2-bit sum. The sum can range from 0 to 2, which requires two bits to express. The low-order bit of the sum may be named HS (half sum), and the high-order bit may be named CO (carry out). We can write the following equations for HS and CO:

Inputs		Outputs	
A	B	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Table 4.32 Truth table for half-adder

Fig. 4.99 Block schematic of half-adder

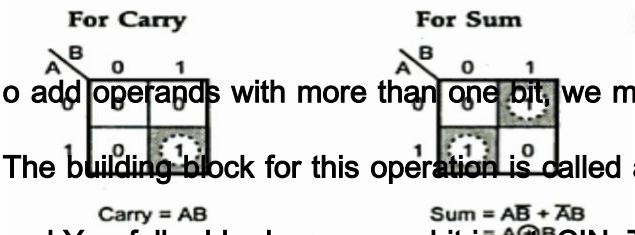
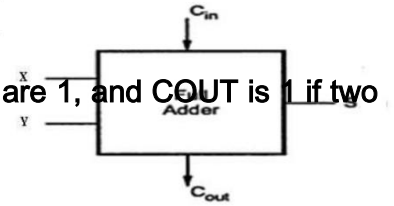


Fig. 4.101 Logic diagram for half-adder

o add operands with more than one bit, we must provide for carries between bit positions. The building block for this operation is called a *full adder*. Besides the addend-bit inputs X and Y, a full adder has a carry-bit input, CIN. The sum of the three inputs can range from 0 to 3, which can still be expressed with just two output bits, S and COUT, having the following equations: Here, S is 1 if an odd number of the inputs are 1, and COUT is 1 if two or more of the inputs are 1.

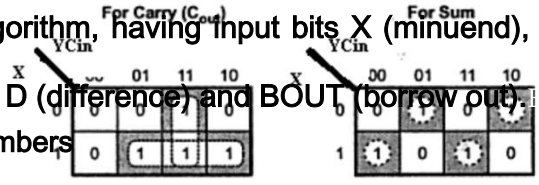
X	Y	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



Block schematic of full adder

Subtractors

A binary subtraction operation analogous to binary addition . A *full subtractor* handles one bit of the binary subtraction algorithm, having input bits X (minuend), Y (subtrahend), and BIN (borrow in), and output bits D (difference) and BOUT (borrow out).



X,Y are n-bit unsigned binary numbers

Addition : S = X + Y

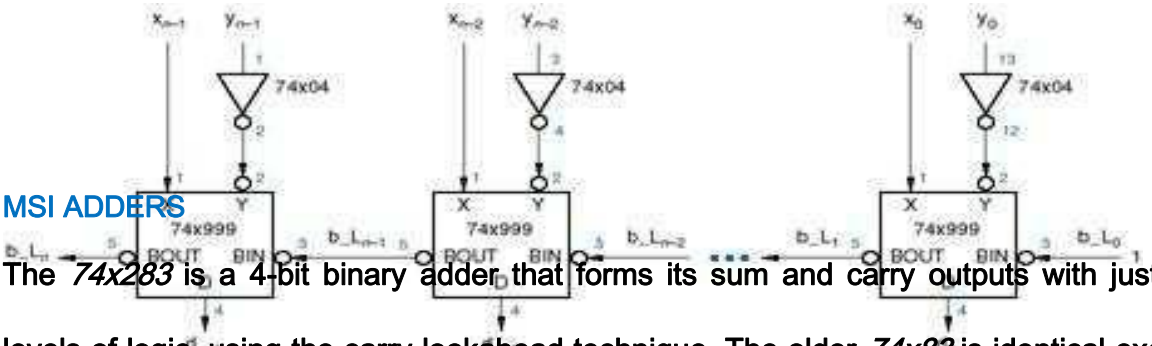
Subtraction : D = X - Y = X + (-Y) =

G: X+ (Two's Complement of Y)

H: X+ (One's Complement of Y) + 1

I: X+ Y'+ 1

Using Adder as a Subtractor



MSI ADDERS

The *74x283* is a 4-bit binary adder that forms its sum and carry outputs with just a few levels of logic, using the carry lookahead technique. The older *74x83* is identical except for its pinout, which has nonstandard locations for power and ground. The logic diagram for the '283, has just a few differences from the general carry-lookahead design that we described in the preceding subsection. First of all, its addends are named A and B instead of X and Y; no big deal. Second, it produces active-low versions of the carry-generate (g*i*) and carrypropagate (p*i*) signals, since inverting gates are generally faster than noninverting ones. Third, it takes advantage of the fact that we can algebraically manipulate the half-sum equation as follows: Thus, an AND gate with an inverted input

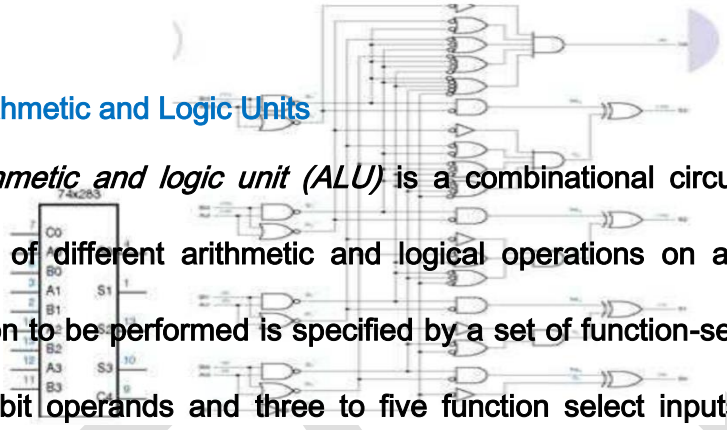
can be used instead of an XOR gate to create each half-sum bit. Finally, the '283 creates the carry signals using an INVERT-OR-AND structure (the DeMorgan equivalent of an AND-OR-INVERT), which has about the same delay as a single CMOS or TTL inverting gate. This requires some explaining, since the carry equations that we derived in the preceding subsection are used in a slightly modified form. In particular, the c*i* 1 equation uses the term p*i* g*i* instead of g*i*. This has no effect on the output, since pi is always 1 when g*i* is 1. However, it allows the equation to be factored as follows: This leads to the following carry equations, which are used by the circuit :

$$
\begin{aligned}
hs_i &= x_i \oplus y_i \\
&= x_i \cdot y_i' + x_i' \cdot y_i \\
&= x_i \cdot y_i' + x_i \cdot x_i' + x_i' \cdot y_i + y_i \cdot y_i' \\
&= (x_i + y_i) \cdot (x_i' + y_i') \\
&= (x_i + y_i) \cdot (x_i \cdot y_i)' \\
&= p_i \cdot g_i'
\end{aligned}
$$

The propagation delay from the C0 input to the C4 output of the '283 is very short, about the

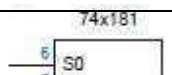
same as two inverting gates. As a result, fairly fast *groupripple adders* with more than four bits can be made simply by cascading the carry outputs and inputs of '283s, as shown in Figure for a 16-bit adder. The total propagation delay from C0 to C16 in this circuit is about the same as that of eight inverting gates.

$$
\begin{aligned}
c_1 &= p_0 \cdot (g_0 + c_0) \\
c_2 &= p_1 \cdot (g_1 + c_1) \\
&= p_1 \cdot (g_1 + p_0 \cdot (g_0 + c_0)) \\
&= p_1 \cdot (g_1 + p_0) \cdot (g_1 + g_0 + c_0)
\end{aligned}
$$



MSI Arithmetic and Logic Units

An *arithmetic and logic unit (ALU)* is a combinational circuit that can perform any of a number of different arithmetic and logical operations on a pair of *b*-bit operands. The operation to be performed is specified by a set of function-select inputs. Typical MSI ALUs have 4-bit operands and three to five function select inputs, allowing up to 32 different functions to be performed. Figure is a logic symbol for the *74x181* 4-bit ALU. The operation performed by the '181 is selected by the M and S3–S0 inputs, as detailed in Table. Note that the identifiers A, B, and F in the table refer to the 4-bit words A3–A0, B3– B0, and F3–F0; The 181's M input selects between arithmetic and logical operations. When M =1, logical operations are selected, and each output Fi is a function only of the corresponding data inputs, Ai and Bi. No carries propagate between stages, and the CIN input is ignored. The S3–S0 inputs select a particular logical operation; any of the 16 different combinational logic functions on two variables may be selected. When M = 0, arithmetic operations are selected, carries propagate between the stages, and CIN is used as a carry input to the least significant stage. For operations larger than four bits, multiple '181 ALUs may be cascaded like the group-ripple adder in the Figure, with the carry-out (COUT) of each ALU connected to the carry-in (CIN) of the next most significant stage. The same function-select signals (M, S3–S0) are applied to all the '181s in the cascade.



To perform two's-complement addition, we use S3–S0 to select the operation "A plus B plus CIN." The CIN input of the least-significant ALU is normally set to 0 during addition operations. To perform two's-complement subtraction, we use S3–S0 to select the operation A minus B minus plus CIN. In this case, the CIN input of the least significant

ALU is normally set to 1, since CIN acts as the complement of the borrow during subtraction. The '181 provides other arithmetic operations, such as "A minus 1 plus CIN," that are useful in some applications (e.g., decrement by 1). It also provides a bunch of weird arithmetic operations, such as "A B plus (A B) plus CIN," that are almost never used in practice, but that "fall out" of the circuit for free. Notice that the operand inputs A3_L–A0_L and B3_L– B0_L and the function outputs F3_L–F0_L of the '181 are active low. The '181 can also be used with active-high operand inputs and function outputs. In this case, a different version of the function table must be constructed. When M 1, logical operations are still performed, but for a given input combination on S3–S0, the function obtained is precisely the dual of the one listed in Table . When M 0, arithmetic operations are performed, but the function table is once again different.

Inputs				Function	
S3	S2	S1	S0	*M = 0 (arithmetic)*	*M = 1 (logic)*
0	0	0	0	F = A minus 1 plus CIN	F = A′
0	0	0	1	F = A · B minus 1 plus CIN	F = A′ + B′
0	0	1	0	F = A · B′ minus 1 plus CIN	F = A′ + B
0	0	1	1	F = 1111 plus CIN	F = 1111
0	1	0	0	F = A plus (A + B′) plus CIN	F = A′ · B′
0	1	0	1	F = A · B plus (A + B′) plus CIN	F = B′
0	1	1	0	F = A minus B minus 1 plus CIN	F = A ⊕ B′
0	1	1	1	F = A + B′ plus CIN	F = A + B′
1	0	0	0	F = A plus (A + B) plus CIN	F = A′ · B
1	0	0	1	F = A plus B plus CIN	F = A ⊕ B
1	0	1	0	F = A · B′ plus (A + B) plus CIN	F = B′
1	0	1	1	F = A + B plus CIN	F = A + B
1	1	0	0	F = A plus A plus CIN	F = 0000
1	1	0	1	F = A · B plus A plus CIN	F = A · B′

Combinational Multipliers

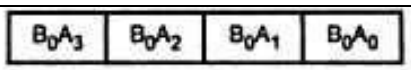
Combinational Multiplier Structures

Multiplier has an algorithm that uses *n* shifts and adds to multiply *n*-bit binary numbers.

Although the shift-and-add algorithm emulates the way that we do paper-and-pencil multiplication of decimal numbers, there is nothing inherently “sequential” or “time dependent” about multiplication. That is, given two *n*-bit input words *X* and *Y*, it is possible to write a truth table that expresses the 2*n*-bit product *P X Y* as a *combinational* function of *X* and *Y*. A *combinational multiplier* is a logic circuit with such a truth table. Most approaches to combinational multiplication are based on the paperand- pencil shift-andadd algorithm. Figure 5-96 illustrates the basic idea for an 8x8 multiplier for two

unsigned integers, multiplicand *X* = *x*7*x*6*x*5*x*4*x*3*x*2*x*1*x*0 and multiplier *Y* = *y*7*y*6*y*5*y*4*y*3*y*2*y*1*y*0. We call each row a *product component*, a shifted multiplicand that is multiplied by 0 or 1 depending on the corresponding multiplier bit. Each small box represents one product-component bit *yixj*, the logical AND of multiplier bit *yi* and multiplicand bit *xj*. The product *P* = *p*15*p*14 . ..*p*2*p*1*p*0 has 16 bits and is obtained by adding together all the product components. Figure shows one way to add up the product components. Here, the product-component bits have been spread out to make space, and each “+” box is a full adder equivalent to Figure 5- 85(c) on page 391. The carries in each row of full adders are connected to make an 8-bit ripple adder. Thus, the first ripple adder ombines the first two product components to product the first partial product. Subsequent adders combine each partial product with the next product component.

Sequential multipliers use a single adder and a register to accumulate the partial products. The partial-product register is initialized to the first product component, and for an *n n*-bit multiplication, *n* 1 steps are taken and the adder is used *n* 1 times, once for each of the remaining *n* 1 product components to be added to the partial-product register. Some sequential multipliers use a trick called *carry-save addition* to speed up multiplication. The idea is to break the carry chain of the ripple adder to shorten the delay of each addition. This is done by applying the carry output from bit *i* during step *j* to the carry input for bit *i*+1 during the *next* step, *j*+1. After the last product component is added, one more step is needed in which the carries are hooked up in the usual way and allowed to ripple from the least to the most significant bit.



