ANALOG AND DIGITAL ELECTRONICS

ANALOG AND DIGITAL ELECTRONICS

(Common to CSE & ISE)

Subject Code: 15CS32	**I.A. Marks : 20**
Hours/Week : 04	**Exam Hours: 03**
Total Hours : 50	**Exam Marks: 80**

MODULE – 1 **10 Hours**

Field Effect Transistors: Junction Field Effect Transistors, MOSFETs, Differences between JFETs and MOSFETs, Biasing MOSFETs, FET Applications, CMOS Devices. Wave-Shaping Circuits: Integrated Circuit(IC) Multivibrators.

Introduction to Operational Amplifier: Ideal v/s practical Opamp, Performance Parameters,

Operational Amplifier Application Circuits:Peak Detector Circuit, Comparator, Active Filters, Non-Linear Amplifier, Relaxation Oscillator, Current-To-Voltage Converter, Voltage-To- Current Converter

MODULE – 2 **10 Hours**

The Basic Gates: Review of Basic Logic gates, Positive and Negative Logic, Introduction to HDL.

Combinational Logic Circuits: Sum-of-Products Method, Truth Table to Karnaugh Map, Pairs Quads, and Octets, Karnaugh Simplifications, Don't-care Conditions, Product-of-sums Method, Product-of-sums simplifications, Simplification by Quine-McClusky Method, Hazards and Hazard covers, HDL Implementation Models

MODULE – 3 **10 Hours**

Data-Processing Circuits: Multiplexers, Demultiplexers, 1-of-16 Decoder, BCD to Decimal Decoders, Seven Segment Decoders, Encoders, Exclusive-OR Gates, Parity Generators and Checkers, Magnitude Comparator, Programmable Array Logic, Programmable Logic Arrays, HDL Implementation of Data Processing Circuits. Arithmetic Building Blocks, Arithmetic Logic Unit

Flip-Flops: RS Flip-Flops, Gated Flip-Flops, Edge-triggered RS FLIP-FLOP, Edge triggered D FLIP-FLOPs, Edge triggered JK FLIP-FLOPs

MODULE – 4 **10 Hours**

Flip- Flops: FLIP-FLOP Timing, JK Master-slave FLIP-FLOP, Switch Contact Bounce Circuits, Various Representation of FLIP-FLOPs, HDL Implementation of FLIP-FLOP.

Registers: Types of Registers, Serial In - Serial Out, Serial In - Parallel out, Parallel In - Serial Out, Parallel In - Parallel Out, Universal Shift Register, Applications of Shift Registers, Register implementation in HDL.

Counters: Asynchronous Counters, Decoding Gates, Synchronous Counters, Changing the Counter Modulus.

MODULE – 5 **10 Hours**

Counters: Decade Counters, Pre settable Counters, Counter Design as a Synthesis problem, A Digital Clock, Counter Design using HDL.

D/A Conversion and A/D Conversion: Variable, Resistor Networks, Binary Ladders, D/A Converters, D/A Accuracy and Resolution, A/D Converter- Simultaneous Conversion, A/D Converter-Counter Method, Continuous A/D Conversion, A/D Techniques, Dual-slope A/D Conversion, A/D Accuracy and Resolution.

Text Book:

1. Anil K Maini, Varsha Agarwal: Electronic Devices and Circuits, Wiley, 2012.
2. Donald P Leach, Albert Paul Malvino & Goutam Saha: Digital Principles and Applications, 8th Edition, Tata McGraw Hill, 2015

TABLE OF CONTENTS

MODULE 1 (CONT.): OPERATIONAL AMPLIFIER

OPERATIONAL AMPLIFIER (OP-AMP)

• An op-amp is a DC-coupled high-gain electronic voltage amplifier with
  → differential input and
  → single-ended output (Figure 16.1).

• An op-amp produces an output-voltage that is typically hundreds of thousands times larger than the voltage difference between its input terminals.

• Opamps are important building blocks for a wide range of electronic circuits.

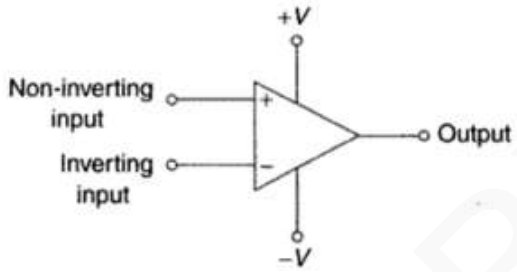


Figure 16.1: Circuit symbol for an op-amp

IDEAL VERSUS PRACTICAL OPAMP

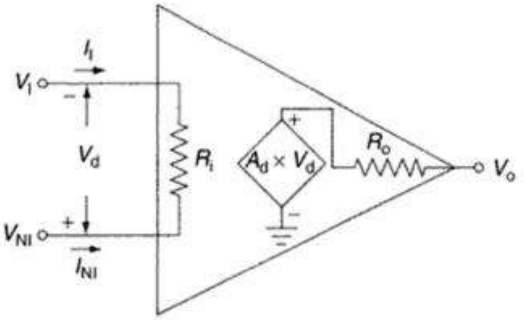


Figure 16.13: Model of practical opamp

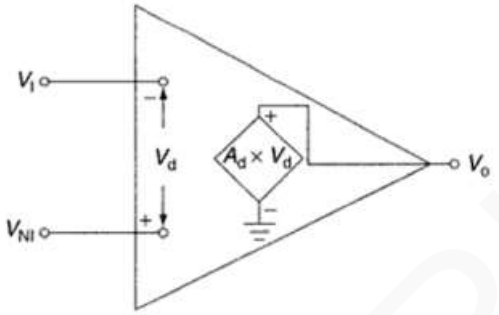


Figure 16.14: Model of ideal opamp

- Here, V_I = Inverting input.
  V_{NI} = Non-inverting input.
- The ideal opamp model was derived to simplify circuit calculations (Figure 16.14).
- The ideal opamp model makes 3 assumptions. They are:
  1. Input impedance, $Z_i = \infty$.
  2. Output impedance, $Z_o = 0$.
  3. Open-loop gain, $A_d = \infty$.
- From the above 3 assumptions, other assumptions can be derived. They are:
  1. Since $Z_i = \infty$, $I_I = I_{NI} = 0$.
  2. Since $Z_o = 0$, $V_o = A_d \times V_d$.
  3. Common mode gain = 0.
  4. Bandwidth = ∞.
  5. Slew Rate = ∞.
  6. Offset Drift = 0.
- Properties of ideal opamp are:
  1. Infinite open-loop differential voltage gain.
  2. Infinite input impedance.
  3. Zero output impedance.
  4. Infinite bandwidth.
  5. Zero DC input and output offset voltages.
  6. Zero input differential voltage.

Properties	**Ideal opamp**	**Practical opamp**
Voltage Gain.	Ideal opamps have infinite open-loop voltage gain (Figure 16.14).	Practical opamps have open-loop gain in the range of 10,000 to 100,000 (Figure 16.13).
Input Impedance.	Ideal opamps have infinite input impedance.	Input impedance varies from hundreds of KΩ for some low-grade opamps to TΩ for high-grade opamps.
Output Impedance.	Ideal opamps have zero output impedance.	Output impedance may be in the range of 10 to 100 KΩ.
Bandwidth.	Infinite bandwidth i.e. ideal opamp amplifies all signals from DC to highest AC frequencies.	Bandwidth is limited and is specified by gain-bandwidth product.
DC Input & output offset voltages.	An ideal opamp produces a zero DC output when both the inputs are grounded.	For real devices, there may be some finite DC output even when both the inputs are grounded. Output offset may vary from few nano-volts for ultra-low offset opamps to kw milli-volts for general-purpose opamps.
Input differential voltage.	Zero input differential voltage. Voltage appearing at 1 input also appears at the other input for linear mode of operation i.e. differential inputs stick together.	Practical Opamp exhibits offsets and non-linearity.

PERFORMANCE PARAMETERS

1. Bandwidth

• Bandwidth refers to range of frequencies the opamp can amplify for a given amplifier-gain.
• When the opamp is used in the closed-loop mode, the bandwidth increases at the cost of the gain.
• The bandwidth is usually expressed in terms of the unity gain crossover frequency (also called *gain-bandwidth product*).
• It is 1 MHz in the case of opamp 741.
It is 1500 MHz in the case of high-bandwidth opamp.

2. Slew Rate

• Slew-rate is defined as the rate of change of output-voltage with time.
• It gives us an idea about how well the output follows a rapidly changing waveform at the input.
• It limits the large signal bandwidth.
• For a sinusoidal signal,
peak-to-peak output-voltage($V_{o\text{-}t\text{-}o}$), slew rate & bandwidth are inter-related by following equation:

$$\text{Bandwidth (highest frequency, } f_{MAX}) = \text{Slew rate}/(\pi \times V_{p\text{-}t\text{-}p})$$

3. Open-loop Gain

• Open-loop gain is the ratio of single-ended output to the differential input.
• This parameter has a great bearing on the gain-accuracy specification of the opamp.
• The ratio of the open-loop gain to the closed-loop gain is called the **loop-gain**.
• Accuracy depends on the magnitude of the loop-gain.
• The magnitude of loop-gain depends directly on the value of the open-loop gain, as the value of closed-loop gain is fixed.

4. Common Mode Rejection Ratio (CMRR)

• CMRR is the ratio of the desired differential gain (A_d) to the undesired common mode gain (A_c).
i.e. CMRR=20 log(A_d/A_c) dB
• It is a measure of the ability of the opamp to suppress common mode signals.

5. Power Supply Rejection Ratio (PSRR)

• PSRR is defined as the ratio of change in the power supply voltage to corresponding change in the output-voltage.
• PSRR should be zero for an ideal opamp.

6. Input Impedance

• Input Impedance is the ratio of input-voltage to input-current.
• It is the impedance looking into the input terminals of the opamp.
• It is expressed in terms of resistance.
• It is assumed to be infinite to prevent any current flowing from the source supply into the amplifiers input circuitry.

7. Output Impedance

• Output Impedance is defined as the impedance between the output terminal of the opamp and ground.
• For ideal opamp, the output impedance is assumed to be zero.
• The opamp acts as a perfect internal voltage-source with no internal resistance so that it can supply as much current as necessary to the load.
• This internal resistance is effectively in series with the load thereby reducing the output-voltage available to the load.

8. Settling Time

• Settling time is a parameter specified in the case of
→ high speed opamps or
→ opamps with a high value of gain-bandwidth product.

9. Offset Voltage (V_{io})

• Output offset voltage is the voltage at the output with both the input terminals grounded.
• Real opamps have some amount of output offset voltage.
• In real opamps, we need to apply a DC differential voltage externally to get a zero output. This externally applied input is referred to as input offset voltage.

Example 16.4

Opamp LM 741 is specified to have a slew rate of 0.5 V/μs. if the opamp were used as an amplifier and the expected peak output-voltage were 10V, determine the highest sinusoidal frequency that would get satisfactorily amplified.

Solution:

1. Highest sinusoidal frequency f_{MAX} that would get satisfactorily amplified is given by
   f_{MAX} = slew rate/$2\Pi V_p$
   where V_p is the expected peak output-voltage.
2. In the present case, slew rate = 0.5 V/μs and V_p = 10 V.
3. Therefore, $f_{MAX} = (0.5*10^6)/(2\Pi*10)=7.96$ KHz

Example 16.5

Differential voltage gain and CMRR of an opamp when expressed in decibels are 110 dB and 100 dB respectively. Determine the common mode gain expressed as a ratio.

Solution:

1. CMMR (in dB) = 20 log (A_d/A_{CM})
   where A_d= differential voltage gain
   A_{CM}= common mode gain
2. CMMR (in dB) = 20 log A_d - 20 log A_{CM}
3. That is 20 log A_{CM} = 20 log A_d - CMMR =110-100=10 dB
4. This gives log A_{CM} = 10/20=0.5
5. Therefore, A_{CM} = antilog(0.5) = 3.16

Example 16.6

In the case of a certain opamp, 0.5 V change in common mode input causes a DC output offset change of 5μV. Determine CMRR in dB.

Solution:

1. CMMR = $\Delta V_{CM}/\Delta V_{OS} = 0.5/(5*10^{-6})=10^5$
2. CMMR in dB= 20 log 10^5 =100 dB.

MODULE 1 (CONT.): OPERATIONAL AMPLIFIER APPLICATION CIRCUITS

PEAK DETECTOR CIRCUIT

- It is one of the applications of opamp (Figure 17.34).
- Peak detector circuit produces a voltage at the output equal to peak amplitude of the input signal.
- Essentially, it is a clipper-circuit with a parallel resistor-capacitor connected at its output.
- Here is how it works:
  - The clipper reproduces the positive half cycles.
  - During this period, the diode D_1 is forward-biased.
  - The capacitor rapidly charges to the positive peak from the output of the opamp.
  - As the input starts decreasing beyond the peak, the diode gets reverse biased, thus isolating the capacitor from the output of the opamp.
  - The capacitor can now discharge only through the resistor (R) connected across it.
- The value of the resistor is much larger than the forward-biased diode's ON resistance.
- The buffer-circuit prevents any discharge of the capacitor due to loading effects of the circuit.
- The circuit can be made to respond to the negative peaks by reversing the polarity of the diode.

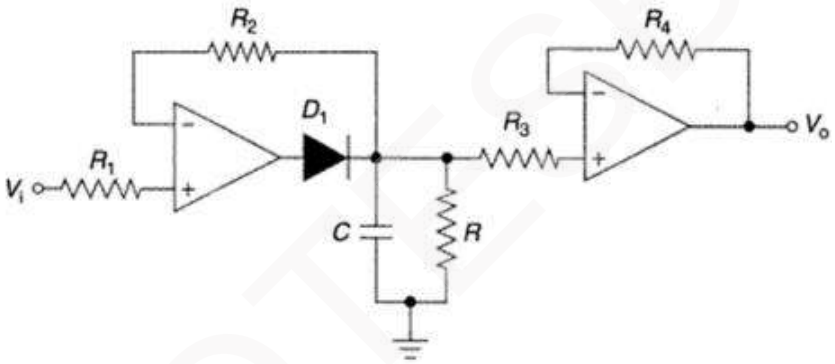


Figure 17.34: Peak Detector Circuit

- The parallel R-C circuit time constant is typically 100 times the time period corresponding to the minimum frequency of operation.
- The R-C time constant also controls the response time.
- Slew rate is the primary specification that needs to be looked into while choosing the right opamp for the clipper portion.

COMPARATOR

- A comparator circuit is a two input, one-output building block.
- It produces a high or low output depending upon the relative magnitudes of the 2 inputs.
- An opamp can be very conveniently used as a comparator when used without negative feedback.
- Because of very large value of open-loop voltage gain, it produces either positively saturated or negatively saturated output-voltage.
- The output-voltage depends on whether the amplitude of the voltage applied at the non-inverting terminal is more or less positive than the voltage applied at the inverting input terminal.

ZERO CROSSING DETECTOR

- The comparator has 2 inputs:
  1) First input is connected to standard reference voltage.
  2) Second input is connected to input-voltage that needs to be compared with the reference voltage.
- In special case, where reference voltage is 0, the circuit is referred to as **zero-crossing detector**.
- Here, we consider 2 cases: 1) Non-inverting zero-crossing detector &
  2) Inverting zero-crossing detector.

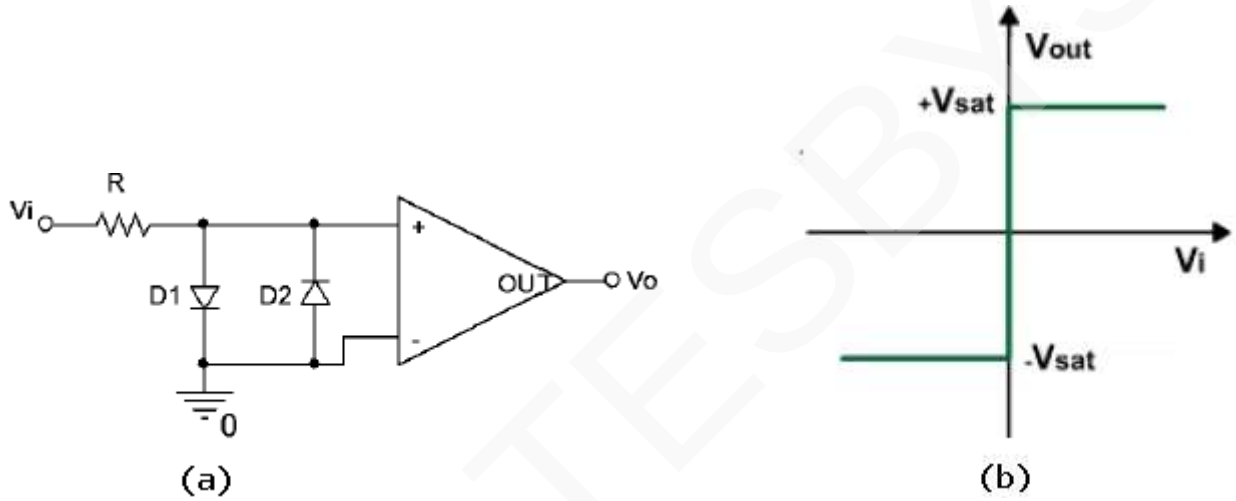


Figure 17.38: Non-inverting zero-crossing detector

- In **non-inverting zero-crossing detector**, input-voltage more positive than zero produces a positively saturated output-voltage (Figure 17.38).
- Diodes D_1 and D_2 connected at the input are to protect the sensitive input circuits inside the opamp from excessively large input-voltages.

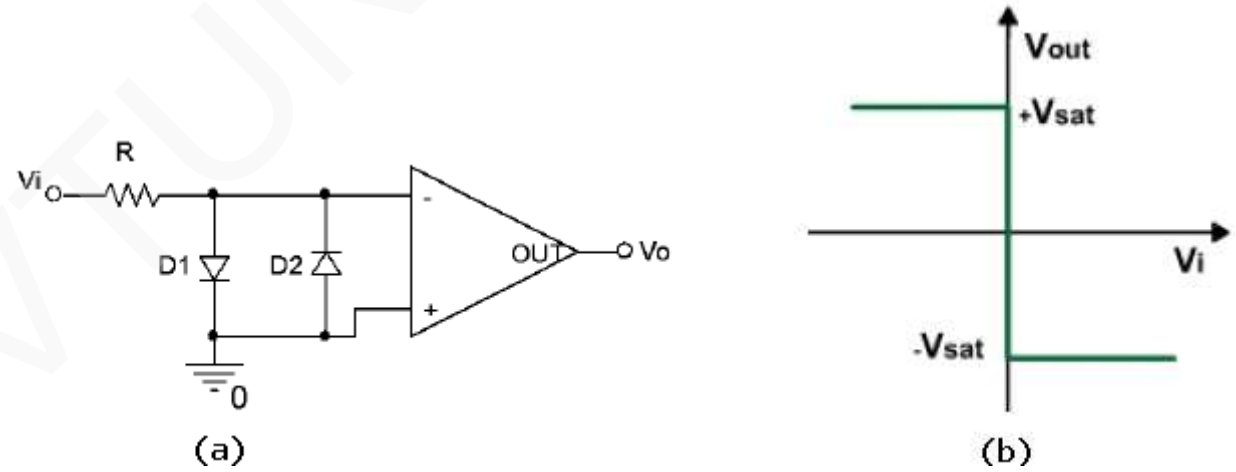


Figure 17.39: Inverting zero-crossing detector

- In **inverting zero-crossing detector**, input-voltage slightly more positive than zero produces a negatively saturated output-voltage (Figure 17.39).

• **Common Application of zero-crossing detector**: To convert sine wave signal to a square wave signal (Figure 17.40).

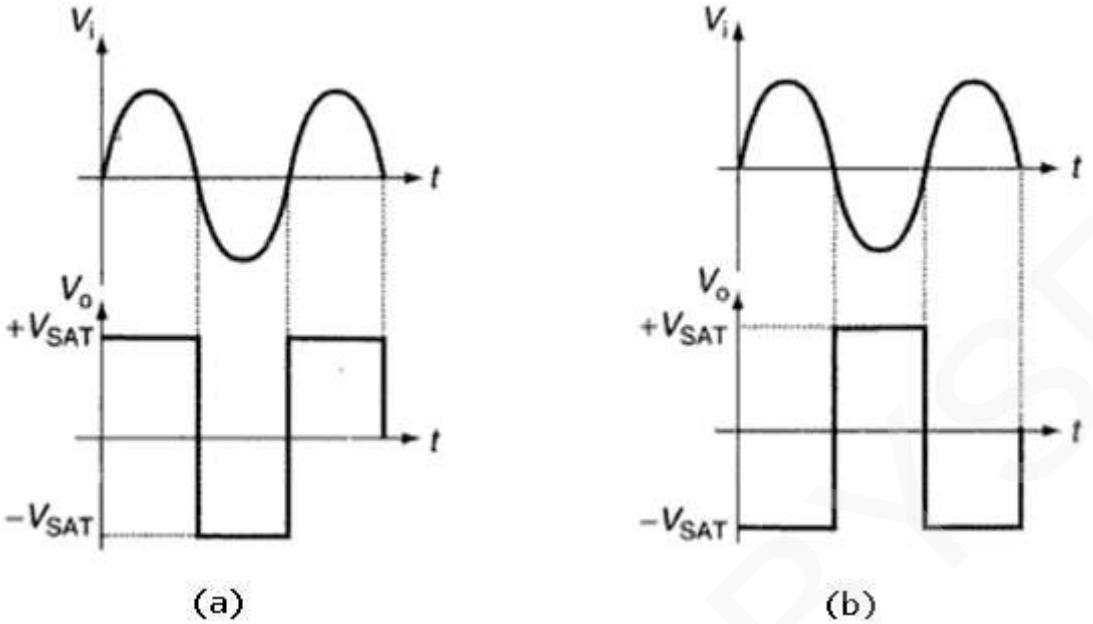


Figure 17.40: Waveform of (a)Non-inverting zero-crossing detector (b)Inverting zero-crossing detector

• In general, reference voltage may be
→ positive (Figure 17.41) or
→ negative voltage (Figure 17.42).

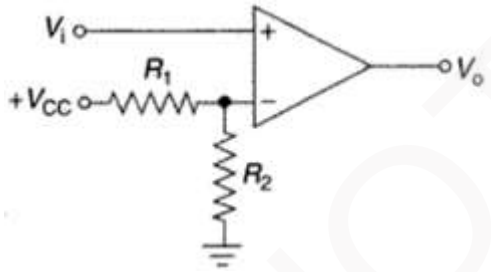


Figure 17.41: Non-inverting comparator with positive reference

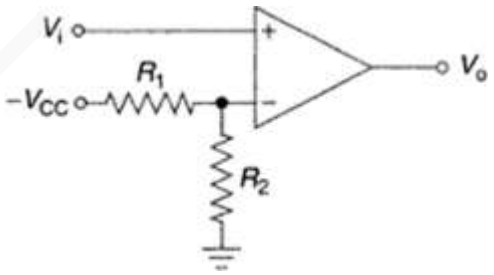


Figure 17.42: Non-inverting comparator with negative reference

• In case of non-inverting comparator,
A positive reference voltage, V_{REF} is given by

$$+V_{CC} \times [R_2/(R_1 + R_2)]$$

A negative reference voltage, V_{REF} is given by

$$-V_{CC} \times [R_2/(R_1 + R_2)]$$

COMPARATOR WITH HYSTERESIS

• Here we consider, 1) Inverting comparator & 2) Non-inverting comparator.

1) Inverting Comparator with Hysteresis

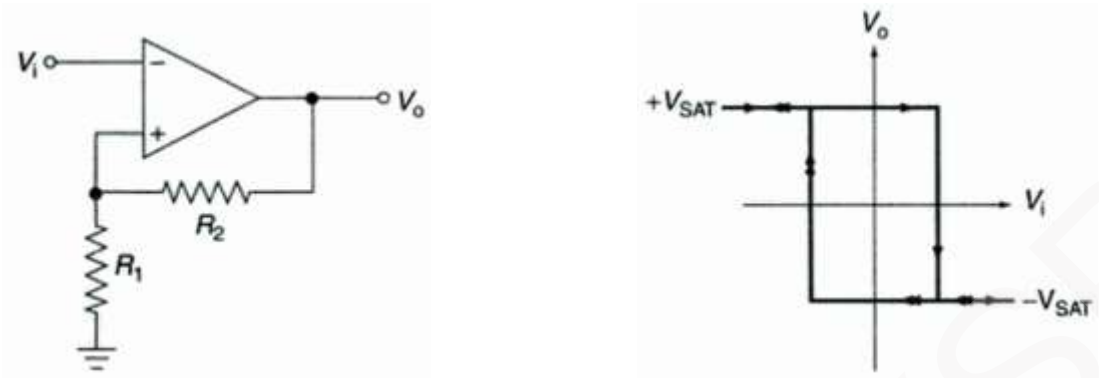


Figure 17.44: Inverting comparator with hysteresis

• Here is how it works:

- Let us assume that the output is in positive saturation($+V_{SAT}$). (Figure 17.44).
- Voltage at non-inverting input is

$$V_{SAT} \times R_1/(R_1 + R_2)$$

- Due to this small positive voltage at the non-inverting input, the output is reinforced to stay in positive saturation.
- Now, the input signal needs to be more positive than this voltage for the output to go to negative saturation.
- Once the output goes to negative saturation ($-V_{SAT}$), voltage fed back to non-inverting input becomes

$$-V_{SAT} \times R_1/(R_1 + R_2)$$

- A negative voltage at the non-inverting input reinforces the output to stay in negative saturation.
- In this manner, the circuit offers a hysteresis of

$$2V_{SAT} \times R_1/(R_1 + R_2)$$

2) Non-inverting Comparator with Hysteresis

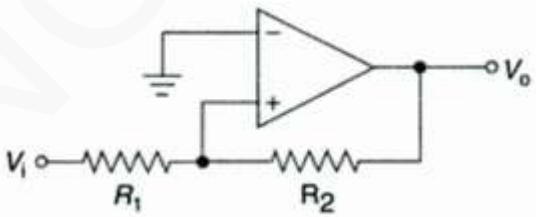


Figure 17.45: Non-inverting comparator with hysteresis

- Non-inverting comparator with hysteresis can be built by applying the input signal to the non-inverting input (Figure 17.45).
- Operation is similar to that of inverting comparator.
- Upper and lower trip points and hysteresis is given by

$$\text{UTP} = +V_{SAT} \times \frac{R_1}{R_2} \qquad \text{LTP} = -V_{SAT} \times \frac{R_1}{R_2} \qquad \text{H} = 2V_{SAT} \times \frac{R_1}{R_2}$$

WINDOW COMPARATOR

• In the case of a conventional comparator, the output changes state when the input-voltage goes above or below the preset reference voltage (Figure 17.46).

• There are 2 reference voltages called **lower and the upper trip points (LTP & UTP)**.(Fig 17.47).

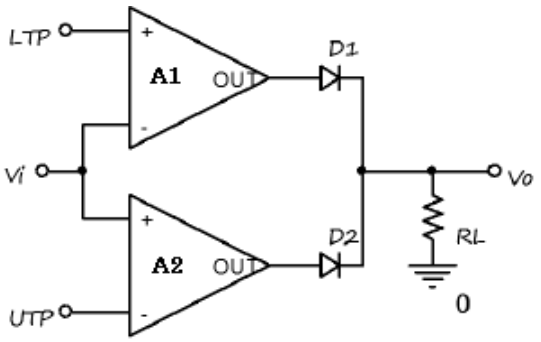


Figure 17.46: Window Comparator

Figure 17.47: Transfer characteristics of Window Comparator

• Here is how it works:

Case 1:

- When the input-voltage is less than the voltage-reference corresponding to the lower trip point (LTP), output of opamp A1 is at $+V_{SAT}$ and the opamp A2 is at $-V_{SAT}$.
- Diodes D_1 and D_2 are respectively forward and reverse biased.
- Consequently, output across R_L is at $+V_{SAT}$.

Case 2:

- When the input-voltage is greater than the reference voltage corresponding to the upper trip point (UTP), the output of opamp A1 is $-V_{SAT}$ and that of opamp A2 is at $+V_{SAT}$.
- Diodes D_1 and D_2 are respectively reverse and forward biased.
- Consequently, output across R_L is at $+V_{SAT}$.

Case 3:

- When the input-voltage is greater than LTP voltage and lower than UTP voltage, the output of both opamps is at $-V_{SAT}$.
- Both diodes D_1 and D_2 are reverse biased.
- Consequently, the output across R_L is zero.

Example 17.11

Refer to the comparator circuit of figure 17.50. Determine the state of LED-1 and LED-2 (whether ON or OFF) when the switch SW-1 is in (a) position-A and (b) position-B. Assume diodes D_1 and D_2 to have forward biased voltage drop equal to 0.7 V each.

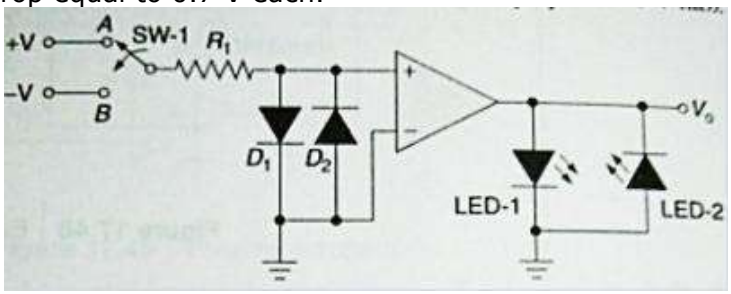


Figure 17.50: solution to Example 17.11

Solution:

1. When the switch SW-1 is in position-A, voltage appearing at non-inverting input is +0.7 V. (equal to forward biased voltage drop across D_1). That is, voltage at non-inverting input is more positive with respect to voltage at inverting input. Therefore, opamp output goes to positive saturation with the result that LED-1 is ON and LED-2 is OFF.
2. When the switch SW-1 is in position-B, voltage appearing at non-inverting input is -0.7 V. (equal to forward biased voltage drop across D_2). That is, voltage at non-inverting input is more negative with respect to voltage at inverting input. Therefore, opamp output goes to negative saturation with the result that LED-1 is OFF and LED-2 is ON.

Example 17.12

Figure 17.51 shows a non-inverting type of window comparator configured around comparator IV LM 339, which is a quad comparator. Determine the lower and upper trip points of the comparator and also draw the output-voltage V_o versus input-voltage V_i transfer characteristics.

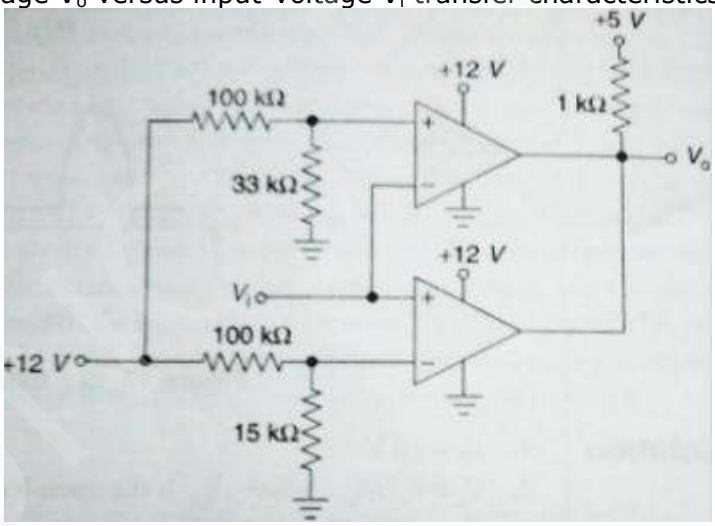


Figure 17.51: Example 17.12

Solution:

1. Lower trip point (LTP) is given by $(12*15*10^3)/(115*10^3)=1.565V$
2. Upper trip point (UTP) is given by $(12*33*10^3)/(133*10^3)=2.977V$
3. Transfer characteristics are shown in figure 17.52.

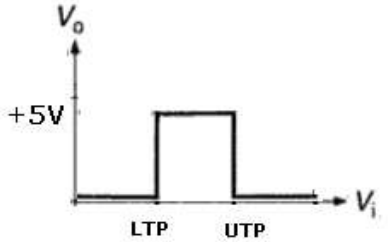


Figure 17.52: Solution to Example 17.12

ACTIVE FILTERS

- Opamp circuits can be used to build:
  1) Low-pass filters.
  2) High-pass filters.
  3) Band-pass filters.
  4) Band-reject filters.
- Also, filters can be classified depending on their order like first-order and second-order.
- Order of an active filter is determined by number of RC sections used in the filter.

FIRST-ORDER FILTERS

- A simple low-pass and high-pass active filters are constructed by connecting lag & lead type of RC sections, respectively, to the non-inverting input of the opamp (Fig 17.54).

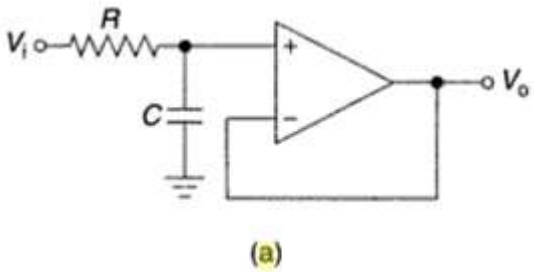


Figure 17.54a: First-order low-pass active filter

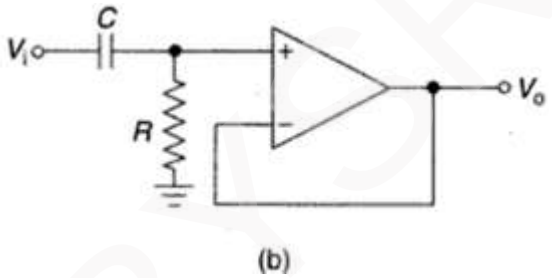


Figure 17.54b: First-order high-pass active filter

- Here is how the low-pass filter works (Figure 17.54a):
  1) At low frequencies, reactance offered by the capacitor is much larger than the resistance value. Therefore, applied input signal appears at the output mostly unattenuated.
  2) At high frequencies, the capacitive reactance becomes much smaller than the resistance value. Thus, forcing the output to be near zero.
- Here is how the high-pass filter works (Figure 17.54b):
  1) At high frequencies, reactance offered by the capacitor is much larger than the resistance value. Therefore, applied input signal appears at the output mostly unattenuated.
  2) At low frequencies, the capacitive reactance becomes much smaller than the resistance value. Thus, forcing the output to be near zero.
- Here, we consider 2 cases: 1) Inverting Filter & 2) Non-Inverting Filter.