4X4 combinational multiplier with carry save

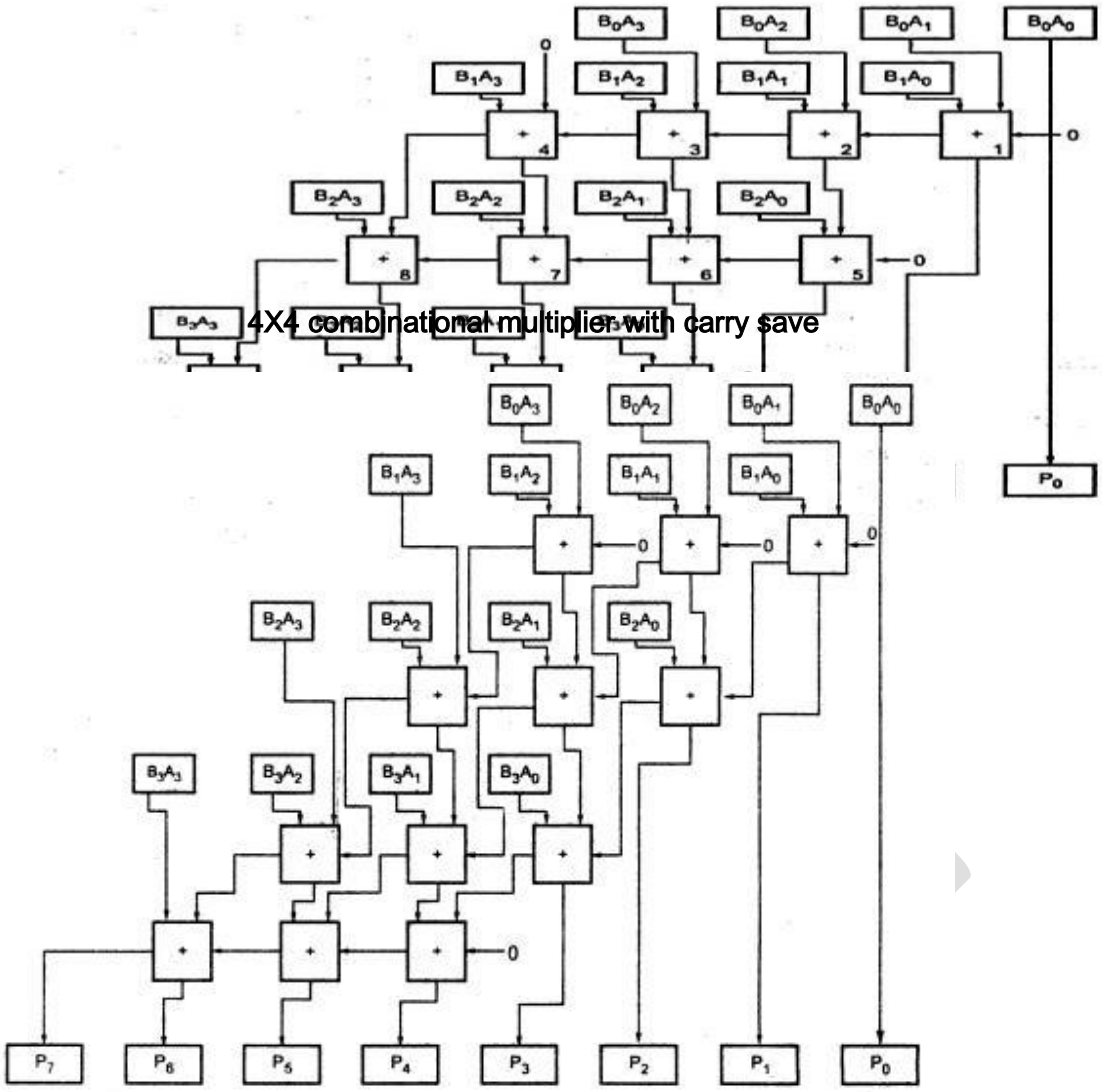


Fig. 4 × 4 combination multiplier with carry save addition

Code Converters

Binary to BCD converter

Binary code				BCD code				
D	C	B	A	B_4	B_3	B_2	B_1	B_0
0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	1
0	0	1	0	0	0	0	1	0
0	0	1	1	0	0	0	1	1
0	1	0	0	0	0	1	0	0
0	1	0	1	0	0	1	0	1
0	1	1	0	0	0	1	1	0
0	1	1	1	0	0	1	1	1
1	0	0	0	0	1	0	0	0
1	0	0	1	0	1	0	0	1
1	0	1	0	1	0	0	0	0
1	0	1	1	1	0	0	0	1
1	1	0	0	1	0	0	1	0
1	1	0	1	1	0	0	1	1
1	1	1	0	1	0	1	0	0
1	1	1	1	1	0	1	0	1

K-map simplification

For B_0

DC \ BA	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	0	1	1	0
10	0	1	1	0

$B_0 = A$

For B_1

DC \ BA	00	01	11	10
00	0	0	1	1
01	0	0	1	1
11	1	1	0	0
10	0	0	0	0

$B_1 = DC\overline{B} + \overline{D}B$

For B_2

DC \ BA	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	0	0	1	1
10	0	0	0	0

$B_2 = \overline{D}C + CB$

For B_3

DC \ BA	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	0	0	0
10	1	1	0	0

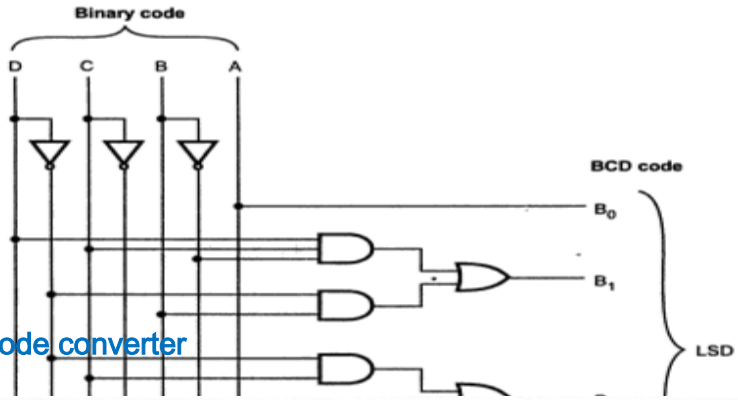
$B_3 = D\overline{C}\,\overline{B}$

For B_4

DC \ BA	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	0	0	1	1

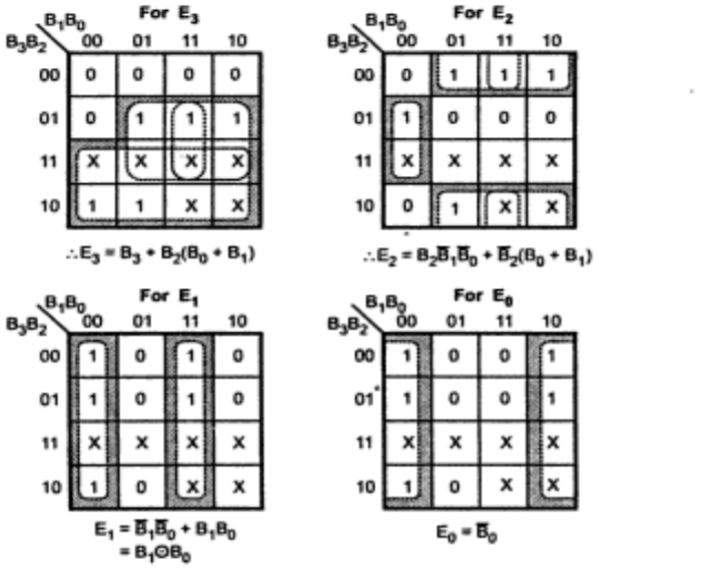
$B_4 = DC + DB$

BCD to Excess-3 code converter

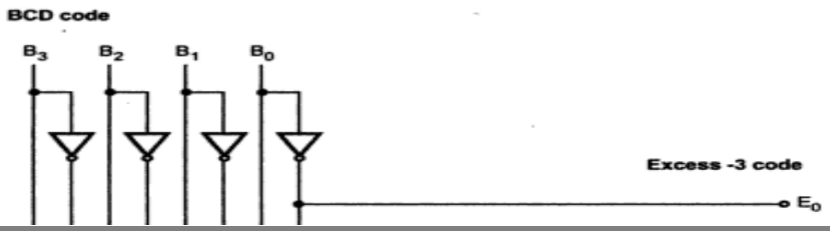


Decimal	B_3	B_2	B_1	B_0	E_3	E_2	E_1	E_0
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

K-map simplification



Logic diagram



VHDL structural program for the decoder

```
library IEEE;
use IEEE.std_logic_1164.all;

entity V2to4dec is
  port (I0, I1, EN: in STD_LOGIC;
        Y0, Y1, Y2, Y3: out STD_LOGIC );
end V2to4dec;

architecture V2to4dec_s of V2to4dec is
  signal NOTI0, NOTI1: STD_LOGIC;
  component inv port (I: in STD_LOGIC; O: out STD_LOGIC ); end component;
  component and3 port (I0, I1, I2: in STD_LOGIC; O: out STD_LOGIC ); end component;
begin
  U1: inv port map (I0,NOTI0);
  U2: inv port map (I1,NOTI1);
  U3: and3 port map (NOTI0,NOTI1,EN,Y0);
  U4: and3 port map (   I0,NOTI1,EN,Y1);
  U5: and3 port map (NOTI0,   I1,EN,Y2);
  U6: and3 port map (   I0,   I1,EN,Y3);
end V2to4dec_s;
```

Dataflow-style VHDL program for a 74x138-like 3-to-8 binary decoder.

```
library IEEE;
use IEEE.std_logic_1164.all;
```

Behavioral VHDL program for a 74x148-like 8-input priority encoder.

ECE

Design Examples (Using VHDL)

Barrel Shifter:

A *barrel shifter* is a combinational logic circuit with *n* data inputs, *n* data outputs, and a set of control inputs that specify how to shift the data between input and output. A barrel shifter that is part of a microprocessor CPU can typically specify the direction of shift (left or right), the type of shift (circular, arithmetic, or logical), and the amount of shift (typically 0 to *n* – 1 bits, but sometimes 1 to *n* bits).

In this subsection we'll look at the design of a 16-bit barrel shifter that does six different types of shifts, as specified by a 3-bit shift-mode input C[2:0]. A 4-bit shift-amount input S[3:0] specifies the amount of shift. For example, if C specifies a left-circular shift and the input word is ABCDEFGHIJKLMNOP (where each letter represents one bit), and S[3:0] is 0101 (5), then the output word is FGHIJKLMNOPABCDE.

Shift Type	*Name*	*Code*	*Note*
Left rotate	Lrotate	000	Wrap-around
Right rotate	Rrotate	001	Wrap-around
Left logical	Llogical	010	0 into LSB
Right logical	Rlogical	011	0 into MSB
Left arithmetic	Larith	100	0 into LSB
Right arithmetic	Rarith	101	Replicate MSB

VHDL behavioral description of a 6-function barrel shifter:

```
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
entity barrel16 is
port ( DIN: in STD_LOGIC_VECTOR (15 downto 0); -- Data inputs
       Z: in UNSIGNED (3 downto 0); -- Shift amount, 0-15
       Z: in STD_LOGIC_VECTOR (2 downto 0); -- Mode control
       DOUT: out STD_LOGIC_VECTOR (15 downto 0) -- Data bus output
       );
        constant Lrotate: STD_LOGIC_VECTOR := "000"; -- Define the coding of
       constant Rrotate: STD_LOGIC_VECTOR := "001"; -- the different shift modes
       constant Llogical: STD_LOGIC_VECTOR := "010";
       constant Rlogical: STD_LOGIC_VECTOR := "011";
       constant Larith: STD_LOGIC_VECTOR := "100";
       constant Rarith: STD_LOGIC_VECTOR := "101";
```

```
end barrel16;
architecture barrel16_behavioral of barrel16 is
        subtype DATAWORD is STD_LOGIC_VECTOR(15 downto 0);
        function Vrol (D: DATAWORD; S: UNSIGNED) return
        DATAWORD is
        variable N: INTEGER;
        variable TMPD: DATAWORD;
begin
        N := CONV_INTEGER(S); TMPD := D;
        for i in 1 to N loop
        TMPD := TMPD(14 downto 0) & TMPD(15);
        end loop;
        return TMPD;
        end Vrol;
        ...
begin
        process(DIN, S, C)
        begin
        case C is
        when Lrotate => DOUT <= Vrol(DIN,S);
        when Rrotate => DOUT <= Vror(DIN,S);
        when Illogical => DOUT <= Vsll(DIN,S);
        when Rlogical => DOUT <= Vsrl(DIN,S);
        when Larith => DOUT <= Vsla(DIN,S);
        when Rarith => DOUT <= Vsra(DIN,S);
        when others => DOUT <= DIN;
        end case;
        end process;
end barrel16_behavioral;
```

VHDL program for a 16-bit barrel shifter for left circular shifts only:

```
library IEEE;
```

```
use IEEE.std_logic_1164.all;
entity rol16 is
port (
DIN: in STD_LOGIC_VECTOR(15 downto 0); -- Data inputs
G: in STD_LOGIC_VECTOR (3 downto 0); -- Shift amount, 0-15
DOUT: out STD_LOGIC_VECTOR(15 downto 0) -- Data bus output );
end rol16;
architecture rol16_arch of rol16 is
begin
process(DIN, S)
variable X, Y, Z: STD_LOGIC_VECTOR(15 downto 0);
begin
if S(0)='1' then X := DIN(14 downto 0) & DIN(15); else X := DIN; end if;
if S(1)='1' then Y := X(13 downto 0) & X(15 downto 14); else Y := X; end if; if
S(2)='1' then Z := Y(11 downto 0) & Y(15 downto 12); else Z := Y; end if;
if S(3)='1' then DOUT <= Z(7 downto 0) & Z(15 downto 8); else DOUT <= Z; end if;
end process;
end rol16_arch;
```