Case 1: Non-Inverting Filter with gain

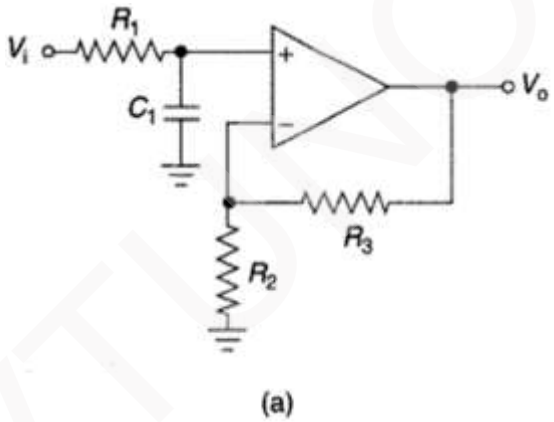


Figure 17.55a: Low-pass filter with gain

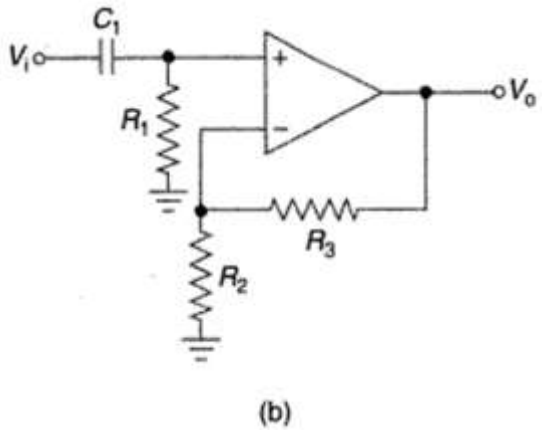


Figure 17.55b: High-pass filter with gain

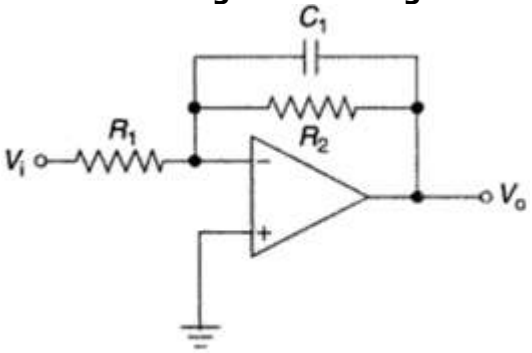
- The cut-off frequency (f_c) in both cases (Figure 17.55a & Figure 17.55b) is given by

$$f_C = \frac{1}{2\pi RC}$$

- The voltage gain (A_v) in both cases is given by

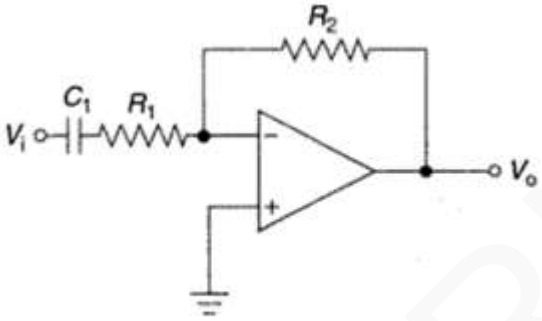
$$A_v = 1 + \frac{R_3}{R_2}$$

Case 2: Inverting Filter with gain



(a)

Figure 17.56a:Inverting Low-pass filter with gain



(b)

Figure 17.56b:Inverting High-pass filter with gain

- In case of inverting filters (Figure 17.56a & Figure 17.56b).
  The cut-off frequency (f_c) & voltage gain (A_v) is given by

$$f_C = \frac{1}{2\pi R_2 C_1} \qquad A_v = -\frac{R_2}{R_1}$$

SECOND ORDER FILTERS (BUTTERWORTH FILTER)

- Butterworth filter is also called as **maximally flat filter**.
- It offers a relatively flat pass and stop band response.

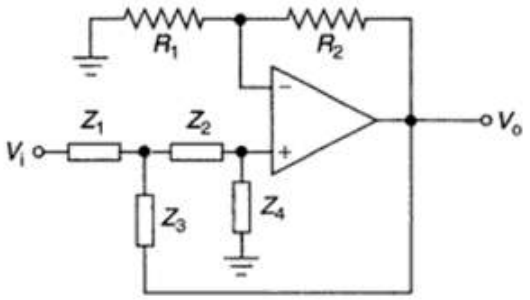


Figure 17.57: Generalized form of second-order Butterworth filter

- Here is how it works (Figure 17.57):
  1. If $Z_1 = Z_2 = R$ and $Z_3 = Z_4 = C$, we get a second-order low-pass filter.
  2. If $Z_1 = Z_2 = C$ and $Z_3 = Z_4 = R$, we get a second-order high-pass filter.
- The cut-off frequency(f_c) & pass band gain(A_v) is given by

$$f_C = \frac{1}{2\pi RC} \qquad A_v = 1 + \frac{R_2}{R_1}$$

- Here, we consider 2 types of filters: 1) Band-pass filters & 2) Band-reject filters.

1) Band-pass filters

- Band-pass filters can be formed by cascading high-pass & low-pass filter sections in series.
- These filters are simple to design and offer large bandwidth.

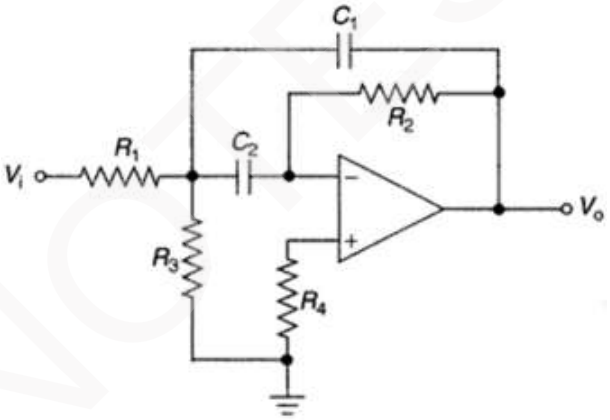


Figure 17.58: Narrow band-pass filter

- Here is how it works (Figure 17.58):
  1) At very low frequencies, C_1 and C_2 offer very high reactance. As a result, the input signal is prevented from reaching the output.
  2) At very high frequencies, the output is shorted to the inverting input, which converts the circuit to an inverting amplifier with zero gain. Again, there is no output.
- At some intermediate band of frequencies, the gain provided by the circuit offsets the loss due to potential divider R_1-R_3. The resonant frequency is given by

$$f_R = 2Q/2\pi R_2 C$$

where Q is the quality factor

- For $C_1 = C_2 = C$, the quality factor and voltage gain is given by

$$Q = \left[R_1 R_2 / 2R_3\right]^{1/2} \qquad A_v = Q/2\pi R_1 f_R C$$

2) Band-reject filters

- Band-reject filters can be implemented by summing together the outputs of the low-pass and high-pass filters.
- These filters are simple to design and have a broad reject frequency range.
- It uses a twin-T network that is connected in series with the non-inverting input of the opamp.

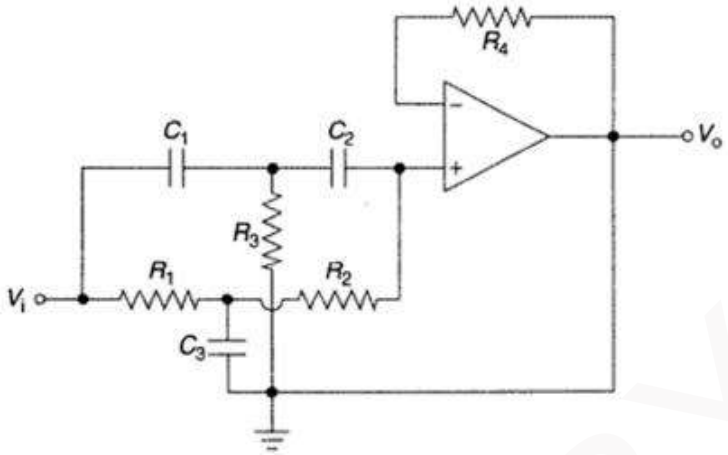


Figure 17.59: Second-order band-reject filter

- Here is how it works (Figure 17.59):
  1) Very low frequency signals find their way to the output via the low-pass filter formed by R_1-R_2-C_3.
  2) Very high frequency signals reach the output through the high-pass filter formed by C_1-C_2-R_3.
- Intermediate band of frequencies pass through both the filters, net signal reaching the non-inverting input and hence the output is zero.
- Component values are chosen by following equations:

$$R_1 = R_2 = R, R_3 = R/2$$

$$C_1 = C_2 = C, C_3 = 2C$$

$$0 \le R_4 \le (R_1 + R_2)$$

$$f_R = \frac{1}{2\pi RC}$$

Example 17.14

Refer to the first order low pass filter of figure 17.60. Determine the cut-off frequency and the gain value at four times the cut-off frequency.

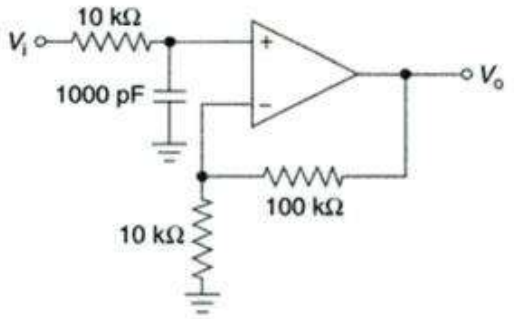


Figure 17.60: Example 17.14

Solution:

1. Cut-off frequency,
   $f_C = 1/(2\pi \times 10 \times 10^3 \times 1000 \times 10^{-12}) = 10^5/2\pi$ Hz = 15.915 kHz
2. Gain, $A_v = (1 + 100 \times 10^3)/(10 \times 10^3) = 11 = 20.827$ dB.
3. Gain at cut-off point = 20.827 – 3 = 17.827 dB.
4. Gain at frequency four times the cut-off frequency will be 12 dB below the value of mid-band gain.
5. Therefore, gain at four times the cut-off frequency = 20.827 – 12 = 8.827 dB.

Example 17.15

Figure 17.61 shows a second-order low-pass filter built around a single opamp. Calculate the values of R_1, R_2, C_1, C_2 and R_3 if the filter had a cut-off frequency of 10 kHz, Q-factor of 0.707 and input impedance not less than 10kΩ.

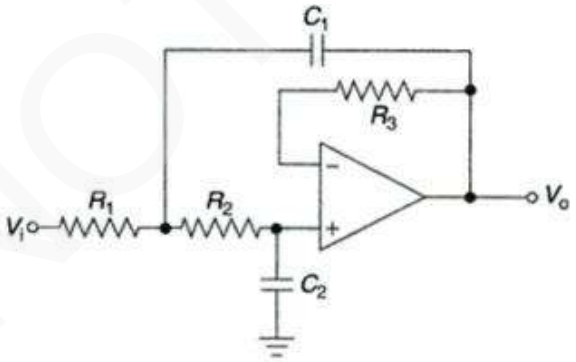


Figure 17.61: Example 17.15

Solution:

1. Q-factor is given by $Q = \left(\frac{1}{2}\right) \times \sqrt{\frac{C_1}{C_2}}$
2. For Q = 0.707, $C_1 = 2C_2$.
3. For input impedance of 10kΩ, $R_1 = 10k\Omega = R_2$, also $R_3 = R_1 + R_2$, $\therefore R_3 = 20$ kΩ
4. $f_C = \frac{1}{2\pi R\sqrt{C_1 C_2}}$, $10 \times 10^3 = \frac{1}{2\pi \times 10 \times 10^3 \times C_2 \times \sqrt{2}}$, $\therefore C_2 = 0.0011\,\mu F$
5. $C_1 = 2C_2 = 0.0022\mu F$

Example 17.16

Design an opamp based twin-T band reject filter having a notch frequency of 100kHz. Specify the small-signal bandwidth of the chosen opamp if the highest expected frequency were 1 MHz.

Solution:

1. Figure 17.62 shows the circuit.
   The notch frequency is given by $f_R=1/2\prod RC$
   where $R_1=R_2=R$, $C_1=C_2=C$, $R_3=R/2$ and $C_3=2C$.
2. Let $C_1=0.0001$ μF. This gives $R_1=1/2\prod*100*10^3*0.0001*10^{-6}=15.92$ KΩ
3. This gives $C_1=C_2=0.0001$ μF, and $C_3=0.0002$ μF
4. $R_1=R_2=15.92$ KΩ and $R_3=(15.92*10^3)/2=7.96$ KΩ
5. $R_4=R_1+R_2 = (15.92 *10^3)+(15.92*10^3)= 31.84$ KΩ

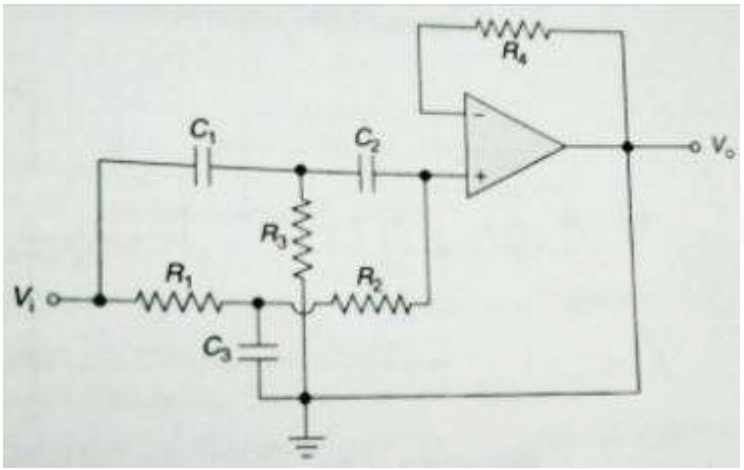


Figure 17.62: Example 17.16

NON-LINEAR AMPLIFIER

- Here, the gain value is a non-linear function of the amplitude of the signal applied at the input.
- For example, the gain may be
  → very large for weak input signals and
  → very small for large input signals.
- This implies that for a very large change in the amplitude of input signal, resultant change in amplitude of output signal is very small.

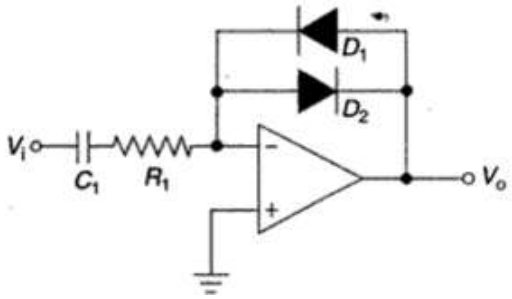


Figure 17.69: Non-linear amplifier

- For small values of input signal,
  → diodes act as open circuit and
  → gain is high due to minimum feedback (Figure 17.69).
- When the amplitude of input signal is large, diodes offer very small resistance and thus gain is low.
- Resistance R_1 decides the compression ratio.
  Higher the value of resistor R_1, lesser is the compression ratio.

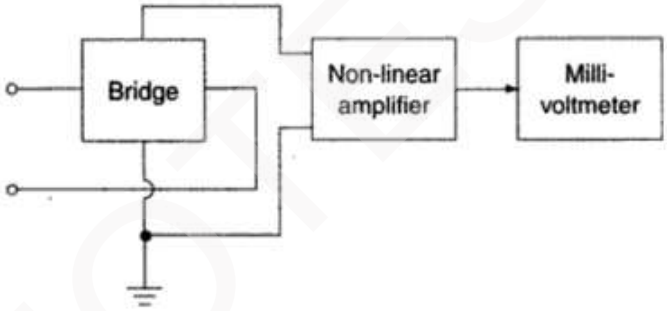


Figure 17.70: Application of non-linear amplifier in AC Bridge balance detector

- **Common Application:** AC Bridge balance detectors (Figure 17.70).
  - The output of bridge may vary over a wide range around its null point.
  - In order to achieve null, the output is usually applied to an AC milli voltmeter.
  - If the bridge output is applied to the non-linear amplifier, the output of the non linear amplifier will vary only in a small range.

RELAXATION OSCILLATOR

- Relaxation oscillator is an oscillator circuit.
- It produces a non-sinusoidal output whose time period is dependent on the charging time of a capacitor.
- The capacitor is connected as a part of the oscillator circuit.
- Here is how it works (Figure 17.71):
  - Let us assume that the output is initially in positive saturation.
  - As a result, voltage at non-inverting input of opamp is
    $$+V_{SAT} \times R_1/(R_1 + R_2)$$
  - This forces the output to stay in positive saturation as the capacitor C is initially in fully discharged state.
  - Capacitor C starts charging towards $+V_{SAT}$ through R.
  - The moment the capacitor voltage exceeds the voltage appearing at the non-inverting input, the output switches to $-V_{SAT}$.
  - The voltage appearing at non-inverting input also changes to
    $$-V_{SAT} \times R_1/(R_1 + R_2)$$
  - The capacitor starts discharging after reaching zero, it begins to discharge towards $-V_{SAT}$.
  - Again, as soon as it becomes more negative than the negative threshold appearing at non-inverting input of the opamp, the output switches back to $+V_{SAT}$.
  - The cycle repeats thereafter.
  - The output is a rectangular wave.
- The expression for time period of output waveform can be derived from the exponential charging and discharging process and is given by

$$T = 2RC \ln\left(\frac{1+B}{1-B}\right)$$

where $B = R_1/(R_1 + R_2)$

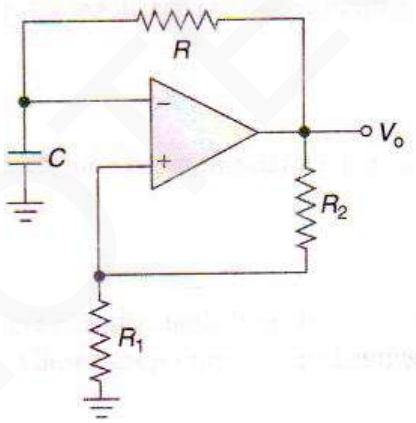


Figure 17.71: Relaxation oscillator

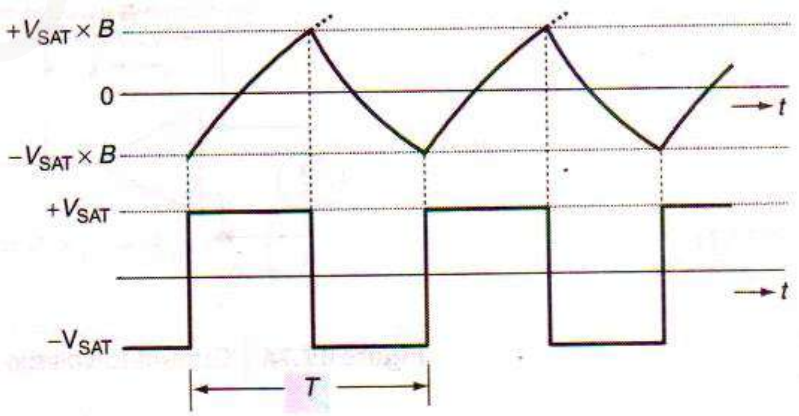


Figure 17.72: Relevant waveforms of Relaxation oscillator

Example 17.19

Refer to the relaxation oscillator circuit of figure 17.76. Determine the peak-to-peak amplitude and frequency of the square wave output given that saturation output-voltage of the opamp is +12.5 V at power supply voltages of +15V.

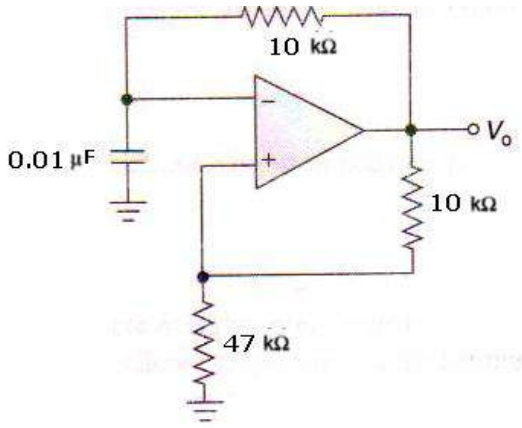


Figure 17.76: Example 17.19

Solution:

1. The feedback factor B is given by $(47*10^3)/(47*10^3+10*10^3)=0.825$
2. Time period T of the output waveform is given by $T=2RC \ln [(1+B)/(1-B)]$
3. That is, $T=2*10*10^3*0.01*10^{-6}*\ln [(1+0.825)/(1-0.825)] =0.469$ ms.
4. Therefore, f=1/0.469 kHz= 2.13 kHz
5. Peak-to-peak amplitude of output = $2V_{SAT}$= 25 V.

CURRENT-TO-VOLTAGE CONVERTER

- Current-to-voltage converter is a transimpedance amplifier (Figure 17.74).
- An ideal transimpedance amplifier makes a perfect current-to-voltage converter, as it has
  → zero input impedance &
  → zero output impedance.
- Opamp wired as transimpedence amplifier very closely approaches a perfect current-to-voltage converter.

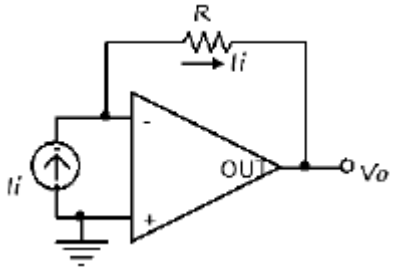


Figure 17.74: Current-to-Voltage converter

- The circuit is characterized by voltage shunt feedback with a feedback factor of unity.
- The output-voltage is given by

$$V_o = I_i \times R \times \left(\frac{A_{OL}}{1 + A_{OL}}\right),$$

For $A_{OL}>>1$, we have

$$V_o = I_i \, X \, R$$

- Closed loop input impedance is given by

$$Z_{in} = \frac{R}{1 + A_{OL}}$$

- Closed loop output impedance is given by

$$Z_o = \frac{R_o}{1 + A_{OL}}$$

where R_0 is the output impedance of the opamp.

Example 17.20

For current-to-voltage converter circuit of figure 17.77, determine output-voltage, closed loop input and output impedance given that chosen opamp has open-loop transimpedence gain of 100,000 input impedance of 1MΩ and output impedance of 100Ω.

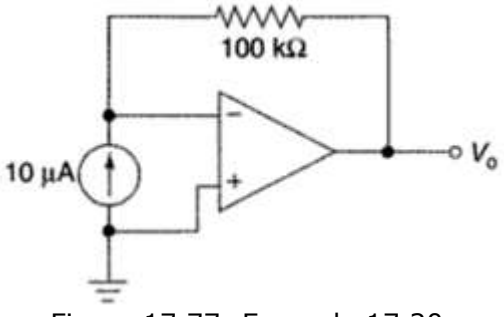


Figure 17.77: Example 17.20

Solution:

1. Output voltage = $10 \times 10^{-6} \times 100 \times 10^3 = 1$ V.
2. Closed-loop input impedance, $Z_{in} = R/(1 + A_{OL}) = 100 \times 10^3/(1 + 100{,}000) = 1\ \Omega$.
3. Closed-loop output impedance, $Z_o = R_o/(1 + A_{OL}) = 100/(1 + 100{,}000) = 0.001\ \Omega$.

VOLTAGE-TO-CURRENT CONVERTER

• Voltage-to-current converter is a transconductance amplifier (Figure 17.75).

• An ideal transconductance amplifier makes a perfect voltage-controlled current source or a voltage-to-current converter.

• Opamp wired as transconductance amplifier very closely approaches a perfect voltage-to-current converter.

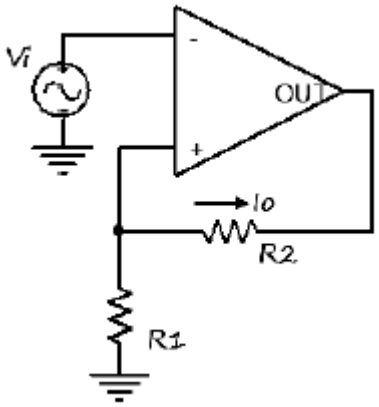


Figure 17.75: Voltage-to-Current converter

• The circuit is characterized by current series feedback.

• The output-voltage is given by

$$I_o = \frac{V_i}{R_1 + \left[\frac{(R_1 + R_2)}{A_{OL}}\right]}$$

For $A_{OL}>>1$, we have

$$I_o = \frac{V_i}{R_1}$$

• Closed loop input impedance is given by

$$Z_{in} = R_i \times \left(1 + A_{OL} \times \frac{R_1}{R_1 + R_2}\right)$$

where R_i is the input impedance of the opamp.

• Closed loop output impedance is given by

$$Z_o = R_1 \times \left(1 + A_{OL} \times \frac{R_1}{R_1 + R_2}\right)$$

MODULE 2: THE BASIC GATES

INTRODUCTION

- A logic gate is a digital circuit with 1 or more input voltages but only 1 output voltage.
- Logic gates are the fundamental building blocks of digital systems.
- By connecting the different gates in different ways, we can build circuits that perform arithmetic and other functions associated with the human brain.
- Because the circuits simulate mental processes, gates are often called logic circuits. NOT, OR & AND gates are the basic types of gates.
- The inter-connection of gates to perform a variety of logical operations is called logic design.
- The operation of a logic gate can be easily understood with the help of "truth table".
- A truth table lists all possible combinations of inputs and the corresponding outputs.

THE BASIC GATES - NOT, OR, AND

NOT GATE (INVERTER)

- It is a gate with only 1 input and a complemented output (Figure 2.1).
- 7404 IC is called a hex inverter (Figure 2.2).
- 7404 IC contains six inverters.
- After applying +5 V dc (the supply voltage for all TTL devices) to pin 14 and grounding pin 7, you can connect any inverters to other TTL devices.
- For example, if you only need one inverter, you can connect an input signal to pin 1 and take the output signal from pin 2.

Figure 2.1: (a)Truth Table (b)Inverter Symbol

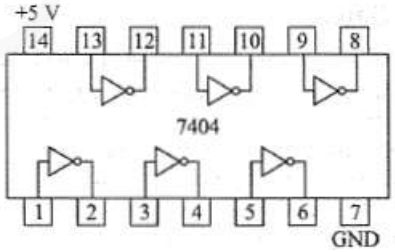


Figure 2.2: Pinout diagram of a 7404

EXAMPLE 2.1

A 1-kHz square wave drives pin 1 of a 7404 (see Fig. 2.2). What does the voltage waveform at pin 2 look like?

Solution Figure 2.3a shows what you will see on a dual-trace oscilloscope. Assuming you have set the sweep timing to get the upper waveform (pin 1), then you would see an inverted square wave on pin 2.

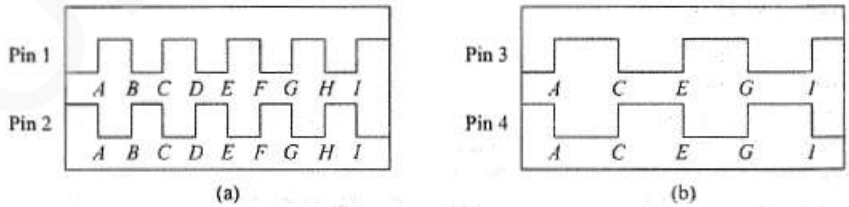


Figure 2.3: Timing Diagram of (a) Example 2.1 (b) Example 2.2

EXAMPLE 2.2

If a 500-Hz square wave drives pin 3 of a 7404(see Fig. 2.2), what is the waveform on pin 4?

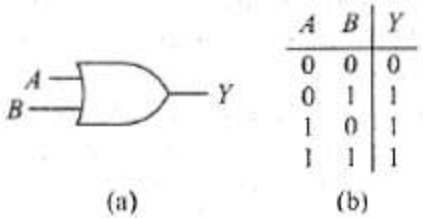
Solution Pins 3 and 4 are the input and output pins of an inverter (see Fig. 2.2). A glance at Fig. 2.3b shows the typical waveforms on the input (pin 3) and output (pin 4) of a 7404. Again, the output waveform is the complement of the input waveform. Because of two-state operation, rectangular waveforms like this are the normal shape of digital signals. Incidentally, a *timing diagram* is a picture of the input and output waveforms of a digital circuit. Examples of timing diagrams are shown in Figs. 2.3a and b.

OR GATE

- It is a gate with 2 or more inputs.
- The output is HIGH when any input is HIGH (Figure 2.4).
- In Boolean equation form, we write: Y = A + B.
- The '+' sign represents logic operation OR.

7432

- 7432 IC is a TTL quad 2-input OR gate (Figure 2.7).
- 7432 contains four 2-input OR gates inside a 14-pin DIP.
- After connecting a supply voltage of +5 V to pin 14 and a ground to pin 7, you can connect one or more of the OR gates to other TTL devices.



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

Figure 2.4: 2-Input OR Gate (a)Truth Table (b)2-Input OR Gate Symbol

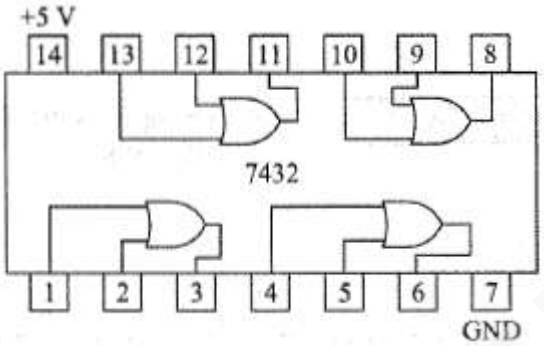


Figure 2.7: Pinout diagram of a 7432

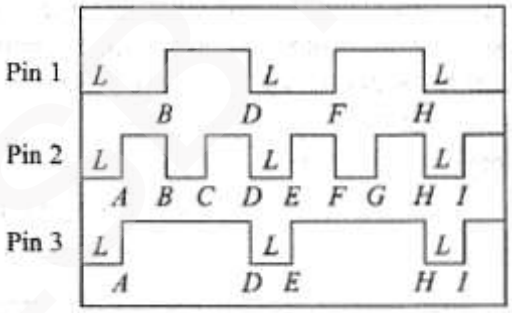
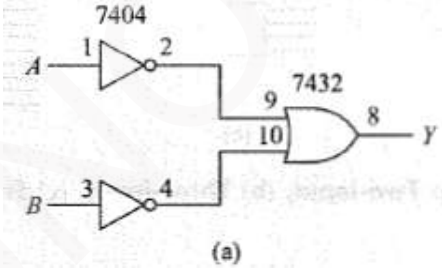


Figure 2.8: Timing diagram

EXAMPLE 2.3

Work out the truth table for Fig. 2.9a.



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

Figure 2.9: Logic circuit and truth table of Example 2.3

Solution With two input signals (*A* and *B*), four input cases are possible: low-low, low-high, high-low, and high-high. For convenience, let *L* stand for low and *H* for high. Then, the input possibilities are *LL*, *LH*, *HL*, and *HH*, as listed in Fig. 2.9b. Here is what happens for each input possibility.

CASE 1 *A* is low and *B* is low. With both input voltages in the low state, each inverter has a high output. This means that the OR gate has a high output, the first entry of Fig. 2.9b.

CASE 2 *A* is low and *B* is high. With these inputs the upper inverter has a high output, while the lower inverter has a low output. Since the OR gate still has a high input, the output *Y* is high.

CASE 3 *A* is high and *B* is low. Now, the upper inverter has a low output and the lower inverter has a high output. Again, the OR gate produces a high output, so that *Y* is high.

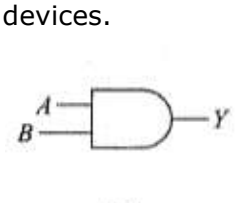
CASE 4 *A* is high and *B* is high. With both inputs high, each inverter has a low output. This time, the OR gate has all inputs in the low state, so that *Y* is low, as shown by the final entry of Fig. 2.9b.

AND GATE

- It is a gate with 2 or more inputs (Figure 2.10).
- The output is HIGH only when all inputs are HIGH.
- In Boolean equation form, we write: Y = A.B
- The '.' sign here represents logic operation AND.

7408

- 7408 IC is a TTL quad 2-input AND gate (Figure 2.13).
- 7408 contains four 2-input AND gates inside a 14-pin DIP.
- After connecting a supply voltage of +5 V to pin 14 and a ground to pin 7, you can connect one or more of the AND gates to other TTL devices.



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

(a) (b)

Figure 2.10: (a)Truth Table (b)2-Input AND Gate Symbol

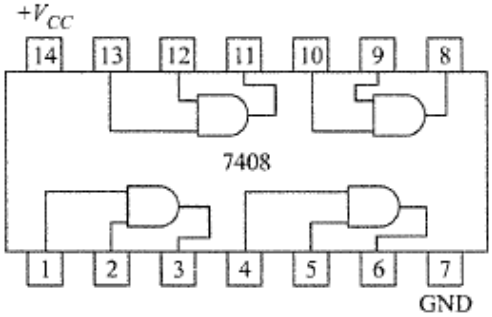


Figure 2.13: Pinout diagram of a 7408

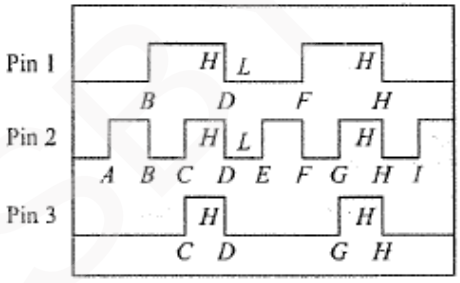
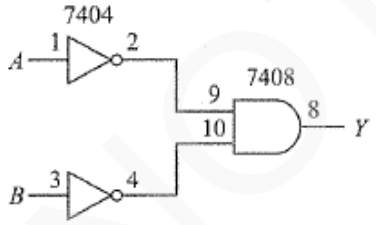


Figure 2.14: Timing diagram

B

EXAMPLE 2.4

Work out the truth table for Fig. 2.15a.



(a)

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

(b)

Figure 2.15: logic circuit and truth table of Example 2.4

Solution We get the final truth table here in slightly different way. Consider, one logic gate at a time as shown in Fig. 2.15b. The NOT gate connected to A gives A' at its output and is shown in column 3. The NOT gate connected to B gives B' at its output and is shown in column 3. Finally, the 4th column shows OR operation on column 3 and 4 to give the final output Y. Here is what happens for each input possibility.

CASE 1 A is low and B is low. With both input voltages in the low state, each inverter has a high output. This means the AND gate has a high output, the first entry of Fig. 2.16.

CASE 2 A is low and B is high. With these inputs the upper inverter has a high output, while the lower inverter has a low output. Since the AND gate produces a low output, Y is low.

CASE 3 A is high and B is low. Now, the upper inverter has a low output and the lower inverter has a high output. Again, the AND gate produces a low output, so Y is low.

CASE 4 A is high and B is high. With both inputs high, each inverter has a low output. Again, the AND gate has a low output, as shown by the final entry of Fig. 2.16.

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

Note that input-output relations described in Fig. 2.15b and Fig. 2.16 are same.

EXAMPLE 2.5

What is the Boolean equation for the logic circuit of Fig. 2.16?

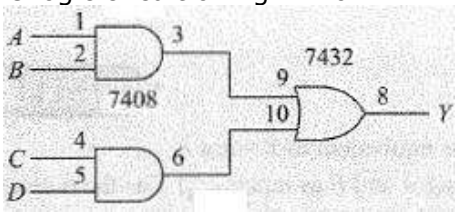


Figure 2.16: Example 2.5

Solution This circuit is called an AND-OR network because input AND gates drive an output OR gate. The intermediate outputs are

$$Y_3 = AB \qquad Y_6 = CD$$

The final output is

$$Y_8 = Y_3 + Y_6$$
$$Y = AB + CD$$

An equation in this form is referred to as a *sum-of-products equation*. AND-OR networks always produce sum-of-products equations.

EXAMPLE 2.6

Write the Boolean equation for Fig. 2.17.

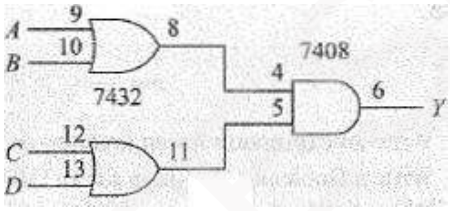


Figure 2.17: Example 2.6

Solution This logic circuit is called an OR-AND network because input OR gates drive an output AND gate. The intermediate outputs are $Y_8 = A + B$ and $Y_{11} = C + D$.

The final output is

$$Y_6 = Y_8 Y_{11}$$

or

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

EXAMPLE 2.7

What is the logic circuit whose Boolean equation is

Y=A'BC+AB'C

Solution This is a sum-of-products equation with some of the inputs in complemented form. Figure 2.18a shows an AND-OR circuit with the foregoing Boolean equation. The upper AND gate produces a logical product of

$$Y_{12} = \bar{A}BC$$

The lower AND gate produces

$$Y_6 = A\bar{B}C$$

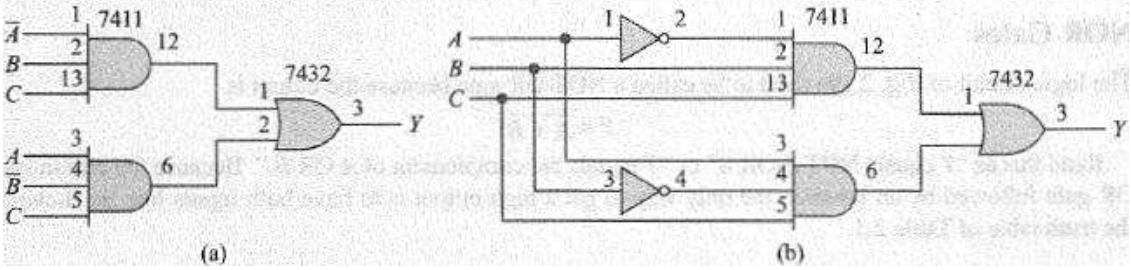


Figure 2.18: (a) Intermediate, (b) Final logic circuit of Example 2.7

UNIVERSAL LOGIC GATES - NOR, NAND

• Any logic function can be realized using only NAND gates or only NOR gates. For this reason, AND & NOR gates are called **universal gates**.