Comparator:

VHDL program for an 8-bit comparator:

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
entity comp8 is
port ( A, B: in STD_LOGIC_VECTOR (7 downto 0);
       EQ, GT: out STD_LOGIC );
end comp8;
architecture comp8_arch of comp8 is
begin
        EQ <= '1' when A = B else '0';
        GT <= '1' when A > B else '0';
end comp8_arch;
```

Floating-point Encoder:

An unsigned binary integer *B* in the range 0 ≤ B < 211 can be represented by 11 bits in "fixed-point" format, *B* = *b*10*b*9…*b*1*b*0. We can represent numbers in the same range with less precision using only 7 bits in a floating-point notation, *F* = *M*· 2*E*, where *M* is a 4-bit mantissa *m*3*m*2*m*1*m*0 and *E* is a 3-bit exponent *e*2*e*1*e*0. The smallest integer in this format is 0·20 and the largest is (24−1) · 27. Given an 11-bit fixed-point integer *B*, we can convert it to our 7-bit floating-point notation by "picking off" four high-order bits beginning with the most significant 1, for example, The last term in each equation is a truncation error that results from the loss of precision in the conversion. Corresponding to this conversion operation, we can write the specification for a fixed-point to floating-point encoder circuit:

G: A combinational circuit is to convert an 11-bit unsigned binary integer *B* into a 7-bit floating-point number *M,E*, where *M* and *E* have 4 and 3 bits, respectively. The numbers have the relationship *B* = *M*·2*E* + *T*, where *T* is the truncation error, 0 ≤ *T*< 2*E*.

Behavioral VHDL program for fixed-point to floating-point conversion:

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
entity fpenc is
port (
        G: in STD_LOGIC_VECTOR(10 downto 0); -- fixed-point number
        G  out STD_LOGIC_VECTOR(3 downto 0); -- floating-point mantissa
        G: out STD_LOGIC_VECTOR(2 downto 0) -- floating-point exponent
        );
end fpenc;
architecture fpenc_arch of fpenc is
begin
process(B)
        variable BU: UNSIGNED(10 downto 0);
        begin
        BU := UNSIGNED(B);
        if BU < 16 then M <= B( 3 downto 0); E <= "000"; elsif BU
        < 32 then M <= B( 4 downto 1); E <= "001"; elsif BU < 64
```

```
        then M <= B( 5 downto 2); E <= "010"; elsif BU < 128
        then M <= B( 6 downto 3); E <= "011"; elsif BU < 256
        then M <= B( 7 downto 4); E <= "100"; elsif BU < 512
        then M <= B( 8 downto 5); E <= "101"; elsif BU < 1024
        then M <= B( 9 downto 6); E <= "110"; else M <= B(10
        downto 7); E <= "111"; end if;
end process;
end fpenc_arch;
```

Alternative VHDL architecture for fixed-point to floating-point conversion:

```
architecture fpence_arch of fpenc is
begin
process(B)
        begin
        if B(10) = '1' then M <= B(10 downto 7); E <= "111";
        elsif B(9) = '1' then M <= B( 9 downto 6); E <= "110";
        elsif B(8) = '1' then M <= B( 8 downto 5); E <= "101";
        elsif B(7) = '1' then M <= B( 7 downto 4); E <= "100";
        elsif B(6) = '1' then M <= B( 6 downto 3); E <= "011";
        elsif B(5) = '1' then M <= B( 5 downto 2); E <= "010";
        elsif B(4) = '1' then M <= B( 4 downto 1); E <= "001";
        else M <= B( 3 downto 0); E <= "000"; end if;
end process;
end fpence_arch;
```

Behavioral VHDL architecture for fixed-point to floating-point conversion with rounding:

```
architecture fpencr_arch of fpenc is
        function round (BSLICE: STD_LOGIC_VECTOR(4 downto 0))
        return STD_LOGIC_VECTOR is
        variable BSU: UNSIGNED(3 downto 0);
        begin
        if BSLICE(0) = '0' then return BSLICE(4 downto 1);
        else
        BSU := UNSIGNED(BSLICE(4 downto 1)) + 1;
        return STD_LOGIC_VECTOR(BSU);
```

```
        end if;
        end;
        begin
        process(B)
        variable BU: UNSIGNED(10 downto 0);
        begin
        BU := UNSIGNED(B);
        if BU < 16 then M <= B( 3 downto 0); E <= "000";
        elsif BU < 32-1 then M <= round(B( 4 downto 0)); E <= "001";
        elsif BU < 64-2 then M <= round(B( 5 downto 1)); E <= "010";
        elsif BU < 128-4 then M <= round(B( 6 downto 2)); E <= "011";
        elsif BU < 256-8 then M <= round(B( 7 downto 3)); E <= "100";
        elsif BU < 512-16 then M <= round(B( 8 downto 4)); E <= "101";
        elsif BU < 1024-32 then M <= round(B( 9 downto 5)); E <= "110";
        elsif BU < 2048-64 then M <= round(B(10 downto 6)); E <= "111";
        else M <= "1111"; E <= "111";
        end if;
        end process;
end fpencr_arch;
```

Dual-Priority Encoder:

In this example we'll use VHDL to create a behavioral description of a PLD priority encoder that identifies both the highest-priority and the second-highest priority asserted signal among a set of request inputs R(0 to 7), where R(0) has the highest priority. We'll use A(2 downto 0) and AVALID to identify the highest-priority request, asserting AVALID only if a highest-priority request is present. Similarly, we'll use B(2 downto 0) and BVALID to identify the second highest-priority request.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
entity Vprior2 is
port (
        G: in STD_LOGIC_VECTOR (0 to 7);
        A, B: out STD_LOGIC_VECTOR (2 downto 0);
        AVALID, BVALID: buffer STD_LOGIC
        );
end Vprior2;
```

```
architecture Vprior2_arch of Vprior2 is
begin
process(R, AVALID, BVALID)
        begin
        AVALID <= '0'; BVALID <= '0'; A <= "000"; B <= "000";
        for i in 0 to 7 loop
        if R(i) = '1' and AVALID = '0' then
        A <= CONV_STD_LOGIC_VECTOR(i,3); AVALID <= '1';
        elsif R(i) = '1' and BVALID = '0' then
        B <= CONV_STD_LOGIC_VECTOR(i,3); BVALID <= '1';
        end if;
        end loop;
end process;
end Vprior2_arch;
```

Alternative VHDL architecture for a dual-priority encoder:

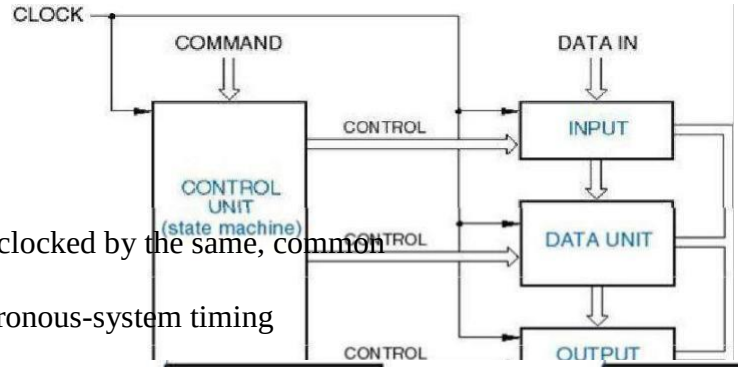
```
architecture Vprior2i_arch of Vprior2 is
begin
process(R, A, AVALID, BVALID)
        begin
        if R(0) = '1' then A <= "000"; AVALID <= '1'; elsif
        R(1) = '1' then A <= "001"; AVALID <= '1'; elsif
        R(2) = '1' then A <= "010"; AVALID <= '1'; elsif
        R(3) = '1' then A <= "011"; AVALID <= '1'; elsif
        R(4) = '1' then A <= "100"; AVALID <= '1'; elsif
        R(5) = '1' then A <= "101"; AVALID <= '1'; elsif
        R(6) = '1' then A <= "110"; AVALID <= '1'; elsif
        R(7) = '1' then A <= "111"; AVALID <= '1'; else A
        <= "000"; AVALID <= '0'; end if;
        if R(1) = '1' and A /= "001" then B <= "001"; BVALID <= '1'; elsif
        R(2) = '1' and A /= "010" then B <= "010"; BVALID <= '1'; elsif
        R(3) = '1' and A /= "011" then B <= "011"; BVALID <= '1'; elsif
        R(4) = '1' and A /= "100" then B <= "100"; BVALID <= '1'; elsif
        R(5) = '1' and A /= "101" then B <= "101"; BVALID <= '1'; elsif
        R(6) = '1' and A /= "110" then B <= "110"; BVALID <= '1'; elsif
```

```
        R(7) = '1' and A /= "111" then B <= "111"; BVALID <= '1'; else B
        <= "000"; BVALID <= '0'; end if;
end process;
end Vprior2i_arch;
```

Sequential Machine Design Practice

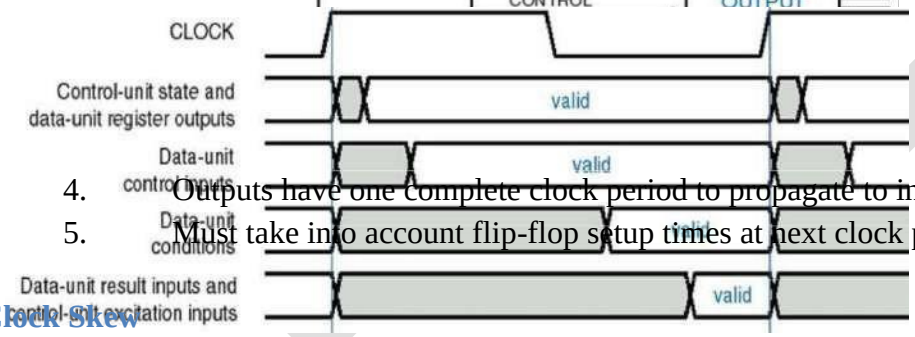
Review of State Machines

Synchronous System Structure



Everything is clocked by the same, common clock

Typical synchronous-system timing



4. Outputs have one complete clock period to propagate to inputs.
5. Must take into account flip-flop setup times at next clock period.

Clock Skew

Clock signal may not reach all flip-flops simultaneously. Output changes of flip-flops receiving ―early‖ clock may reach D inputs of flip-flops with ―late‖ clock too soon.

Reasons for slowness:

wiring delays
capacitance
incorrect design

Clock-skew calculation

*t*ffpd(min) + *t*comb(min) - *t*hold - *t*skew(max) >= 0

First two terms are minimum time after clock edge that a D input changes

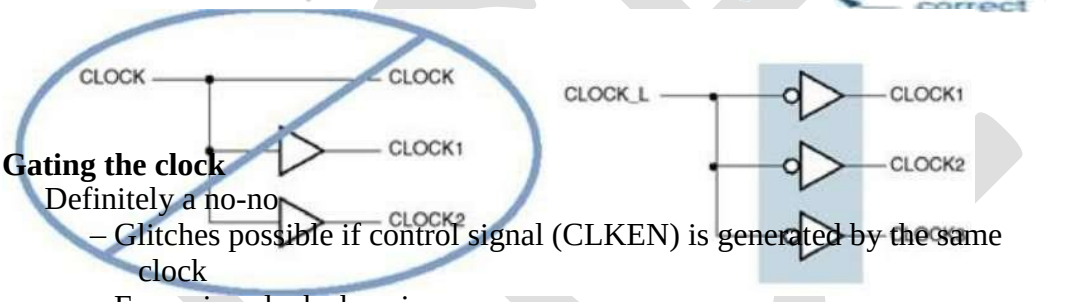
Hold time is earliest time that the input may change

Clock skew subtracts from the available hold-time margin

Compensating for clock skew:

- Longer flip-flop propagation delay
- Explicit combinational delays
- Shorter (even negative) flip-flop hold times

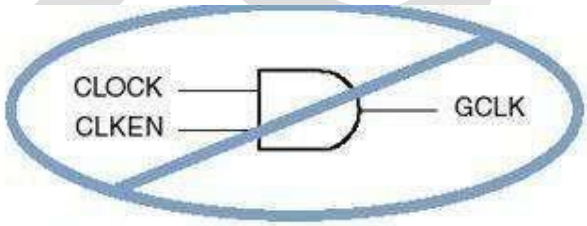
Example of bad clock distribution



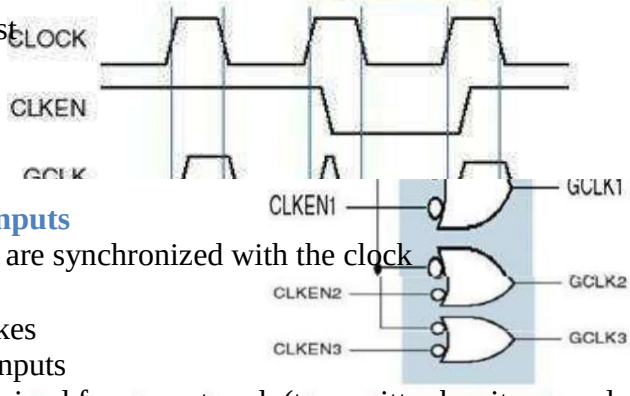
Gating the clock

Definitely a no-no

- Glitches possible if control signal (CLKEN) is generated by the same clock
- Excessive clock skew in any case.



If you really must gate the clock...



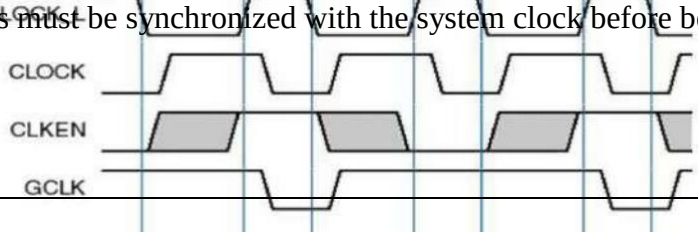
Asynchronous inputs

Not all inputs are synchronized with the clock

Examples:

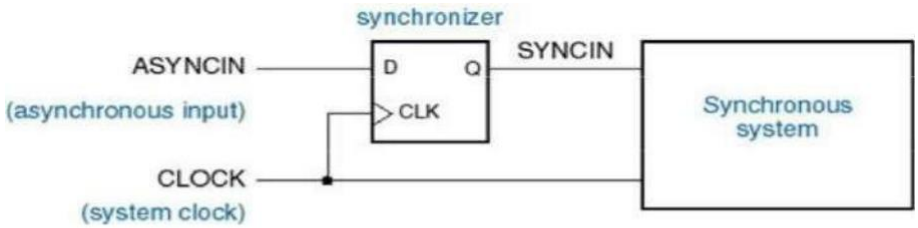
- Keystrokes
- Sensor inputs
- Data received from a network (transmitter has its own clock)

Inputs must be synchronized with the system clock before being applied to

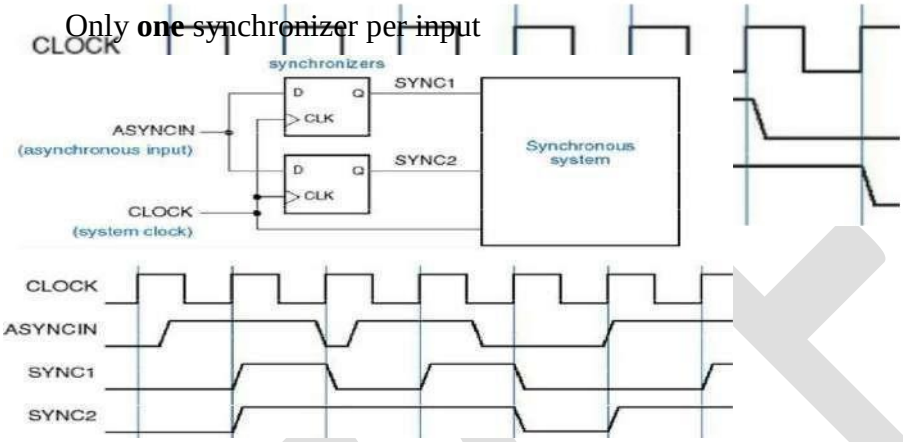


a synchronous system.

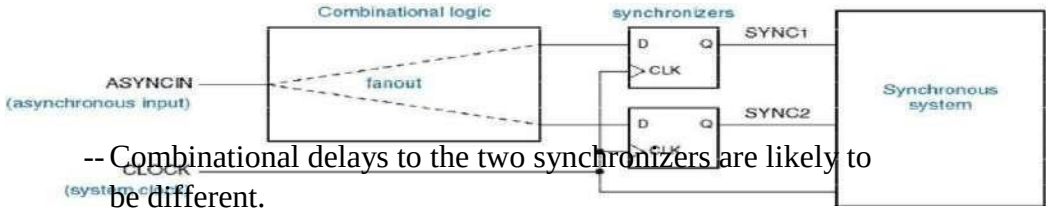
A simple synchronizer



Only **one** synchronizer per input



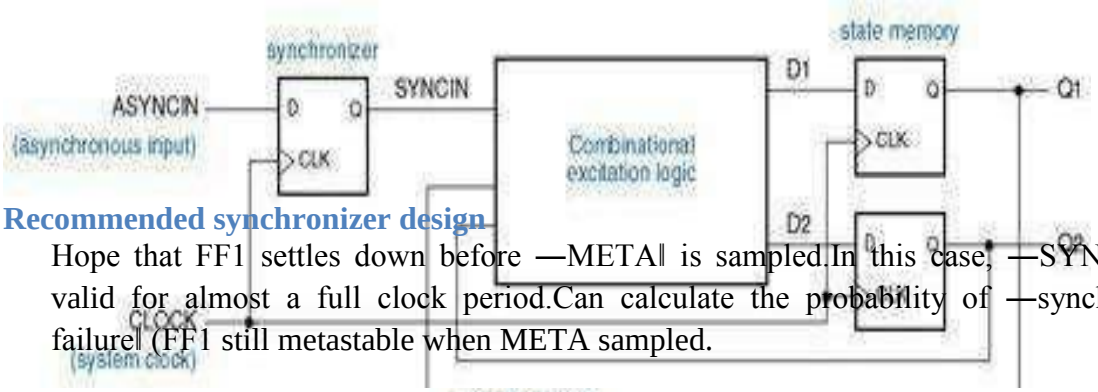
Even worse



-- Combinational delays to the two synchronizers are likely to be different.

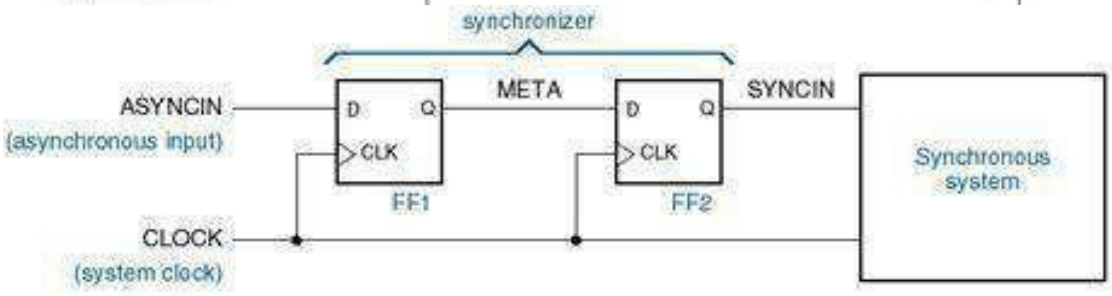
way to do it

One synchronizer per input. Carefully locate the synchronization points in a sys .But still a problem -- the synchronizer output may become metastable when setup and hold time are not met.



Recommended synchronizer design

Hope that FF1 settles down before ―META‖ is sampled.In this case, ―SYNCIN‖ is valid for almost a full clock period.Can calculate the probability of ―synchronizer failure‖ (FF1 still metastable when META sampled.



Impediments to synchronous design

-- Clock skew

– Definition

The difference between arrival times of the clock at different memory

devices Example of clock skew

Influence of clock skew

Reduce the setup and hold time margins. For proper

operation tffpd(min) + tcomb(min) - thold - tskew(max) > 0

tsetup -tclk -tffpd(max) - tcomb(max) - tskew(max) > 0

Reducing clock skew

proper buffering the clock Better clock distribution .

Gating clock

Why not to gate the clock .

An acceptable way.

Asynchronous inputs

Why use the asynchronous inputs?

Problem with asynchronous inputs

Meta-stable

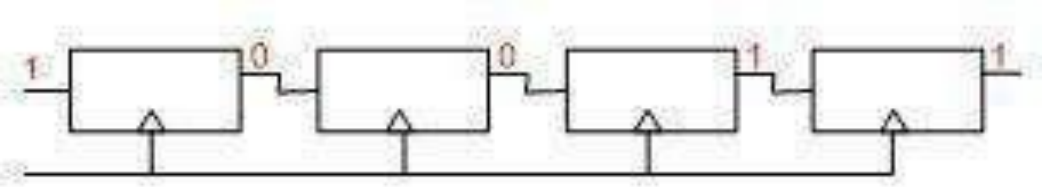
Need synchronizers

A simple one.

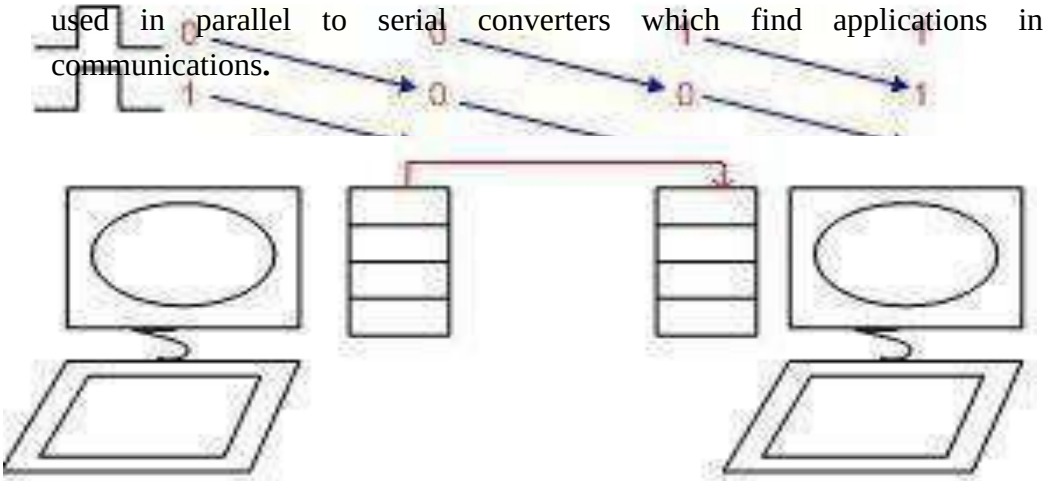
Shift Register

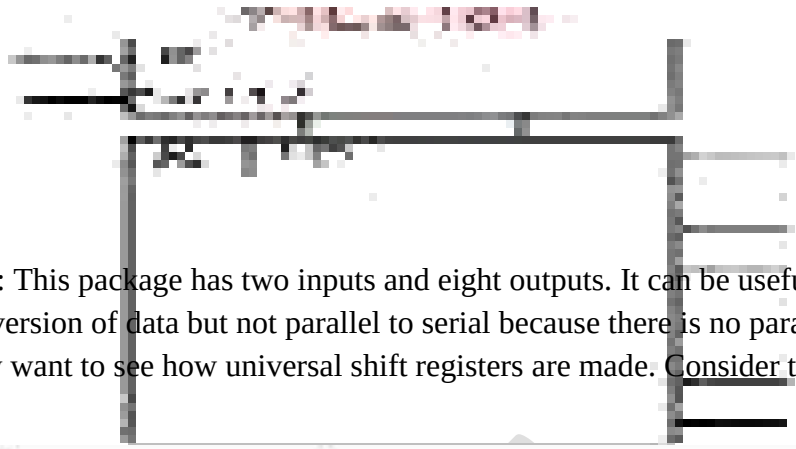
Shift registers are a type of sequential logic circuit, mainly for storage of digital data. They are a group of flip-flops connected in a chain so that the output from one flip-flop becomes the input of the next flip-flop. Most of the registers possess no characteristic internal sequence of states. All the flip-flops are driven by a common clock, and all are set or reset simultaneously. In this chapter, the basic types of shift registers are studied, such as Serial In AA: Serial Out, Serial In - Parallel Out, Parallel In - Serial Out, Parallel In - Parallel Out, and bidirectional shift registers. A special form of counter - the shift register counter, is also introduced.

Let's observe the values of the flip flops in this shift register for the next couple of clock pulse:



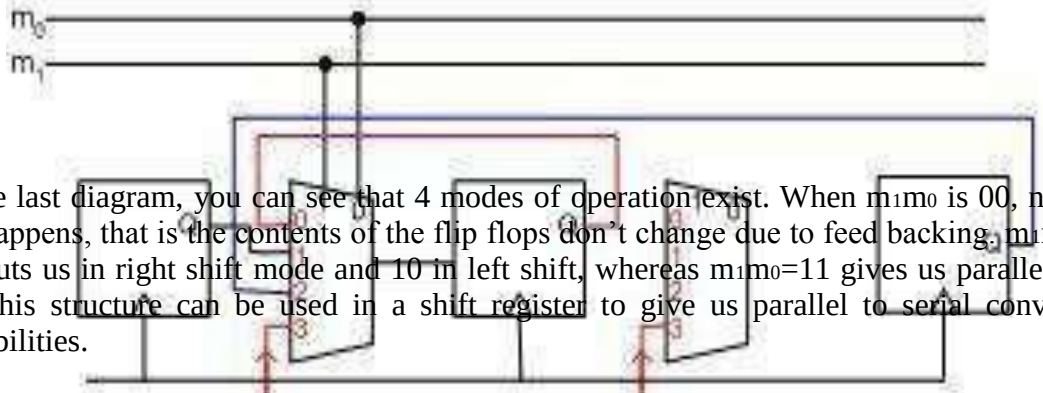
We are actually shifting our data to the right on every clock pulse.Shift registers are widely used in parallel to serial converters which find applications in computer communications.





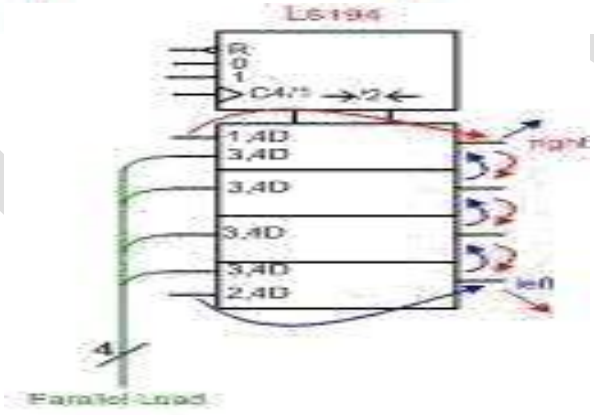
74ls164: This package has two inputs and eight outputs. It can be useful in serial to parallel conversion of data but not parallel to serial because there is no parallel loading.