NOR GATE

• This represents an OR gate followed by an inverter (Figure 2.19a).

• 7402 IC is a quad 2-input NOR gate in a 14-pin DIP (Figure 2.19d).

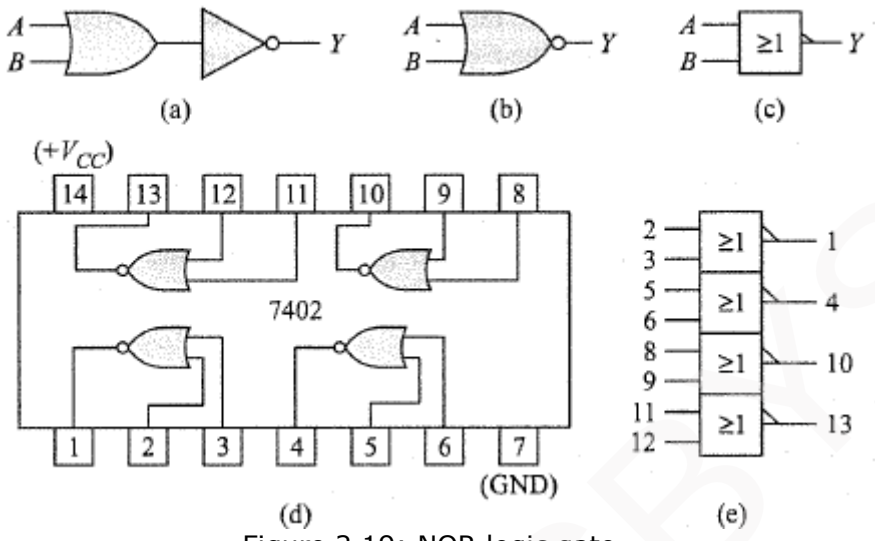


Figure 2.19: NOR logic gate

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

Truth Table 2.1: NOR gate

Bubbled AND Gate

• Bubbles on the inputs are a reminder of inversion that takes place before AND operation (Fig 2.20).

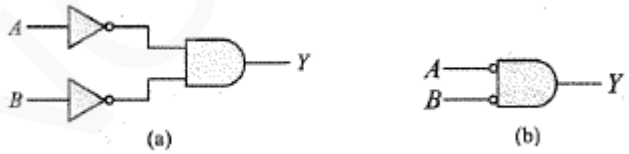


Figure 2.20: (a) AND gate with inverted inputs, (b) Equivalent symbol

De Morgan's First Theorem

• It says the complement of a sum equals the product of the complements.

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

Universality of NOR Gate

• Realization of other gates using only NAND gates (Figure 2.21).

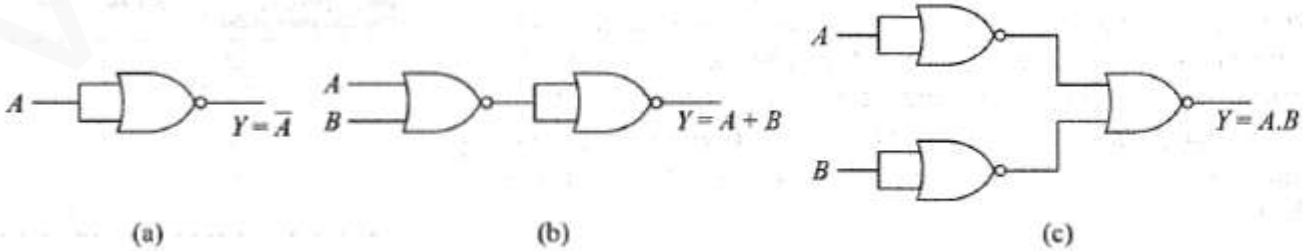


Figure 2.21: Universality of NOR gate (a)NOT from NOR (b)OR from NOR (c)AND from NOR

EXAMPLE 2.8

A 7402 is a quad 2-input NOR gate. This TTL IC has four 2-input NOR gates in a 14-pin DIP. What is the Boolean equation for the output of Fig. 2.22a?

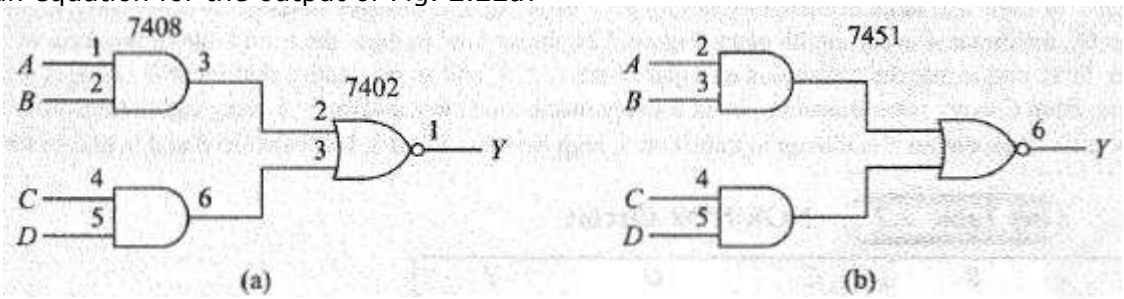


Figure 2.22: AND-OR-INVERT network

Solution The AND gates produce AB and CD. These are ORed to get $AB + CD$. The final inversion gives

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

The circuit of Fig. 2.22a is known as an AND-OR-INVERT network because it starts with ANDing, follows with ORing, and ends with INVERTing.

The AND-OR-INVERT network is available as a separate TTL gate. For instance, the 7451 is a dual 2-input 2-wide AND-OR-INVERT gate, meaning two networks like Fig. 2.22a in a single 14-pin TTL package. Appendix 3 shows the pinout diagram. Figure 2.22b shows how we can use half of a 7451 to produce the same output as the circuit of Fig. 2.22a.

EXAMPLE 2.190

Prove that Fig. 2.23c is logically equivalent to Fig. 2.23a.

Solution De Morgan's first theorem says we can replace the final NOR gate of Fig. 2.23a by a bubbled AND gate to get the equivalent circuit of Fig. 2.23b. If you invert a signal twice, you get the original signal back again. Put another way, double inversion has no effect on the logic state; double invert a low and you still have a low; double-invert a high and you still have a high. Therefore, each double inversion in Fig. 2.23b (a pair of bubbles on the same signal line) cancels out, leaving the simplified circuit of Fig. 2.23c. Therefore, Fig. 2.23a and Fig. 2.23c are equivalent or interchangeable.

Why would anyone want to replace Fig. 2.23a by 2.23c? Suppose your shelves are full of AND gates and OR gates. If you have just run out of NOR gates and you are trying to build a NOR-NOR network like Fig. 2.23a, you can connect the OR-AND circuit of Fig. 2.23c because it produces the same output as the original circuit. In general, this idea applies to any circuit that you can rearrange with De Morgan's theorem. You can build whichever equivalent circuit is convenient.

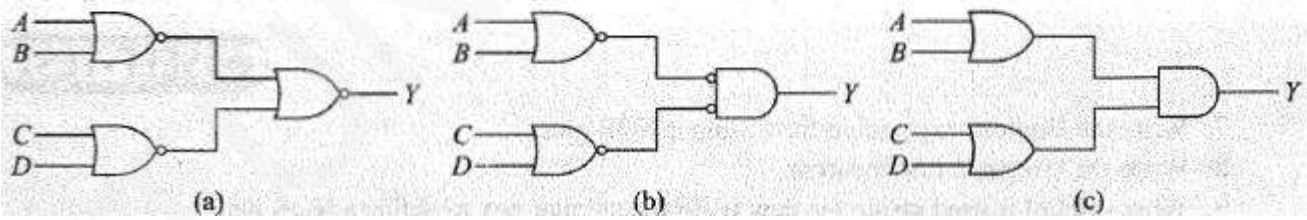


Figure 2.23: Equivalence among logic circuits: Example 2.9

EXAMPLE 2.10

What is the truth table for the NOR-NOR circuit of Fig. 2.23a?

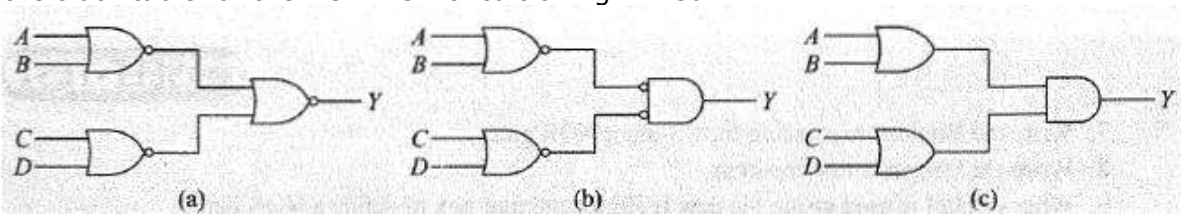


Figure 2.23: Equivalence among logic circuits: Example 2.9

Solution First, realize that the truth table of Fig. 2.23a is the same for 2.23b and 2.23c because all circuits are equivalent. Therefore, we can analyze whichever circuit we want to. Figure 2.23c is the easiest for most people to analyze, so let us use it in the following discussion.

Table 2.2 lists every possibility starting with all inputs low and progressing to all inputs high. Notice that the total number of entries equals 2^4 or 16. By analyzing each input possibility, we can work out the corresponding output. For instance, when all inputs are low in Fig. 2.23c. both OR gates have low outputs, so the AND gate produces a low output. This is the first entry of Table 2.2. Proceeding like this, we can determine the output for the remaining possibilities and arrive at all the entries shown in Table 2.2.

A	*B*	*C*	*D*	*Y*
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Table 2.2

EXAMPLE 2.11

Convert Table 2.2 into a timing diagram.

Solution In Table 2.2, input *D* changes states for each entry, input *C* changes states every other entry, input *B* every fourth entry, and input *A* every eighth entry. Figure 2.24 shows how to draw the truth table in the form of a timing diagram. First, notice that the transitions on input *D* are 1, 2, 3, and so on. Notice that input *D* changes states each transition, input *C* every other transition, input *B* every fourth transition, and input *A* every eighth transition. To agree with the truth table, output *Y* is low up to transition 5, high between 5 and 8, low between 8 and 9, and so forth.

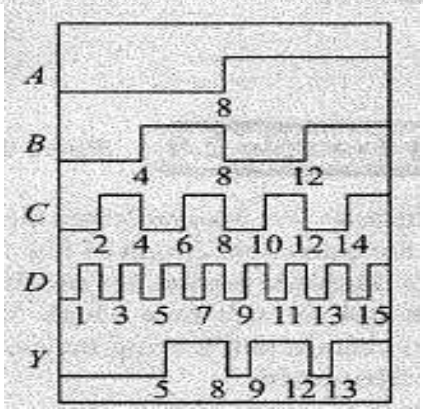


Figure 2.24: timing diagram

NAND GATE

- This represents an AND gate followed by an inverter (Figure 2.25a).
- 7400 is a quad 2-input NAND gate in a 14-pin DIP (Figure 2.25d).

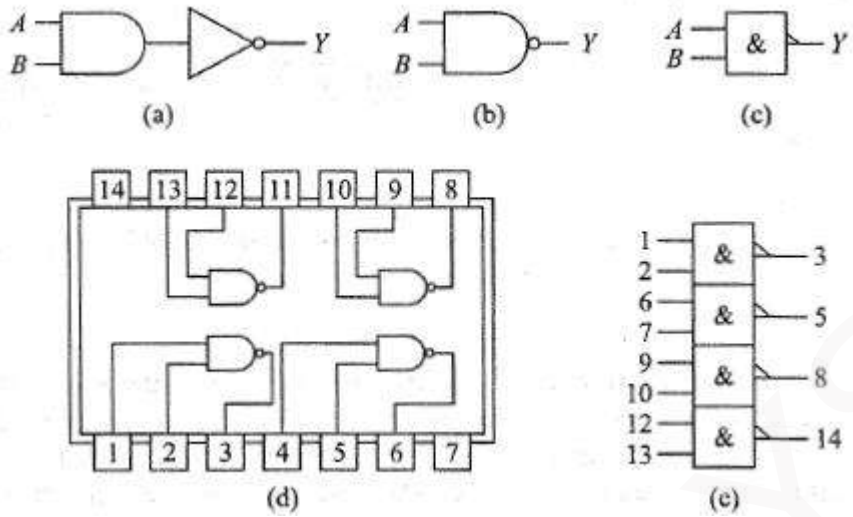


Figure 2.25: NAND logic gate

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

Truth Table 2.3: NAND gate

Bubbled OR Gate

- Bubbles on the inputs are a reminder of inversion that takes place before OR operation (Figure 2.26).

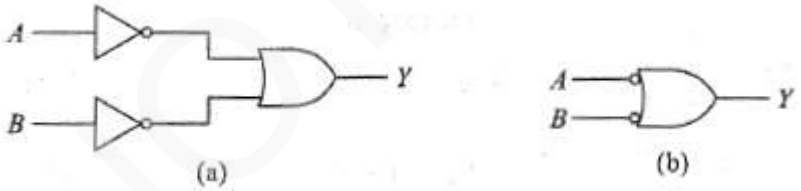


Figure 2.26: (a) OR gate with inverted inputs, (b) Equivalent symbol

De Morgan's Second Theorem

- It says the complement of a product equals the sum of the complements.

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

Universality of NAND Gate

- Realization of other gates using only NOR gates (Figure 2.27).

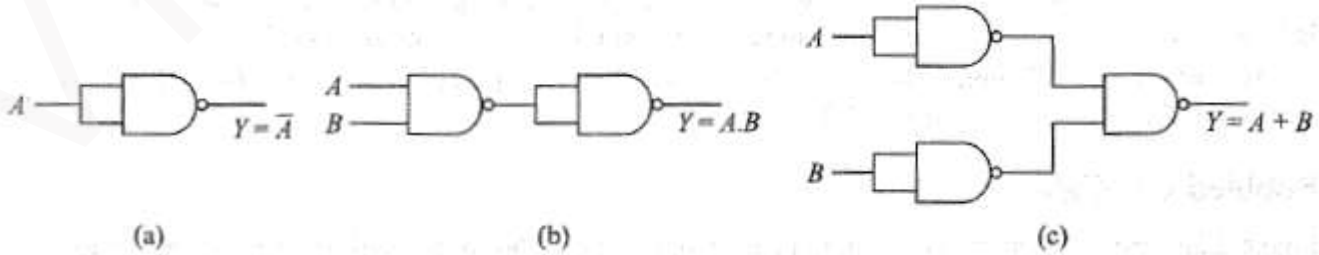


Figure 2.27: Universality of NAND gate (a)NOT from NAND (b)OR from NAND (c)AND from NAND

EXAMPLE 2.12

Realize Y=AB+C' using only one type of gate.

Solution The function to be realized involves AND, NOT and OR operations. We can realize this expression using universal logic gate.

In Method-1, we realize it using NOR gate. We realize individual logic operations like AND, NOT and OR as depicted in Fig. 2.21. The solution is given in Fig. 2.41.

In Method-2, we realize it using NAND gate. We realize individual logic operations like AND, NOT and OR as depicted in Fig. 2.27. The solution is given in Fig. 2.42.

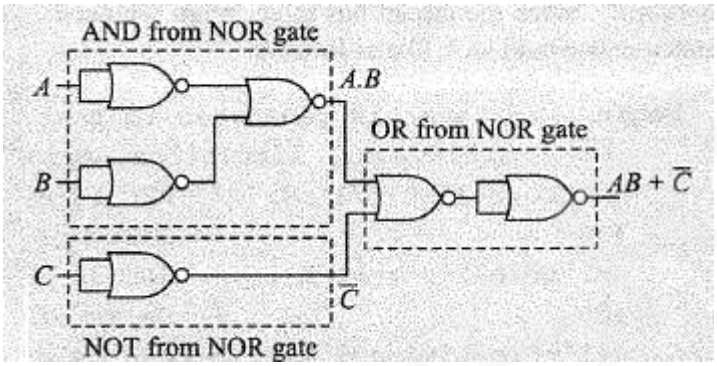


Figure 2.41: Realization of Y=AB+C' using only NOR

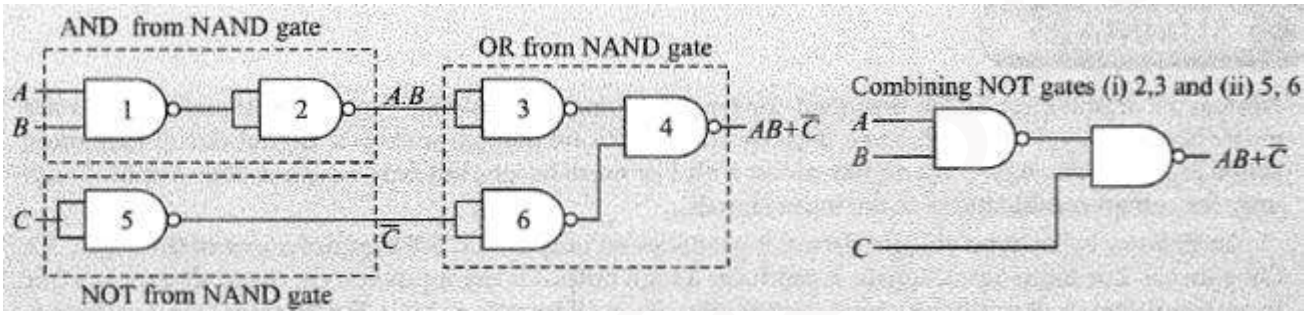


Figure 2.42: Realization of Y=AB+C' using only NAND gate

TTL NAND Gates

• The NAND gate is the backbone of the 7400 TTL series because most devices in this family are derived from it.