We now want to see how universal shift registers are made. Consider the following circuit:



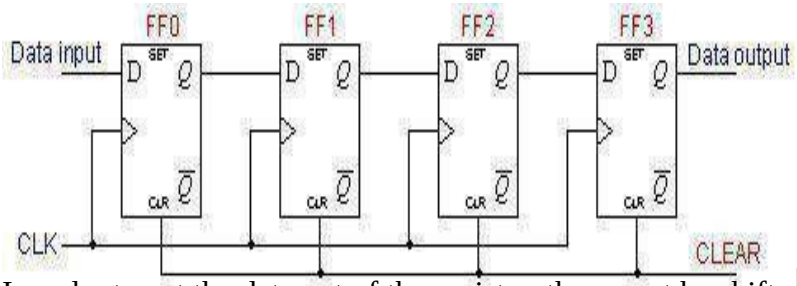
In the last diagram, you can see that 4 modes of operation exist. When m_1m_0 is 00, nothing happens, that is the contents of the flip flops don't change due to feed backing. m_1m_0=01 puts us in right shift mode and 10 in left shift, whereas m_1m_0=11 gives us parallel load. This structure can be used in a shift register to give us parallel to serial conversion abilities.

74ls194: This package give us right and left shifting as well as parallel load in mode 11.

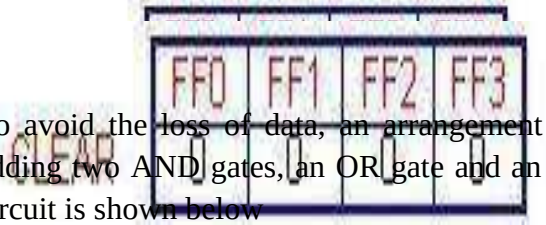


Serial In - Serial Out Shift Registers

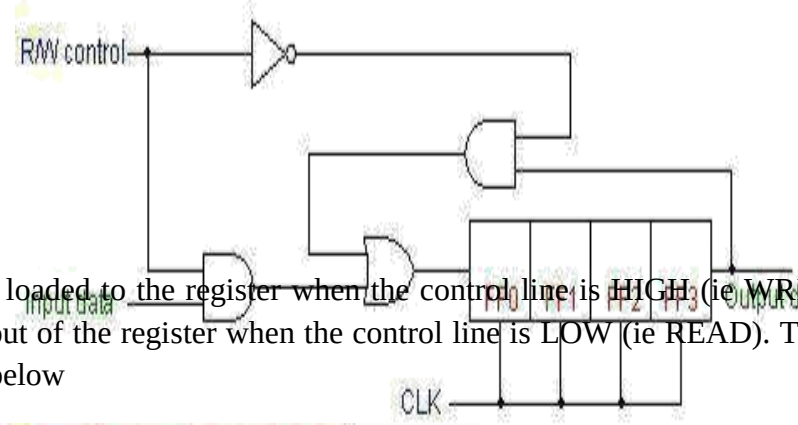
A basic four-bit shift register can be constructed using four D flip-flops, as shown below. The operation of the circuit is as follows. The register is first cleared, forcing all four outputs to zero. The input data is then applied sequentially to the D input of the first flip-flop on the left (FF0). During each clock pulse, one bit is transmitted from left to right. Assume a data word to be 1001. The least significant bit of the data has to be shifted through the register from FF0 to FF3.



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



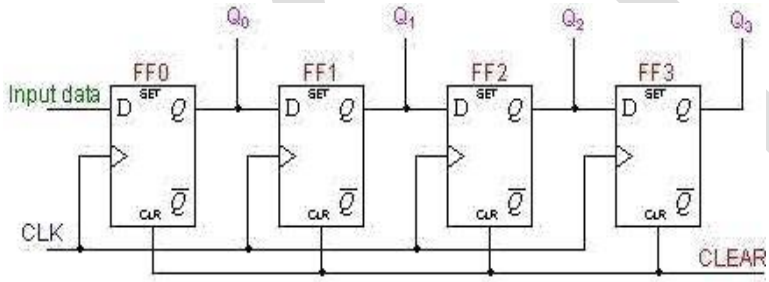
To avoid the loss of data, an arrangement for a non-destructive reading can be done by adding two AND gates, an OR gate and an inverter to the system. The construction of this circuit is shown below



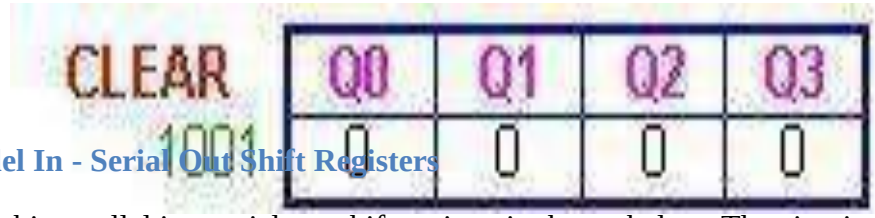
The data is loaded to the register when the control line is HIGH (ie WRITE). The data can be shifted out of the register when the control line is LOW (ie READ). This is shown in the animation below

Serial In - Parallel Out Shift Registers

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

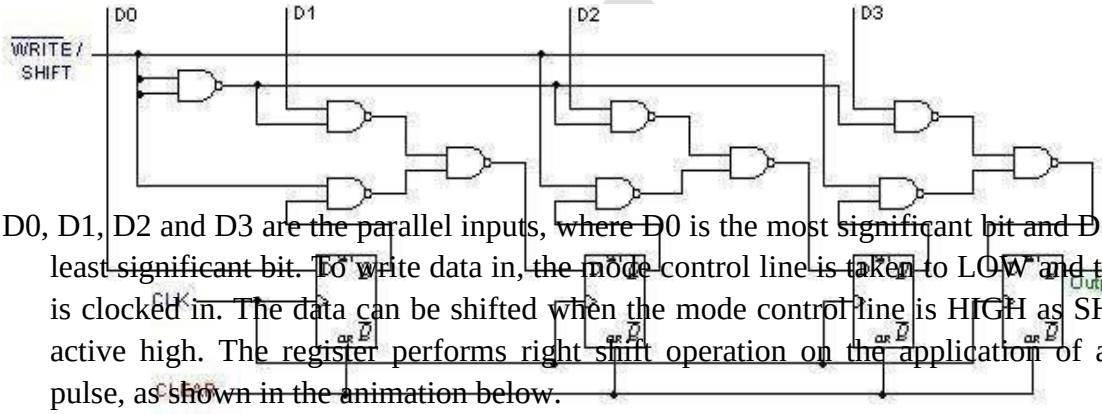


In the animation below, we can see how the four-bit binary number 1001 is shifted to the Q outputs of the register.

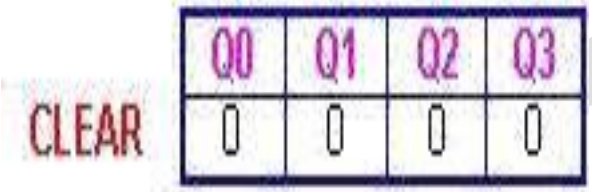


Parallel In - Serial Out Shift Registers

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

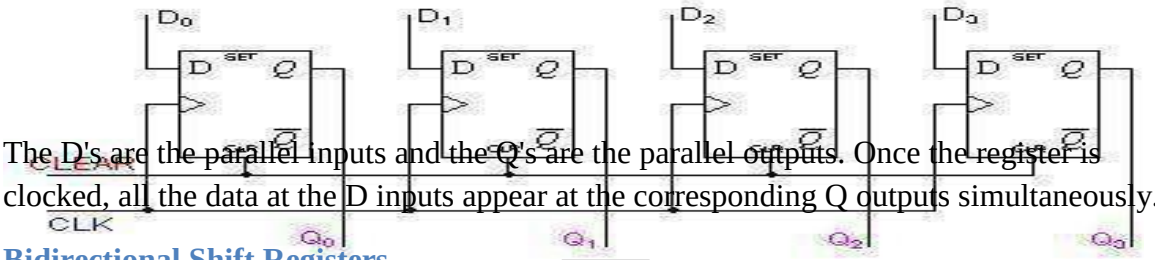


D0, D1, D2 and D3 are the parallel inputs, where D0 is the most significant bit and D3 is the least significant bit. To write data in, the mode control line is taken to LOW and the data is clocked in. The data can be shifted when the mode control line is HIGH as SHIFT is active high. The register performs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



Parallel In - Parallel Out Shift Register

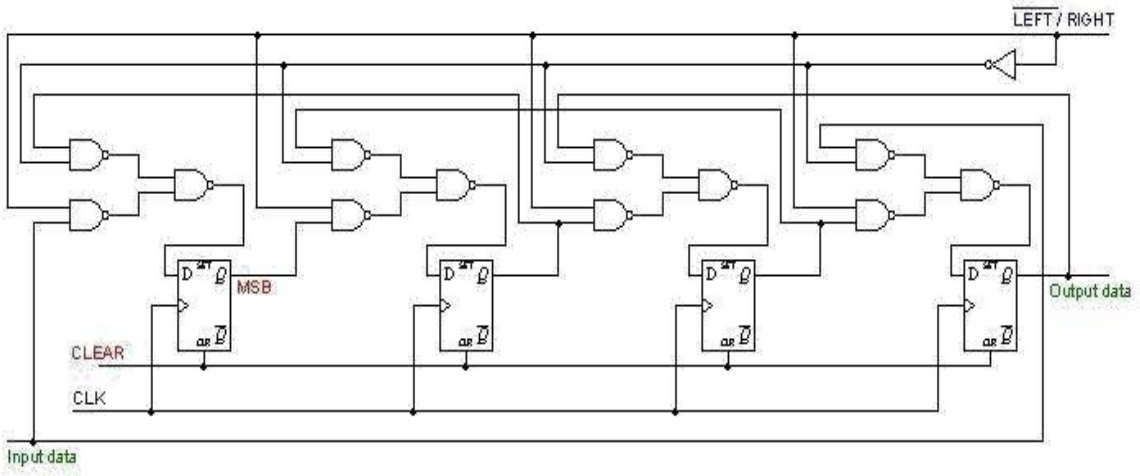
For parallel in - parallel out shift registers, all data bits appear on the parallel outputs immediately following the simultaneous entry of the data bits. The following circuit is a four-bit parallel in - parallel out shift register constructed by D flip-flops.



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.

Bidirectional Shift Registers

The registers discussed so far involved only right shift operations. Each right shift operation has the effect of successively dividing the binary number by two. If the operation is reversed (left shift), this has the effect of multiplying the number by two. With suitable gating arrangement a serial shift register can perform both operations. A *bidirectional*, or *reversible*, shift register is one in which the data can be shift either left or right. A four-bit bidirectional shift register using D flip-flops is shown below.



Here a set of NAND gates are configured as OR gates to select data inputs from the right or left adjacent bistables, as selected by the LEFT/RIGHT control line. The animation below performs right shift four times, then left shift four times. Notice the order of the four output bits are not the same as the order of the original four input bits. They are actually reversed**!**

(SEQUENTIAL LOGIC DESIGN)

Basic Bistable Element

A combinational system is a system whose outputs depends only upon its current inputs.

A sequential system is a system whose output depends on current input *and* past history of inputs.

All systems we have looked at to date have been combinational systems.

Outputs depends on the current inputs and the system's current state.

State‖ embodies all the information about the past needed to predict current output based on current input.

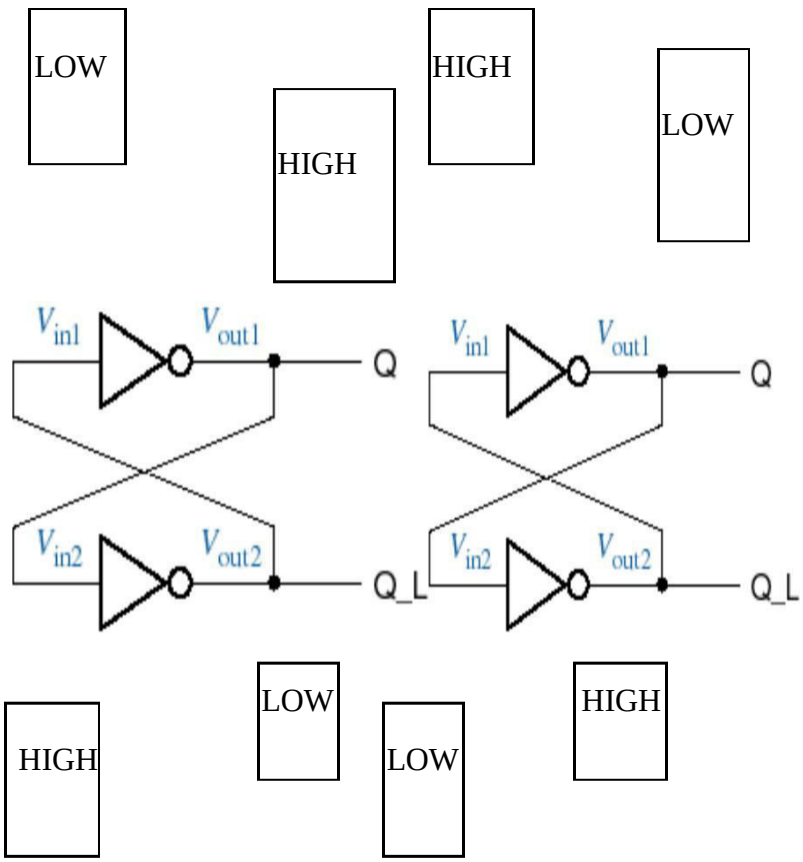
State variables, one or more bits of information.

The state is a collection of state variables whose values at any one time contain all the information about the past necessary to account for the circuit's future behavior.

The simplest sequential circuit, no way to control its state.

Two states

One state variable, say, Q, two possible states



ECL

How to control it?

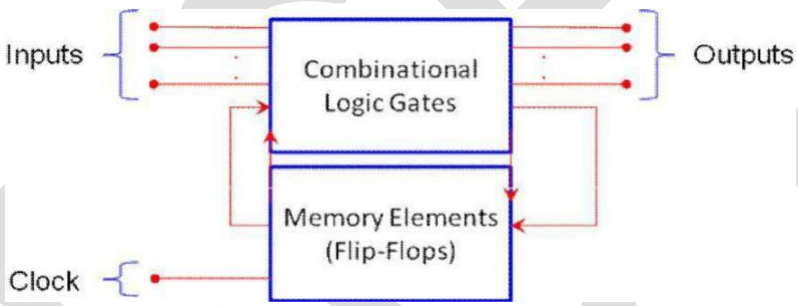
Control inputs

A *bistable memory device* is the generic term for the elements we are studying.

Latches and Flip flops

Latches and flip-flops (FFs) are the basic building blocks of sequential circuits.

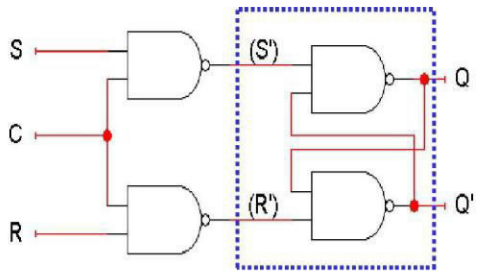
- □ latch: bistable memory device with level sensitive triggering (no clock), watches all of its inputs continuously and changes its outputs at any time, independent of a clocking signal.
- □ flip-flop: bistable memory device with edge-triggering (with clock), samples its inputs, and changes its output only at times determined by a clocking signal.



- Here a is an SR latch with control input C.
  - Notice the hierarchical design!
  - The dotted blue box is the S'R' latch.
  - The additional NAND gates are simply used to generate the correct inputs for the S'R' latch.

The control input acts just like an enable.

ECL



					Q
0			1	1	No change
1	0	0	1	1	No change
1	0	1	1	0	0 (reset)
1	1	0	0	1	1 (set)
1	1	1	0	0	Avoid!

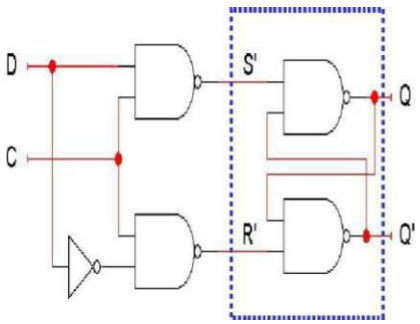
6. latch

Finally, a D latch is based on an S'R' latch. The additional gates generate the S' and R' signals, based on inputs D (—data‖) and C (—control‖).

– When C = 0, S' and R' are both 1, so the state Q does not change.

– When C = 1, the latch output Q will equal the input D.

No more messing with one input for set and another input for reset!



	Q
0	No change
1	0
1	1

12. Also, this latch has no —bad‖ input combinations to avoid. Any of the four possible assignments to C and D are valid.

Flip-flops

-- Here is the internal structure of a D flip-flop.

– The flip-flop inputs are C and D, and the outputs are Q and Q'.

– The D latch on the left is the master, while the SR latch on the right is called the slave.

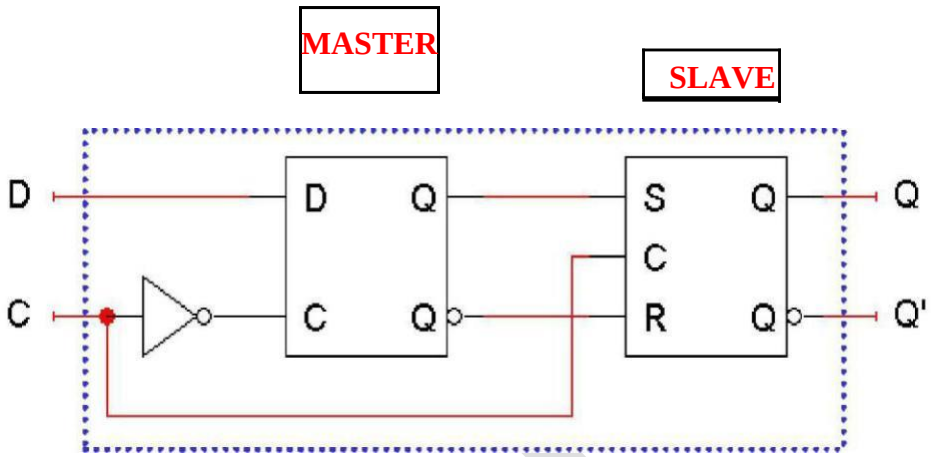
-- Note the layout here.

– The flip-flop input D is connected directly to the master latch.

– The master latch output goes to the slave.

– The flip-flop outputs come directly from the slave latch.

ECL



-- flip-flops when C=0

The D flip-flop's control input C enables *either* the D latch or the SR latch, but not both.

When C = 0:

– The master latch is enabled, and it monitors the flip-flop input D. Whenever D changes, the master's output changes too.

– The slave is disabled, so the D latch output has no effect on it. Thus, the slave just maintains the flip-flop's current state.

D flip-flops when C=1

-- *As soon as C becomes 1,*

– The master is disabled. Its output will be the *last* D input value seen just before C became 1.

– Any subsequent changes to the D input while C = 1 have no effect on the master latch, which is now disabled.

– The slave latch is enabled. Its state changes to reflect the master's output, which again is the D input value from right when C became 1.

Positive edge triggering

J: This is called a positive edge-triggered flip-flop.
The flip-flop output Q changes *only* after the positive edge of C.

ECE

– The change is based on the flip-flop input values that were present right at the positive edge of the clock signal.

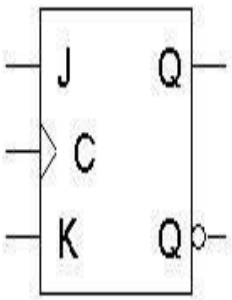
The D flip-flop's behavior is similar to that of a D latch except for the positive edge-triggered nature, which is not explicit in this table

	Q
	No change
	AA: (
	reset) 1
	(set)

Flip-flop variations

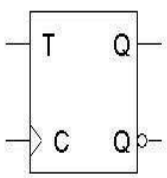
H: We can make different versions of flip-flops based on the D flip-flop, just like we made different latches based on the S'R' latch.

I: A JK flip-flop has inputs that act like S and R, but the inputs JK=11 are used to *complement* the flip-flop's current state.



C	J	K	Q_{next}
0	x	x	No change
1	0	0	No change
1	0	1	0 (reset)
1	1	0	1 (set)
1	1	1	Q'current

A T flip-flop can only maintain or complement its current state



	Qnext
	No change
0	No change
1	Q'current

Characteristic equations

H: We can also write characteristic equations, where the next state Q(t+1) is defined in terms of the current state Q(t) and inputs.

D	Q(t+1)	Operation
0	0	Reset
1	1	Set

Q(t+1) = D

J	K	Q(t+1)	Operation
0	0	Q(t)	No change
0	1	0	Reset
1	0	1	Set
1	1	Q'(t)	Complement

Q(t+1) = K'Q(t) + JQ'(t)

T	Q(t+1)	Operation
0	Q(t)	No change
1	Q'(t)	Complement

t+1) = T'Q(t) + TQ'(t)

= T Q(t)

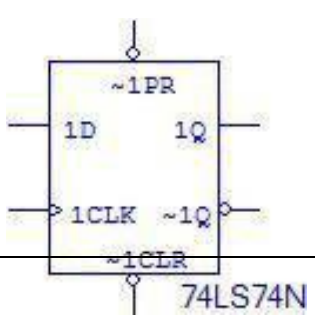