• Because of its central role in TTL technology, the NAND gate has become the least expensive and most widely used TTL gate.

• Furthermore, NAND gate is available in more configurations than other gates, as shown in Table 2.4.

• Notice that the NAND gate is available as a 2-, 3-, 4-, or 8-input gate. The other gates have fewer configurations, with the OR gate available only in 2-input form.

Type	*Quad 2-Input*	*Triple 3-Input*	*Dual 4-Input*	*Single 8-Input*
NAND	7400	7410	7420	7430
NOR	7402	7427	7425	
AND	7408	7411	7421	
OR	7432			

Table 2.4

EXAMPLE 2.13

Prove that Fig. 2.29c is logically equivalent to Fig. 2.29a

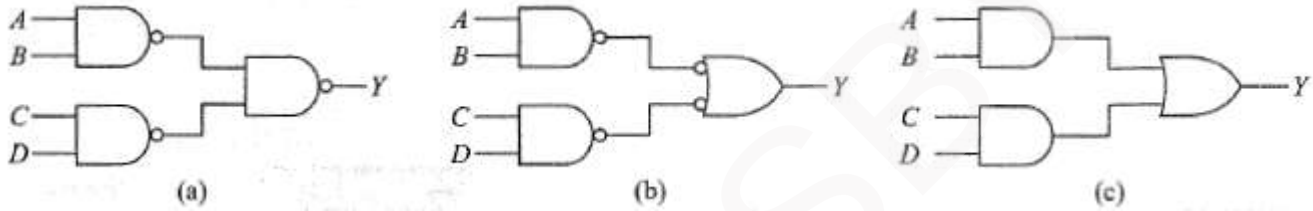


Figure 2.29: Equivalence of logic gates: Example 2.13

Solution De Morgan's second theorem says we can replace the final NAND gate of Fig. 2.29a by a bubbled OR gate to get the equivalent circuit of Fig. 2.29b. Each double inversion in Fig. 2.29b cancels out, leaving the simplified circuit of Fig. 2.29c. Therefore, Figs. 2.29a and 2.29c are equivalent.

Incidentally, most people find Fig. 2.29b easy to analyze because they learn to ignore the double inversions and see only the simplified AND-OR circuit of Fig. 2.29c. For this reason, if you build a NAND-NAND Network like Fig. 2.29a, you can draw it like Fig. 2.29b. Anyone who sees Fig. 2.29b on a schematic diagram will know it is two input NAND gates driving an output NAND gate. Furthermore, when troubleshooting the circuit, they can ignore the bubbles and visualize the easy-to-analyze AND-OR circuit of Fig. 2.29c.

Table 2.5 gives information of the IC numbers along with their functionality.

Device Number	*Description*
7400	Quad 2-input NAND gates
7402	Quad 2-input NOR gates
7404	Hex inverter
7408	Quad 2-input AND gates
7410	Triple 3-input NAND gates
7411	Triple 3-input AND gates
7420	Dual 4-input NAND gates
7421	Dual 4-input AND gates
7425	Dual 4-input NOR gates
7427	Triple 3-input NOR gates
7430	8-input NAND gate
7486	Quad 2-input XOR gates

Table 2.5: Standard TTL

EXAMPLE 2.14

What is the truth table for the NAND-NAND circuit of Fig. 2.29a?

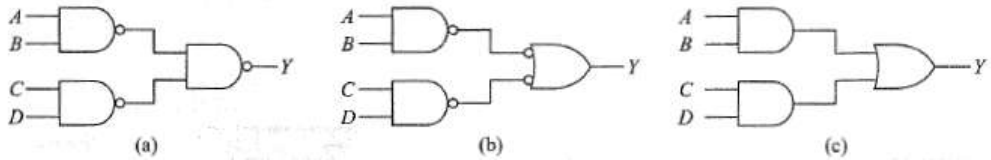


Figure 2.29: Equivalence of logic gates: Example 2.14

Solution Let us analyze the equivalent circuit of Fig. 2.29c because it is simpler to work with. Table 2.5 lists every possibility starting with all inputs low and progressing to all inputs high. By analyzing each input possibility, we can determine the resulting output. For instance, when all inputs are low in Fig. 2.29c, both AND gates have low outputs, so the OR gate produces a low output. This is the first entry of Table 2.5. Proceeding like this, we can arrive at the output for the remaining possibilities of Table 2.5.

A	B	C	D	Y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Table 2.5

EXAMPLE 2.15

Show a timing diagram for the NAND-NAND circuit of Fig. 2.29a.

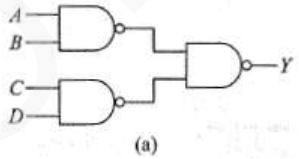


Figure 2.29: Example 2.14

Solution All you have to do is convert the low-high states of Table 2.5 into low-high waveforms like Fig. 2.30. First, notice that the transitions on input D are numbered 1, 2, 3, and so on. Input D changes states each transition, input C every other transition, input B every fourth transition, and input A every eighth transition. To agree with the truth table, output Y is low up to transition 3, high between 3 and 4, low between 4 and 7, and so forth.

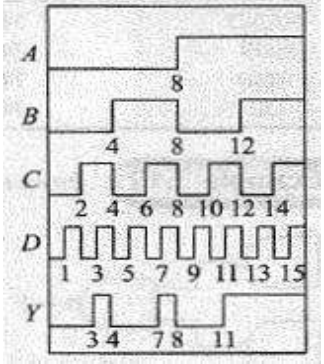


Figure 2.30: Timing diagram

POSITIVE AND NEGATIVE LOGIC

• We know that, in binary logic, two voltage levels represent the two binary digits, 1 and 0.

A	*B*	*Y*
Low	Low	Low
Low	High	High
High	Low	High
High	High	High

Table 2.8

• In positive logic, the lower voltage level is assigned binary 0 & higher voltage level is assigned binary 1. So, we can convert table 2.8 to table 2.9.

HIGH=1
LOW=0

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

Table 2.9

• In negative logic, the lower voltage level is assigned binary 1 & higher voltage level is assigned binary 0. So, we can convert table 2.8 to table 2.10.

HIGH=0
LOW=1

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

Table 2.10

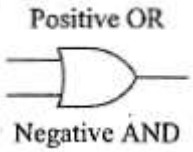


Figure 2.35: Meaning of symbol depends on whether you use positive or negative logic

Gate	*Definition*
Positive OR/negative AND	Output is high if any input is high.
Positive AND/negative OR	Output is high when all inputs are high.
Positive NOR/negative NAND	Output is low if any input is high.
Positive NAND/negative NOR	Output is low when all inputs are high.

Table 2.9: voltage definition for basic gates

ASSERTION LEVEL

• To activate, if an input line has a bubble on it, you assert the input by making it low. If there is no bubble, you assert the input by making it high. This is called as Assertion level.
• It means that you draw chips with the kind of input that causes something to happen, or with the kind of output that indicates something has happened.
• If a low input signal turns on a chip, you show a bubble on that input.
• If a low output is a sign of chip action, you draw a bubble on that output. Once you get used to assertion-level logic, you may prefer drawing logic circuits this way.

What happens when the inputs are asserted?

• An input is asserted when it is active.
• This means it may be low or high, depending on whether it is an active-low or active-high input.
• For instance, given a positive AND gate, all inputs must be asserted (high) to get a high output.
• As another example, the STROBE input of a TTL multiplexer must be asserted (low) to turn on the multiplexer.
• In short, you can equate the word assert with activate.
• You assert, or activate, the inputs of a gate or device to get something to happen.

EXAMPLE 2.14

(a) The number stored in a register may be zero (all bits low). Show how to detect this condition.
(b) What change in (a) will detect presence of the word 10110101 in the 8-bit register?

Solution

(a) Figure 2.36 shows a design using assertion-level logic. The bits go to a bubbled AND gate (the same as positive NOR gate). When all the bits are low, output ZERO is high. Because of the inverter, the final output $\overline{\text{ZERO}}$ is active-low. Therefore, when the sum is zero, $\overline{\text{ZERO}}$ is negative true.

(b) Some of the bubbles at the input of the bubbled AND gate need to be removed. These are for locations in the code word where '1' is present, specifically S_7, S_5, S_4, S_2 and S_0.

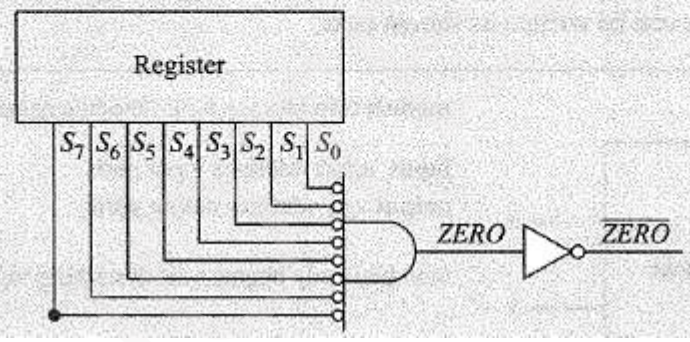


Figure 2.36: Assertion-level logic diagram ,showing the detection of zero and minus accumulator contents

HDL (HARDWARE DESCRIPTION LANGUAGE)

• This is textual description of a digital circuit.

Advantages of HDL

1) To describe large complex design requiring hundreds of logic gates in a convenient manner.
2) To use software test-bench to detect functional error and correct it (called **simulation**) and
3) To get hardware implementation details (called **synthesis**).

• Currently, there are 2 widely used HDLs:

1) Verilog &
2) VHDL (Very high speed integrated circuit Hardware Description Language).

• Verilog is considered simpler of the two and is more popular.

VERILOG HDL

• This describes a digital system as a set of modules.

• In a digital circuit, there are a set of inputs and a set of outputs which are called as ports.

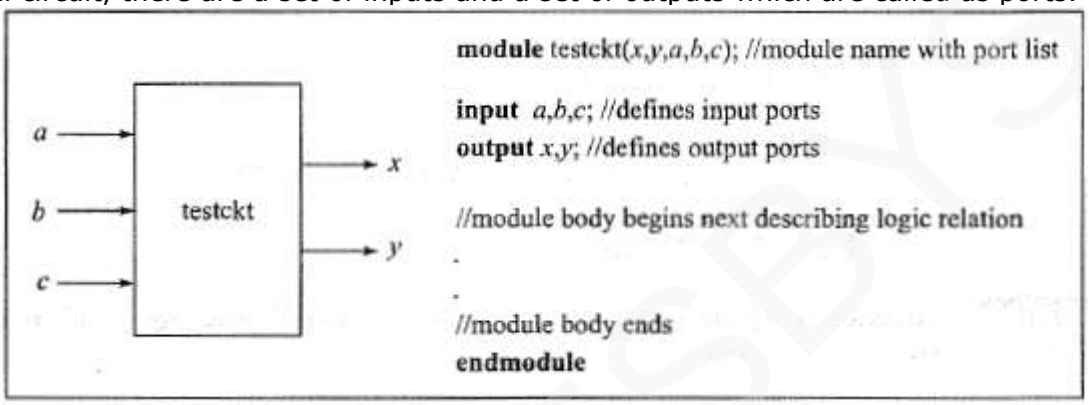


Figure 2.37: Input/output definition in Verilog HDL for logic circuit described within black-box testckt

Describing input/output

• **module** and **endmodule** are the keywords.

• Module describes a design-entity with a name or identifier selected by user (here, testckt) followed by input-output port-list.

• The symbol '//' is used
→ to put comments &
→ to improve readability for a human.

• The module-body describes the logic within the black box which
→ acts on the inputs a, b, c and
→ generates output x, y.

• Semicolon ';' is used to indicate end the statement.

Writing module body

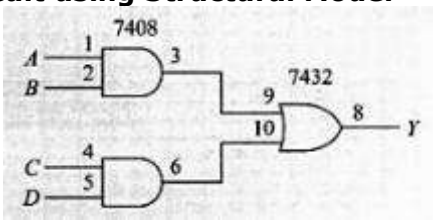
• There are 3 different models of writing module body in Verilog HDL. They are:
1) Structural model
2) Data flow modeling &
3) Behavioral modeling.

• Each model has its own advantage and suited for certain kind of design.

Write a verilog for following gate using Structural Model


```
module or_gate(A,B,Y);
  input A,B;          //defines two input port
  output Y;           //define one output port
  or g1(Y,A,B);       //represents OR gate
endmodule
```

Write a verilog for following circuit using Structural Model




```
module fig2_24(A,B,C,D,Y)
  input A,B,C,D;
  output Y;
  wire op1,op2;          //internal connections
  and g1(op1,A,B);       //g1 represents upper AND gate
  and g2(op2,C,D);      //g2 represents lower AND gate
  or g3(Y,op1,op2);      //g3 represents the OR gate
endmodule
```

PREPARATION OF TEST BENCH

• Here, we write a verilog code for simulating a OR gate.
• The test bench creates an input in the form of a timing waveform and passes this to OR gate module through a function or procedural call.
• To generate timing waveform, we use time delay available in the form of #n where n=number in decimal that gives delay in nanoseconds.
• Input values to a variable can be provided through syntax m'tn
where m=number of digits,
t=type of number and
n=value to be provided.
• The keyword 'reg' is used to hold value of a data object in a procedural assignment.
• The keyword 'initial' ensures sequential execution of codes following it, but once.
• The keyword 'always' is used for sequential execution but for infinite time.

```
module testor; //Simulation module given a name testor
reg A,B;       //Storage of data for passing it to module or_gate
wire x;
or_gate org(A,B,x); //circuit is instantiated with the name, or_gate
initial    // Starts simulation
  begin   /* Input is generated to test the circuit through following
    statements, simulation begins*/
A=1'b0;B=1'b0; /* 1'b0 signifies on binary digit with a value 0, AB is
assigned 00*/
    #20                 // Delay of 20 ns
    A=1'b0;B=1'b1;  //After 20ns AB=01
    #20          // Another Delay of 20 ns
    A=1'b1;B=1'b0; //After 40ns from start point  AB=10
    #20
    A=1'b1;B=1'b1; //After 60ns from start point  AB=11
    #20 $finish;// the simulation terminates after 80 ns
  end
endmodule

module or_gate(A,B,x); // OR gate used as a procedure in simulation
    input A,B;    // defines two input port
    output x;     // defines one output port
    or #(20) g1(x,A,B);  /*Gate declaration with a gate delay of 20ns,
     output is effected after 20 ns*/
endmodule
```

Execution of above Verilog code generates following timing diagram.

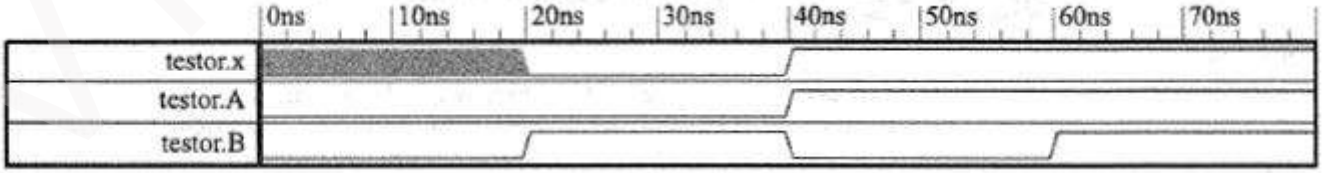


Figure 2.39: Verilog simulation of 2 input OR gate with 20ns gate delay

MODULE 2 (CONT.): COMBINATIONAL LOGIC CIRCUITS

SUM OF PRODUCTS METHOD

- The fundamental products are also called minterms (Figure: 3.3 & 3.4).
- Product-terms are represented as follows (Table: 3.1):

  A'B → m_0
  A'B → m_1
  AB' → m_2
  AB → m_3

- For 'n' variable, there can be 2^n number of minterms.

  Example: For 2 variable, there are 4 minterms.
  For 3 variable, there are 8 minterms (Table: 3.2).

A	B	Fundamental Product
0	0	$\bar{A}\bar{B}$
0	1	$\bar{A}B$
1	0	$A\bar{B}$
1	1	AB

Table 3.1: Fundamental Products for two inputs

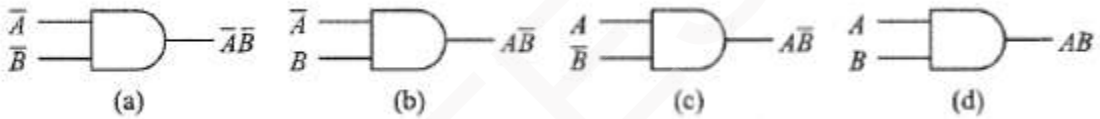


Figure 3.3: ANDing two variables and their complements

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

Table 3.2: Fundamental Products for three inputs

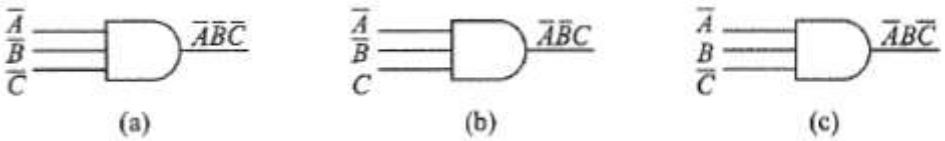


Figure 3.4: ANDing three variables and their complements

SUM OF PRODUCTS EQUATION

• **Sum-of-products equation** means the logical sum of fundamental products that produce output 1s in the truth table.

• Each product-term is called **minterm**.

For example, A.B, A.B.C, A.B.C.D etc

• Here, we have to locate output 1 in the truth table and write down the minterm.

• Consider truth table: Table 3.4.