Flip-Flop Vs. Latch

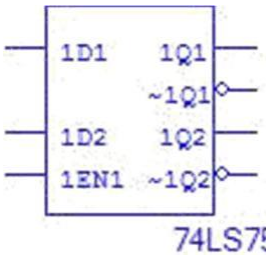
H: The primary difference between a D flip-flop and D latch is the EN/CLOCK input.

I: The flip-flop's CLOCK input is edge sensitive, meaning the flip-flop's output changes on the edge (rising or falling) of the CLOCK input.

J: The latch's EN input is level sensitive, meaning the latch's output changes on the level (high or low) of the EN input.

Flip-Flops & Latches

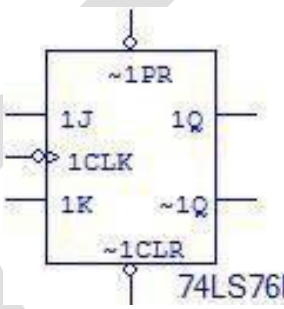




74LS74

Dual Positive-Edge-Triggered D Flip-Flops with Preset, Clear, and Complementary Outputs

74LS76



Dual Negative-Edge-Triggered J-K Flip-Flops with Preset, Clear, and Complementary Outputs 74LS75 Quad Latch

74LS74: D Flip-Flop

Function Table

Inputs				Outputs	
PR	CLR	CLK	D	Q	$\overline{Q}$
L	H	X	X	H	L
H	L	X	X	L	H
L	L	X	X	H (Note 1)	H (Note 1)
H	H	↑	H	H	L
H	H	↑	L	L	H
H	H	L	X	Q_0	$\overline{Q}_0$

H = HIGH Logic Level
X = Either LOW or HIGH Logic Level
L = LOW Logic Level
↑ = Positive-going Transition
Q_0 = The output logic level of Q before the indicated input conditions were established.

Note 1: This configuration is nonstable; that is, it will not persist when either the preset and/or clear inputs return to their inactive (HIGH) level.

ECE

74LS76: J/K Flip-Flop

Function Table

Inputs					Outputs	
PR	CLR	CLK	J	K	Q	$\overline{Q}$
L	H	X	X	X	H	L
H	L	X	X	X	L	H
L	L	X	X	X	H (Note 1)	H (Note 1)
H	H	⎍	L	L	Q_0	$\overline{Q}_0$
H	H	⎍	H	L	H	L
H	H	⎍	L	H	L	H
H	H	⎍	H	H	Toggle	

H = High Logic Level
L = Low Logic Level
X = Either Low or High Logic Level
⎍ = Positive pulse data. The J and K inputs must be held constant while the clock is high. Data is transfered to the outputs on the falling edge of the clock pulse.
Q_0 = The output logic level before the indicated input conditions were established.
Toggle = Each output changes to the complement of its previous level on each complete active high level clock pulse.

Note 1: This configuration is nonstable; that is, it will not persist when the preset and/or clear inputs return to their inactive (high) level.

74LS75: D Latch

Function Table (Each Latch)

Inputs		Outputs	
D	Enable	Q	$\overline{Q}$
L	H	L	H
H	H	H	L
X	L	Q_0	$\overline{Q}_0$

H = HIGH Level
L = LOW Level
X = Don't Care
Q_0 = The Level of Q Before the HIGH-to-LOW Transition of ENABLE

ECE

Counters

H Counters are a specific type of sequential circuit.

I Like registers, the state, or the flip-flop values themselves, serves as the ―output.‖

J The output value increases by one on each clock cycle.

K After the largest value, the output ―wraps around‖ back to 0.

L Using two bits, we'd get something like this:

resent State		xt State	
A	B	A	B
0	0		1
0	1	1	0
1	0	1	1
1	1		0

H: Counters can act as simple clocks to keep track of ―time.‖

I: You may need to record how many times something has happened.

– How many bits have been sent or received?

– How many steps have been performed in some computation?

J: All processors contain a program counter, or PC.

– Programs consist of a list of instructions that are to be executed one after another (for the most part).

– The PC keeps track of the instruction currently being executed.

– The PC increments once on each clock cycle, and the next program instruction is then executed.

Asynchronous Counters

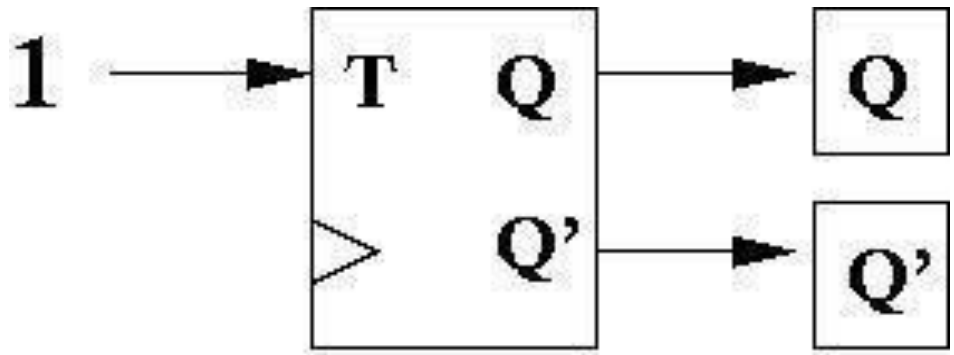
H: This counter is called *asynchronous* because not all flip flops are hooked to the same clock.

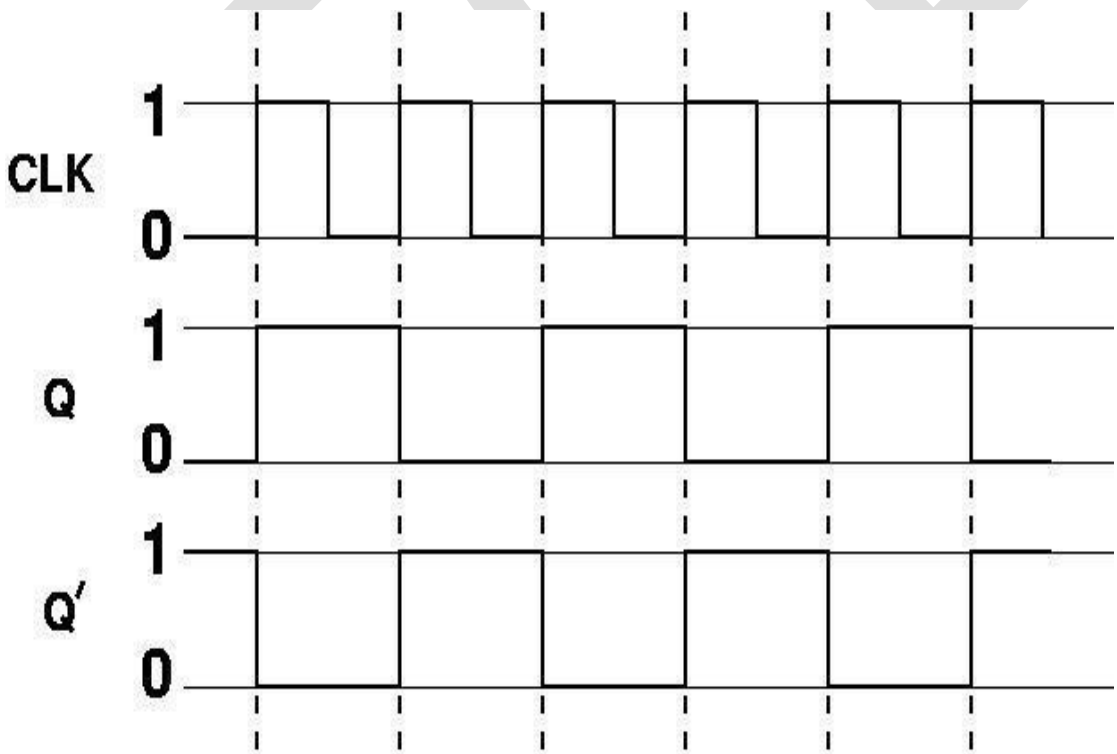
I: Look at the waveform of the output, **Q**, in the timing diagram. It resembles a clock as well. If the period of the clock is T, then what is the period of **Q**, the output of the flip flop? It's 2T!

We have a way to create a clock that runs twice as slow. We feed the clock into a T flip flop, where T is hardwired to 1. The output will be a clock who's period is twice as long.

If the clock has period T. **Q0** has period 2T. **Q1** period is 4T

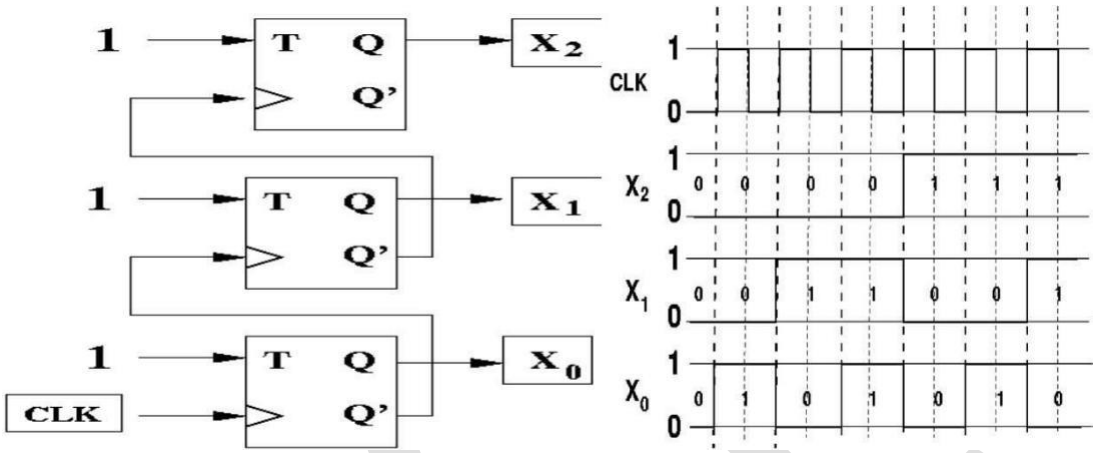
C With n flip flops the period is 2^n





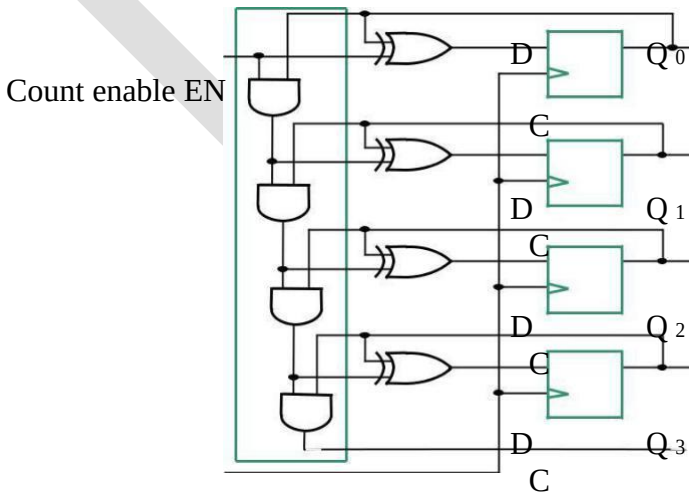
3 bit asynchronous "ripple" counter using T flip flops

This is called as a *ripple counter* due to the way the FFs respond one after another in a kind of rippling effect.



Synchronous Counters

- To eliminate the "ripple" effects, use a common clock for each flip-flop and a combinational circuit to generate the next state.
- For an up-counter, use an incrementer =>
- Internal details =>



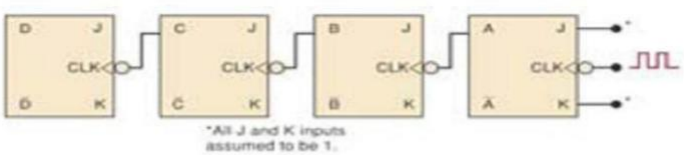
- Internal Logic
  (a) Logic Diagram-Serial Gating
- XOR complements each bit
- AND chain causes complement of a bit if all bits toward LSB from it equal 1
- Count Enable
- Forces all outputs of AND chain to 0 to ―hold‖ the state
- Carry Out
- Added as part of incrementer
- Connect to Count Enable of additional 4-bit counters to form larger counters

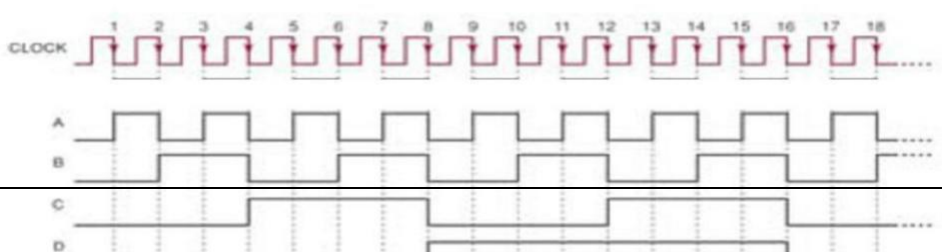
Design of Counters using digital ICs

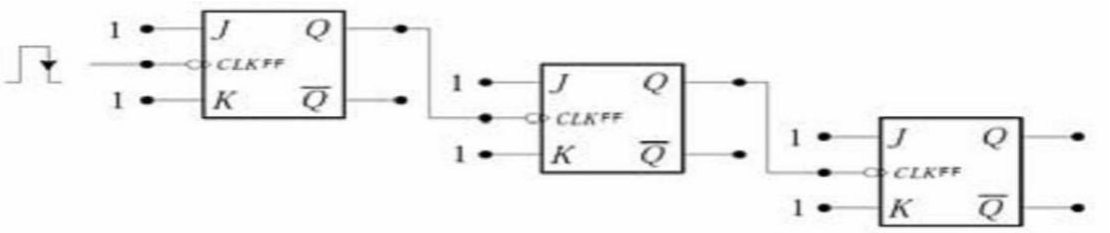
Asynchronous (Ripple) Counters

Clock is applied only to FF A. J and K are high in all FFs to toggle on every clock pulse. Output of FF A is CLK of FF B and so forth.FF outputs D, C, B, and A are a 4 bit binary number with D as the MSB.After the negative transistion of the 15th clock pulse the 0000.This is an asynchronous counter because state is not changed in exact synchronism with the clock.

Four-bit asynchronous (ripple) counter Frequency division The output frequency of each FF = the clock frequency of input / 2. The output frequency of the last FF = the clock frequency / MOD





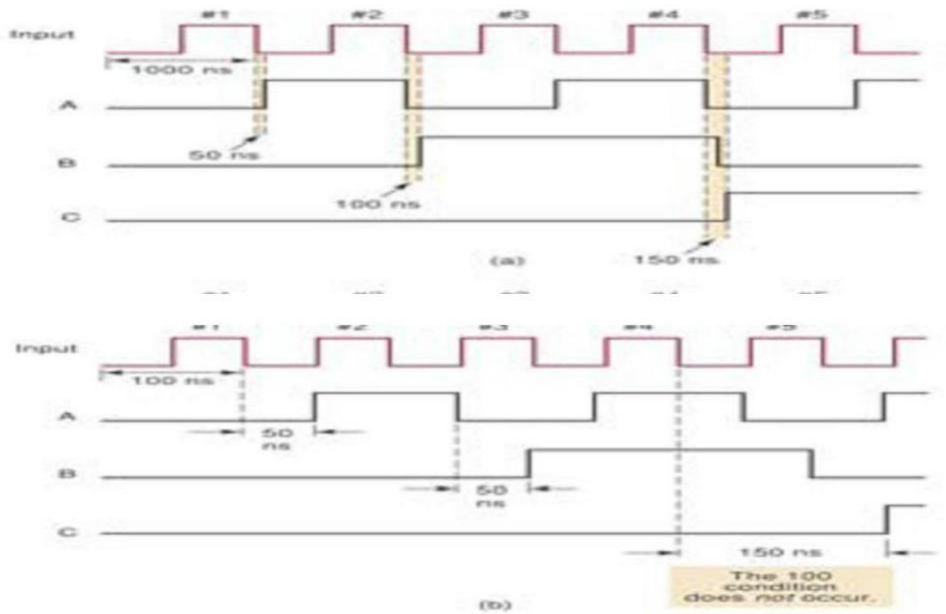


Propagation Delay in Ripple Counters

Ripple counters are simple, but the cumulative propagation delay can cause problems at high Frequencies. For proper operation the following apply:

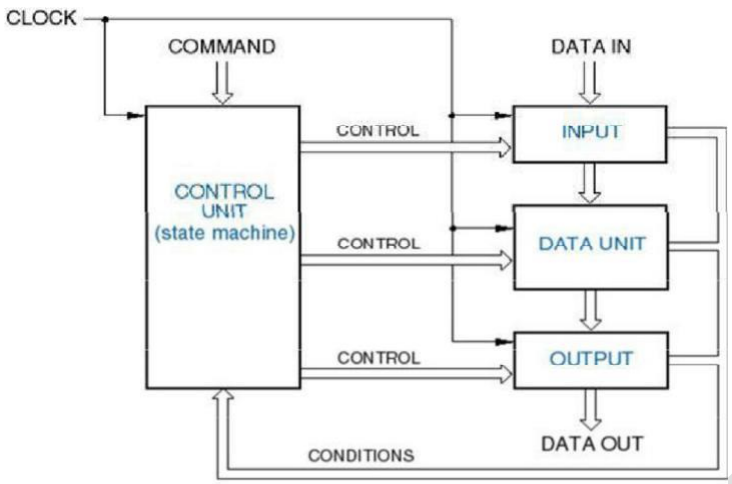
$T\text{clock} \geq N \times tpd$

$Fmax = 1/(N \times tpd)$



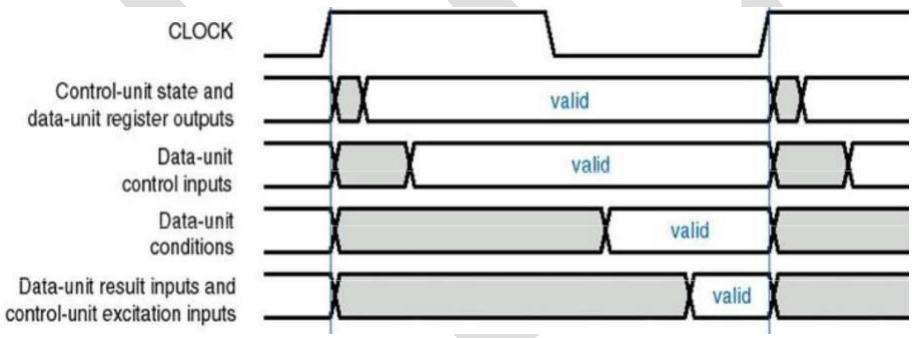
Synchronous design methodology

Synchronous System Structure



Everything is clocked by the same, common clock

Typical synchronous-system timing



• Outputs have one complete clock period to propagate to inputs.

• Must take into account flip-flop setup times at next clock period.

Clock Skew

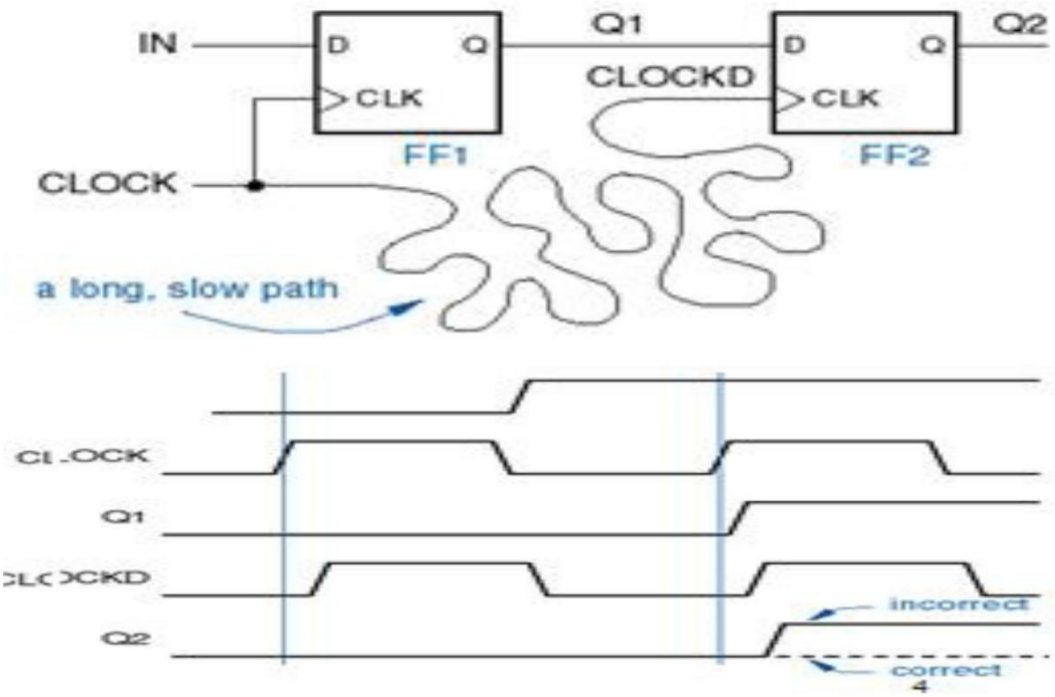
Clock signal may not reach all flip-flops simultaneously. Output changes of flip-flops receiving ―early‖ clock may reach D inputs of flip-flops with ―late‖ clock too soon.

Reasons for slowness:

(a) wiring delays

(b) capacitance

(c) incorrect design



Clock-skew

calculation

$t_{ffpd(min)} + t_{comb(min)} - t_{hold} - t_{skew(max)} > 0$

First two terms are minimum time after clock edge that a D input changes Hold

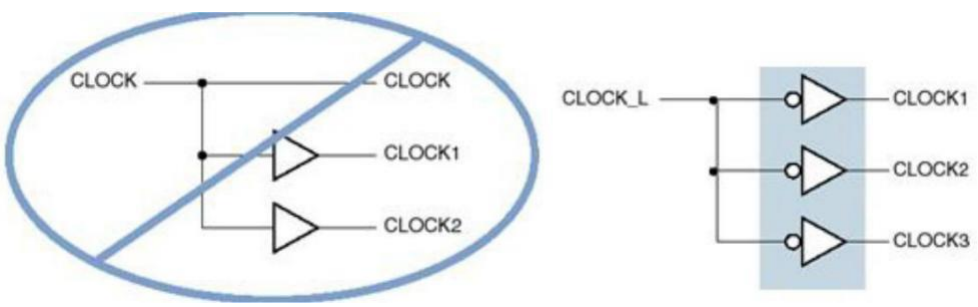
time is earliest time that the input may change

Clock skew subtracts from the available hold-time margin

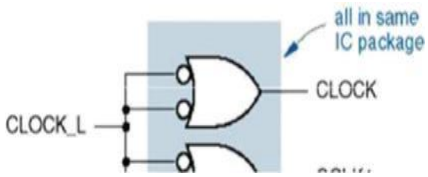
Compensating for clock skew:

- Longer flip-flop propagation delay
- Explicit combinational delays
- Shorter (even negative) flip-flop hold times

Example of bad clock distribution



Gating the clock

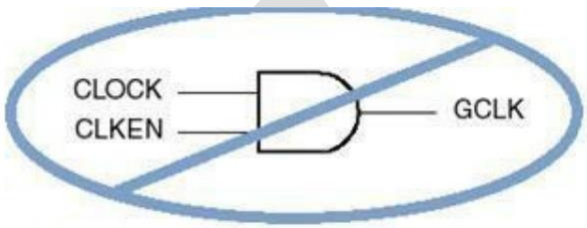


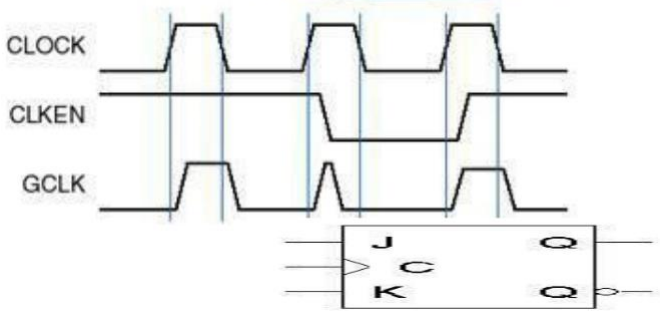
Definitely a no-no

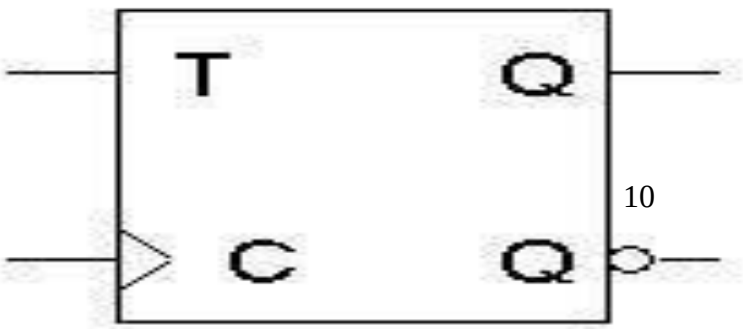
– Glitches possible if control signal (CLKEN) is generated by the same clock

– Excessive clock skew in any case.

If you really must gate the clock...







Asynchronous inputs

Not all inputs are synchronized with the clock

ECE

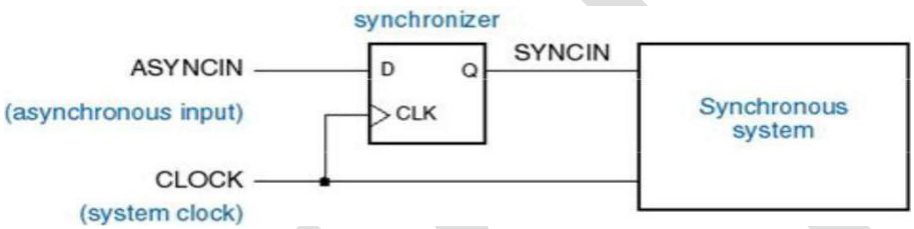
Examples:

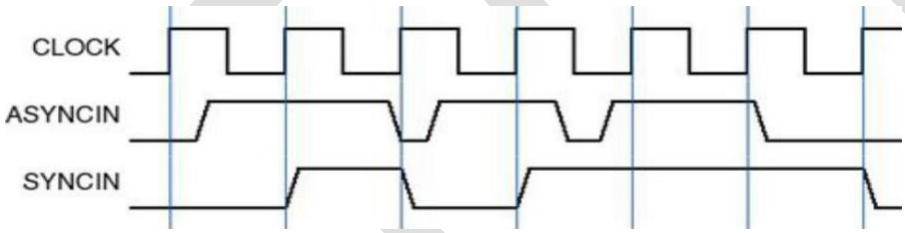
- Keystrokes
- Sensor inputs

- Data received from a network (transmitter has its own clock)

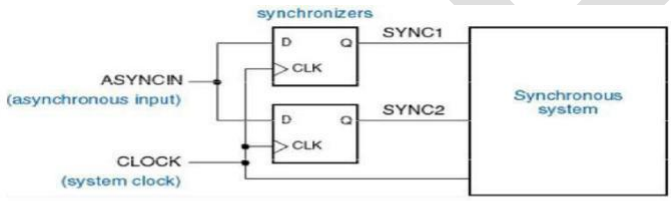
Inputs must be synchronized with the system clock before being applied to a synchronous system.

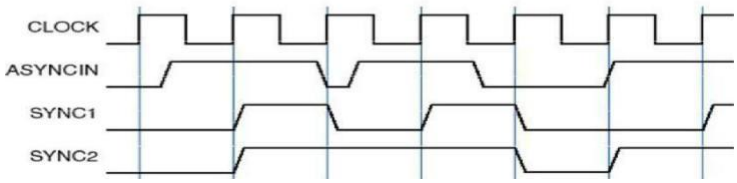
A simple synchronizer



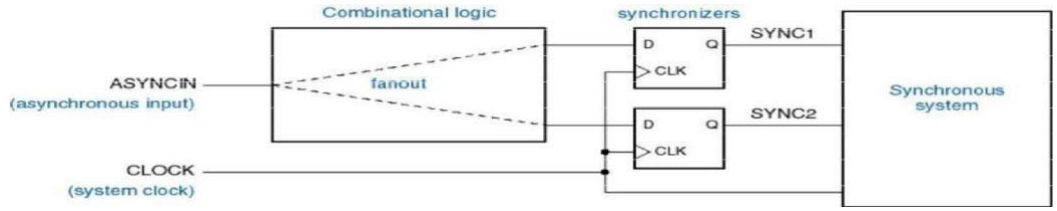


Only **one** synchronizer per input





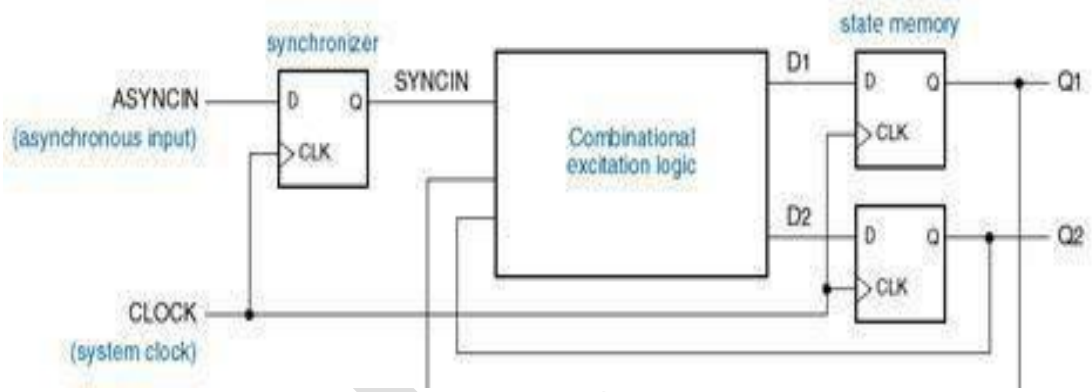
Even worse



- Combinational delays to the two synchronizers are likely to be different.

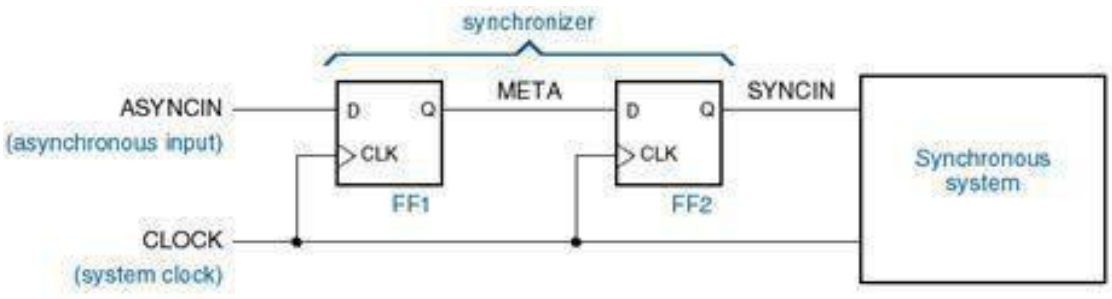
way to do it

One synchronizer per input. Carefully locate the synchronization points in a sys .But still a problem -- the synchronizer output may become metastable when setup and hold time are not met.



Recommended synchronizer design

Hope that FF1 settles down before ―META‖ is sampled.In this case, ―SYNCIN‖ is valid for almost a full clock period.Can calculate the probability of ―synchronizer failure‖ (FF1 still metastable when META sampled.



Impediments to synchronous design

- Clock skew

– Definition

The difference between arrival times of the clock at different memory devices

Example of clock skew

Influence of clock skew

Reduce the setup and hold time margins. For proper operation tffpd(min) + tcomb(min) - thold - tskew(max) > 0

tsetup -tclk -tffpd(max) - tcomb(max) - tskew(max) > 0 Reducing clock skew

proper buffering the clock Better clock distribution .

Gating clock

Why not to gate the clock . An acceptable way.

Asynchronous inputs

Why use the asynchronous inputs? Problem with asynchronous inputs

Meta-stable

Need synchronizers

A simple one.