- First output 1 appears for an input A=0, B=1 and C=1. The corresponding minterm is A'BC.
- Second output 1 appears for A=1, B=0 and C=1. The corresponding minterm is AB'C.
- Third output 1 appears for A=1, B=1 and C=0. The corresponding minterm is ABC'.
- Fourth output 1 appears for A=1, B=1 and C=1. The corresponding minterm is ABC.

• Y=F(A,B,C) means Y is a function of 3 boolean variables A,B and C.

• To get the sum of products equation, we have to OR the minterms.

Y=F(A,B,C)=A'BC+AB'C+ABC'+ABC

Y=F(A,B,C)=$m_3+m_5+m_6+m_7$ where minterm is denoted by m_i

Y=F(A,B,C)=Σm(3,5,6,7) where ∑ denotes summation i.e. OR operation

• This kind of representation of a truth table is also known as **canonical sum form**.

• The logic circuit for Y is shown in Figure: 3.5.

• The corresponding logic circuit is either

→ AND-OR circuit or

→ NAND-NAND circuit.

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

Table 3.4: Fundamental Products

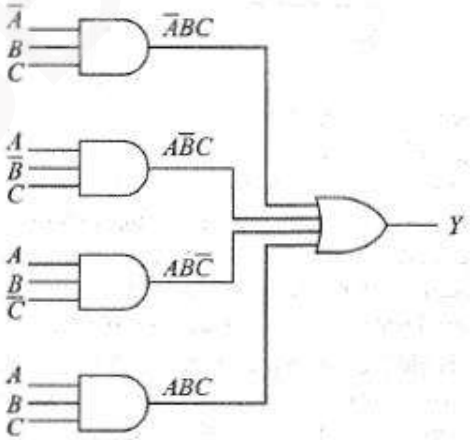


Figure 3.5: AND-OR solution

EXAMPLE 3.4

Suppose a three-valuable truth table has a high output for these input conditions: 000, 010, 100, and 110. What is the sum-of-products circuit?

Solution Here are the fundamental products:

$000 : \overline{A}\,\overline{B}\,\overline{C}$

$010 : \overline{A}B\overline{C}$

$100 : A\overline{B}\,\overline{C}$

$110 : AB\overline{C}$

When you OR these products, you get

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

The circuit of Fig. 3.6 will work if we reconnect the input lines to the bus as follows:

$\overline{A}$: pins 1 and 3

$\overline{B}$: pins 2 and 10

$\overline{C}$: pins 13, 5, 11, and 13

A : pins 9 and I

B : pins 4 and 2

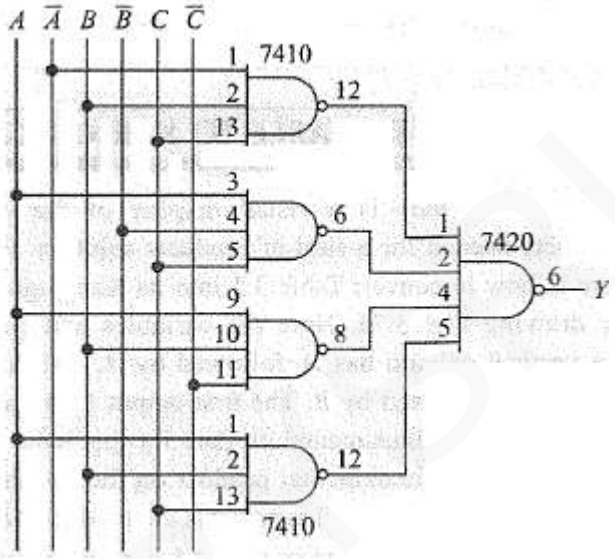


Figure 3.6: Combinational logic circuit

EXAMPLE 3.5

Simplify the following Boolean equation and describe the logic circuit.

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

Solution The Boolean equation is

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

Since $\overline{C}$ is common to each term, factor as follows:

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

Again, factor to get

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

Now, simplify the foregoing as follows:

$$Y = [\overline{A}\,(1) + A(1)]\overline{C} = (\overline{A} + A)\overline{C}$$

or

$$Y = \overline{C}$$

This final equation means that you don't even need a logic circuit. All you need is a wire connecting input $\overline{C}$ to output Y.

TRUTH TABLE TO KMAP (KMAP)

• **Kmap** is a drawing that shows all the fundamental products and the corresponding output values of a truth table (Figure 3.7).

• In Table 3.5,

i) The first output 1 appears for A=1 and B=0.

The minterm for this input condition is AB'.

So, enter 1 into cell of kmap identified by row A and column B'.

ii) Similarly, enter 1 into cell identified by row A and column B.

iii) Finally, enter 0s in the remaining cells.

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

Table 3.5

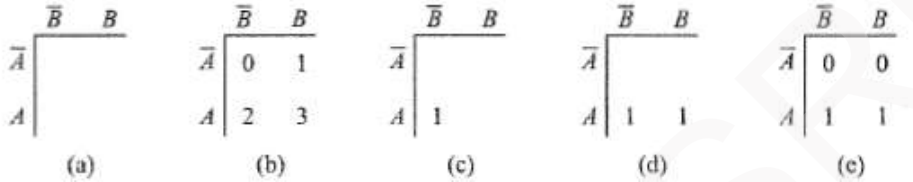


Figure 3.7: Constructing a kmap

Three-Variable Maps

• Kmap for logic equation Y=F(A,B,C)=∑m(2,6,7) is shown in Figure 3.8

• Table 3.6 gives truth table for given logic equation.

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

Table 3.6

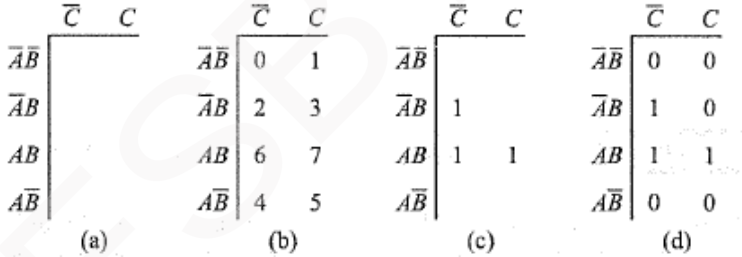


Figure 3.8: Three variable kmap

Four-Variable Maps

• Kmap for logic equation Y=F(A,B,C,D)=∑m(2,6,7,14) is shown in Figure 3.9

• Table 3.7 gives truth table for given logic equation.

A	B	C	D	Y
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

Table 3.7

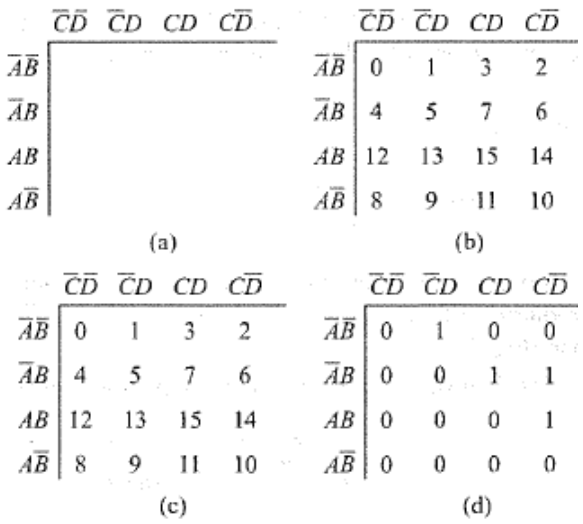


Figure 3.9: Four variable kmap

ENTERED VARIABLE MAP (EVM)

- Entered variable map is an alternative to Kmap.
- Here, one of the input variables is placed as output variable inside the kmap. This is done separately noting how an input variable is related to the output variable.
- This reduces the kmap size by 1 degree.
- This technique is particularly useful for mapping problems with more than 4 input variables.
- Entered variable map for Table: 3.6 is constructed as follows.
    1) For AB=00, we find Y=0 and is not dependent on C (Figure: 3.10).
    2) For AB=01, we find Y is complement of C thus we can write Y=C'.
    3) For AB=10, Y=0.
    4) For AB=11, Y=1.

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

Table 3.6

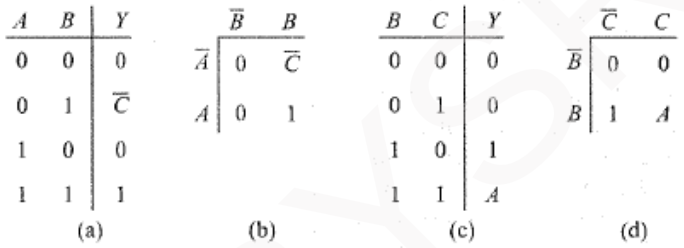


Figure 3.10: Entered variable map

PAIRS, QUADS AND OCTETS

- **Pair** means two horizontally or vertically adjacent 1s on a Kmap.
  It eliminates one variable and its complement (Figure: 3.12).
  For ex, A & A' is eliminated.
- **Quad** means four horizontal, vertical, or rectangular 1s on a Kmap.
  It eliminates 2 variables and their complements (Figure: 3.13).
  For ex, A, B, A' & B' are eliminated.
- **Octet** means eight adjacent 1s in a 2 x 4 shape on a Kmap.
  It eliminates three variables and their complements (Figure: 3.14).
  For ex, A, B, C, A', B' & C' are eliminated.

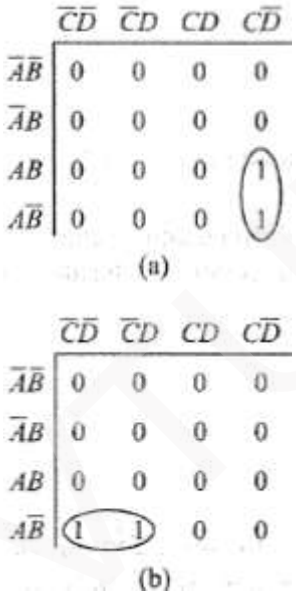


Figure 3.12: Pairs example

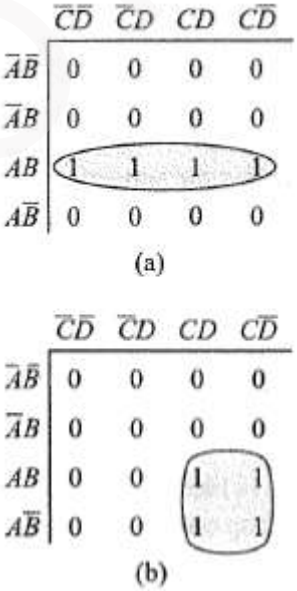


Figure 3.13: Quads example

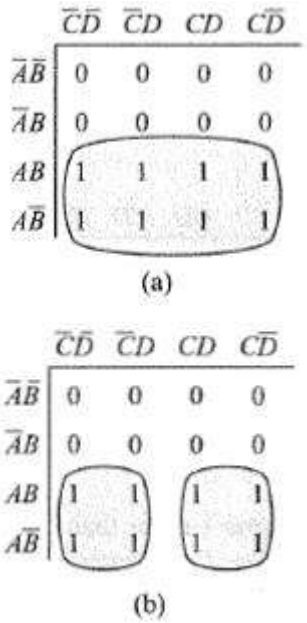


Figure 3.14: Octet example

KARNAUGH SIMPLIFICATIONS

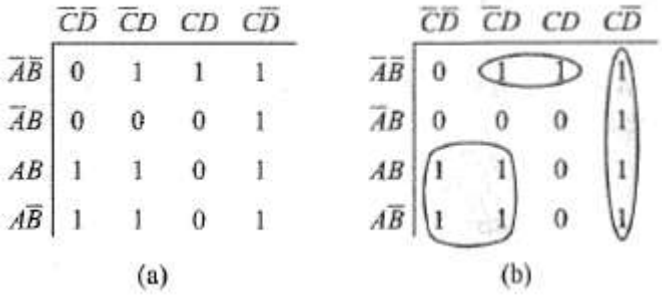


Figure 3.15: Encircling octets, quads and pairs

- In above kmap (Figure 3.15), we have
  1) The pair which represents A'B'D.
  2) The lower quad which represents AC'.
  3) The quad on the right which represents CD'.
- By ORing these simplified products, we get the Boolean equation corresponding to the entire Kmap:

  Y=A'B'D+AC'+CD'

Overlapping Groups

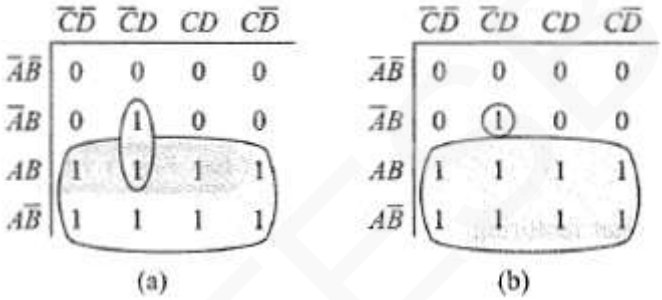


Figure 3.16: Overlapping groups

- **Overlapping groups** means using the same 1 more than once when looping the 1s of a Kmap.
- In Figure 3.16, the simplified equation for the overlapping groups is

  Y=A+BC'D
- So, always overlap groups if possible. That is, use the 1s more than once to get the largest groups you can.

Rolling the Map

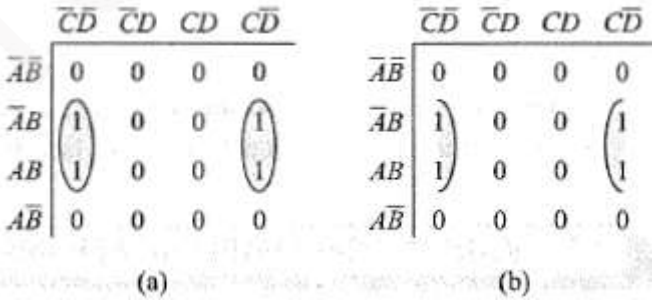


Figure 3.17: Rolling the Kmap

- Visualize picking up the Kmap and rolling it so that the left side touches the right side.
- If you are visualizing correctly, you will realize the two pairs actually form a quad.
- In Figure 3.17b, the quad has the equation

  Y=BD'
- Thus, 1s on the edges of a Kmap can be grouped with ls on opposite edges.

Rolling and Overlapping

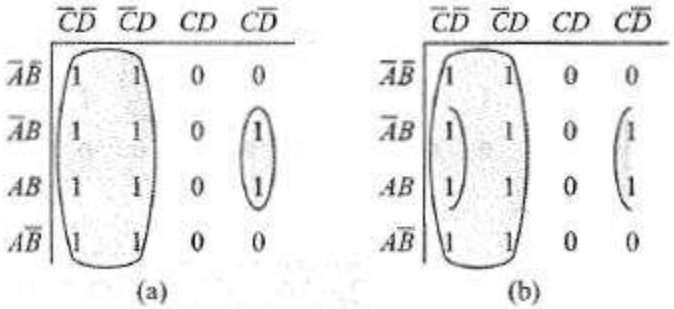


Figure 3.18: Rolling and overlapping

• In Figure. 3.18, we are rolling as well as overlapping; thus the Boolean equation is
Y= C'+BD'

Eliminating Redundant Groups

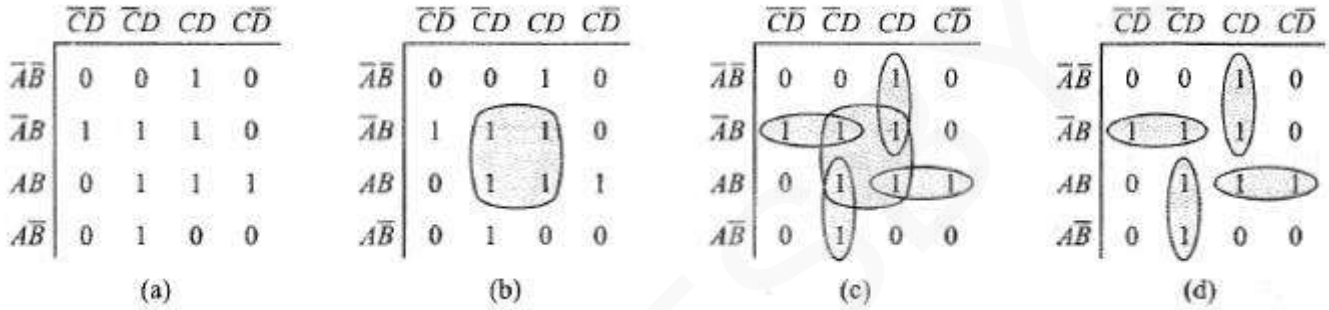


Figure 3.20: Eliminating an unnecessary group

• **Redundant group** means a group of 1s on a kmap that are all part of other groups (Figure 3.20).
• You can eliminate any redundant group.
• After you have finished encircling groups, eliminate any redundant group. This is a group whose 1s are already used by other groups.

KMAP METHOD FOR SIMPLIFYING BOOLEAN EQUATIONS

1) Enter a 1 on map for each fundamental product that produces a 1 output in truth table. Enter 0s elsewhere.
2) Encircle the octets, quads and pairs.
3) If any isolated 1s remain, encircle each.
4) Eliminate any redundant group.
5) Write boolean equation by ORing the products corresponding to the encircled group.

EVM METHOD FOR SIMPLIFYING BOOLEAN EQUATIONS

• Here, we make use of following identities:
  i) 1=1+C'
  ii) 1=1+C
  iii) 1=C+C'

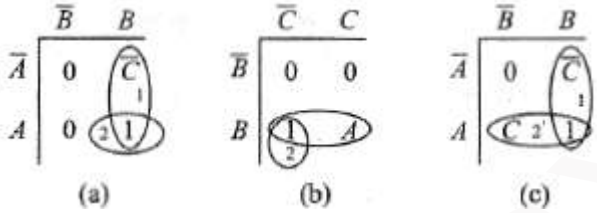


Figure 3.21: Simplification of entered variable map

Case (i): Example for using 1=1+C'

• As shown in Figure. 3.21a, C' is grouped with 1 to get a larger group as 1 can be written as 1=1+C'.
• Next, the product-term representing each group is obtained by including map entered variable in the group as an additional ANDed term.
• Here,
  Group-1 gives B.(C')=BC' and
  Group-2 gives AB.(1)=AB
  Thus, Y=BC'+ AB.

Case (ii): Example for using 1=1+C

• As shown in Figure. 3.21b, A is grouped with 1 to get a larger group as 1 can be written as 1=1+A.
• Here,
  Group-1 gives B.(A)=AB and
  Group-2 gives BC'.(1)=BC'
  Thus, Y=BC'+ AB.

Case (iii): Example for using 1=C+C'

• As shown in Figure. 3.21c, EVM
  → has only two product-terms and
  → doesn't need a separate coverage of 1.
• This is because one can write 1=C+C' and C is included in one group while C' in other.
• Here,
  Group-1 gives B.(C')=BC' and
  Group-2 gives A.(C)=AC
  Thus, Y =AC+ BC'.

EXAMPLE 3.6

What is the simplified Boolean equation for the following logic equation expressed by minterms?

Y=F(A,B,C,D)=Σm(7,9, 10, 11, 12, 13, 14, 15)

Solution We know, each minterm makes corresponding location in Karnaugh map 1 and thus Fig. 3.22a represents the given equation. There are no octets, but there is a quad as shown in Fig. 3.22b. By overlapping, we can find two more quads (see Fig. 3.22c). We can encircle the remaining 1 by making it part of an overlapped pair (Fig. 3.22d). Finally, there are no redundant groups.

The horizontal quad of Fig. 3.22d corresponds to a simplified product AB. The square quad on the right corresponds to AC, while the one on the left stands for AD. The pair represents BCD. By ORing these products, we get the simplified Boolean equation:

$$Y = AB + AC + AD + BCD \quad (3.30)$$

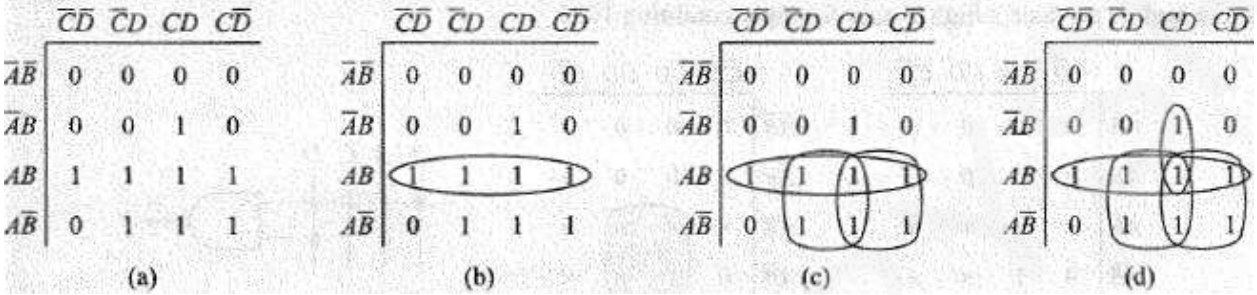


Figure 3.21: Using the Kmap

DON'T CARE CONDITION (X)

- This is an input-output condition that never occurs during normal operation.
- Since the condition never occurs, you can use an 'X' on the Kmap (Table: 3.8).
- This X can be a 0 or a 1, whichever you prefer (Figure: 3.23)

A	B	C	D	Y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X

Table 3.8: Truth table with Don't care conditions

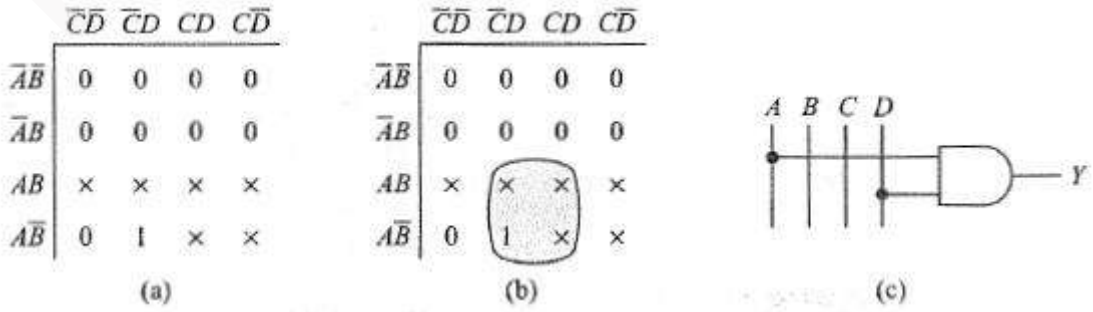


Figure 3.23:Don't care conditions

HOW TO USE DON'T CARE CONDITIONS(X) IN KMAP SIMPLIFICATION

1) Given the truth table, draw a kmap with 0s, 1s and don't cares.
2) Encircle the actual 1s on the kmap in the largest groups you can find by treating the don't cares as 1s.
3) After the actual 1s have been included in groups, discard the remaining don't cares by visualizing them as 0s.

EXAMPLE 3.8

Give the simplest logic circuit for following logic equation where d represents don't-care condition for following locations

$$F(A,B,C,D)=\Sigma m(7)+d(10,11,12,13,14,15)$$

Solution Figure 3.25a is the Karnaugh map. The most efficient encircling is to group the 1s into a pair using the don't-care as shown. Since this is the largest group possible, all remaining don't cares are treated as 0s. The equation for the pair is

$$Y = BCD$$

and Fig. 3.25b is the logic circuit. This 3-input AND gate produces a high output only for an input of $A = 0$, $B = 1$, $C = 1$, and $D = 1$ because the input possibilities range only from 0000 to 1001.

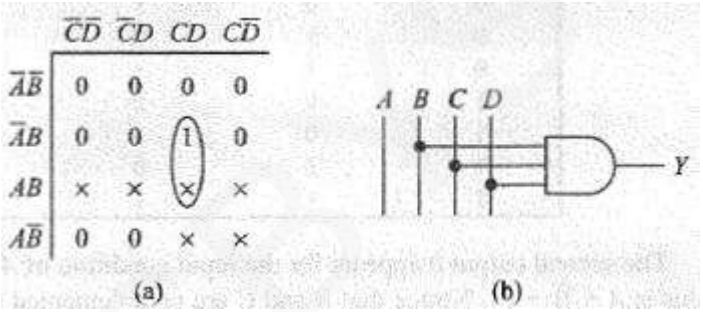


Figure 3.25: Decoding 0111

PRODUCT OF SUMS METHOD

• **Product-of-sums equation** means the logical product of those fundamental sums that produce output 0s in the truth table.
• Each sum-term is called **maxterm**.
For example, (A+B), (A+B+C), (A+B+C+D) etc
• Here, we have to locate output 0 in the truth table and write down the maxterm (Table 3.9).
• Consider Table 3.4,
➢ First output 0 appears for an input A=0, B=0 and C=0. The corresponding maxterm is A+B+C.
➢ Second output 0 appears for A=0, B=1 and C=1. The corresponding maxterm is A+B'+C'.
➢ Third output 0 appears for A=1, B=1 and C=0. The corresponding maxterm is A'+B'+C.
• Y=F(A,B,C) means Y is a function of 3 boolean variables A,B and C.
• To get the product-of-sums equation, we have to AND the maxterms.
Y=F(A,B,C)=(A+B+C).(A+B'+C').(A'+B'+C)
Y=F(A,B,C)=$M_0.M_3.M_6$ where maxterm is denoted by M_i.
Y=F(A,B,C)=∏M(0,3,6) where ∏ denotes product, i.e. AND operation
• This kind of representation of a truth table is also known as **canonical product form**.
• the logic circuit for Y is shown in Figure: 3.26.
• The corresponding logic circuit is
→ OR-AND circuit or
→ NOR-NOR circuit (Figure 3.27).

A	B	C	Y	Maxterm
0	0	0	$0 \rightarrow A+B+C$	M_0
0	0	1	1	M_1
0	1	0	1	M_2
0	1	1	$0 \rightarrow A+\overline{B}+\overline{C}$	M_3
1	0	0	1	M_4
1	0	1	1	M_5
1	1	0	$0 \rightarrow \overline{A}+\overline{B}+C$	M_6
1	1	1	1	M_7

Table 3.9: Fundamental Sums for Three inputs

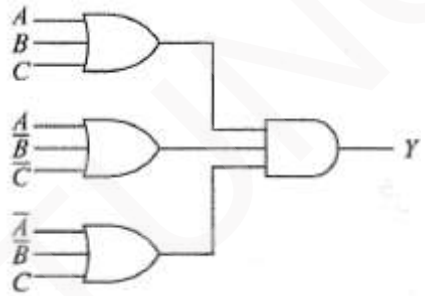


Figure 3.26:Product-of-sums circuit

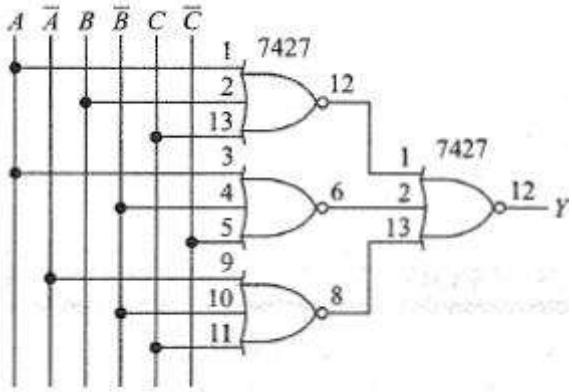


Figure 3.27:using only NOR gates

STEPS TO CONVERT BETWEEN STANDARD SOP & POS FORM

1) Identify complementary locations.
2) Changing minterm to maxterm or reverse.
3) Changing summation by product or reverse.

• This is known are **conversion between canonical forms**.
For example,
If Y=F(A,B,C)=A'BC'+AB'C+A'B'C' then Y'=F'(A,B,C)=(A+B'+C).(A'+B+C').(A+B+C)
If Y=F(A,B,C)=∏M(0,3,6) then Y'=F'(A,B,C)=Σm(1,2,4,5,7)

EXAMPLE 3.9

Suppose a truth table has a low output for the first three input conditions: 000, 001, and 010. If all other outputs are high, what is the product-of-sums circuit?

Solution The product-of-sums equation is

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

The circuit of Fig. 3.27 will work if we reconnect the input lines as follows:

A : pins 1, 3, and 9
B : pins 2 and 4
C : pins 13 and 11
$\overline{B}$: pin 10
$\overline{C}$: pin 5

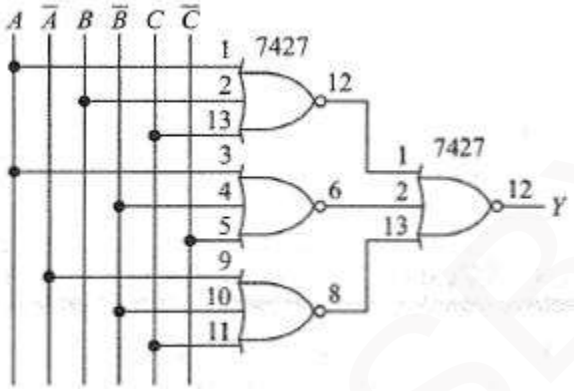


Figure 3.27: Using only NOR gates

LIMITATIONS (OR DRAWBACKS) OF KMAP

1) The map method depends on the user's ability to identify patterns that gives largest size.
2) The map method becomes difficult to adapt for simplification of 5 or more variables.

PRODUCT-Of-SUMS SIMPLIFICATION

- Procedure to simplify boolean equation in product-of-sums form:
  1) Convert the truth table into a Kmap.
  2) Complement the Kmap i.e. interchange 0 and 1.
  3) Group the 1s, write the sum-of-products equation for Y'.
  4) Using duality theorem, convert the sum-of-products equation to equivalent product-of-sums equation.

EXAMPLE PROBLEM

- Consider truth table: Table 3.10.
  1) Convert the truth table into a Kmap as shown in 3.28a.
  2) Complement the Kmap i.e. interchange 0 and 1 as shown in Fig. 3.28b.
  3) After grouping the 1s, we have the sum-of-products equation Y'.
     Y'=A'B+AB'C'
  4) Using duality theorem, the sum-of-products equation is converted to equivalent product-of-sums equation as follows:
     Y=(A+B').(A'+B+C)
- The logic circuit for Y is shown in Figure: 3.28c.

A	B	C	D	Y
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Table 3.10

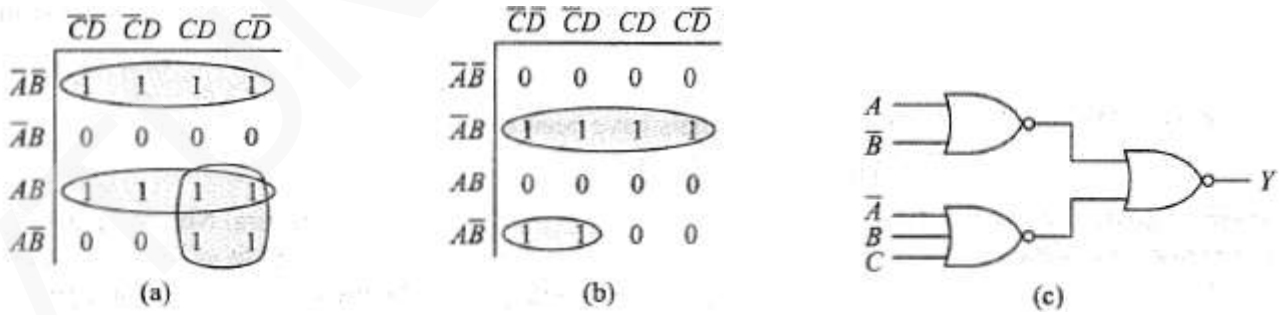


Figure 3.28: Deriving the product-of-sums circuit

EXAMPLE 3.10

Show the sum-of-products and product-of-sums circuits for the Kmap of Fig. 3.30a.

Solution The Boolean equation for Fig. 3.30a on the next page is

$$Y = A + BC\overline{D}$$

Figure 3.30b is the sum-of-products circuit.

After complementing and simplifying the Karnaugh map, we get Fig. 3.30c. The Boolean equation for this is

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

Figure 3.30d is the sum-of-products circuit for the $\overline{Y}$. As shown earlier, we can convert the dual circuit into a NOR-NOR equivalent circuit to get Fig. 3.30e.

The two design choices are Fig. 3.30b and 3.30e. Figure 3.30b is simpler.

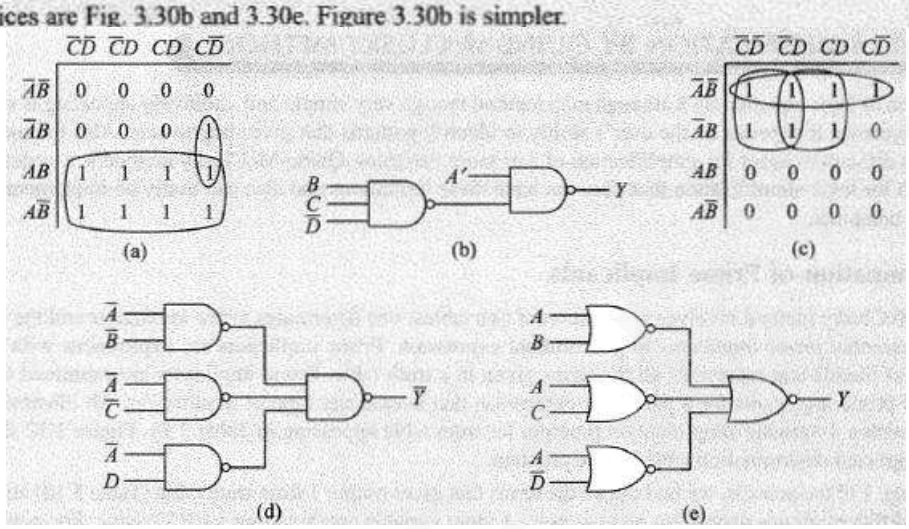


Figure 3.30

EXAMPLE 3.11

Give simplest POS form of Kmap shown in Fig. 3.30a by grouping zeros.

Solution Refer to grouping of zeros as shown in Fig. 3.31a. Three groups cover all the zeros that give three sum terms. The first group has A' and C' constant within the group that gives sum term $(A + C)$. Group 2 has A' and D constant giving sum term $(A + D')$. Group 3 has A' and B' constant generating $(A + B)$ as sum term.

The final solution is thus product of these three sum terms and expressed as

$$Y = (A + B)(A + C)(A + D')$$

Note that, the above relation can be realized by OR-AND circuit or NOR-NOR (Fig. 3.30e) circuit.

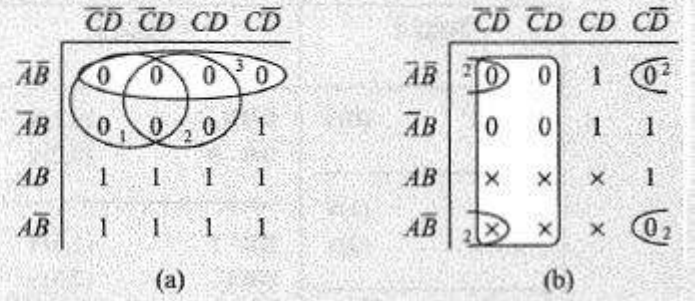


Figure 3.31: Simplification by grouping zeros

EXAMPLE 3.12

Give simplest POS form of Kmap shown in Fig. 3.31b by grouping zeros.

Solution In a Karnaugh map if don't care conditions exist, we may consider them as zeros if that gives larger group size. This in turn reduces number of literals in the sum term. Refer to grouping of zeros in Fig. 3.31b. We require minimum two groups that includes all the zeros and are also largest in sizes. In group 1, only C' is constant that gives only one literal in sum term as C. Group 2 has B' and D' constant giving sum term $(B + D)$. The final solution is thus product of these two sum terms and expressed as

$$Y = C(B + D)$$

SIMPLIFICATION BY QUINE MCCLUSKY METHOD

- **Quine-McClusky method** is a tabular method for logic simplification.
- It does not have the limitations of Kmap.
- This method involves preparation of 2 tables:
  - → one determines prime implicants and
  - → other selects essential prince implicants to get minimal expression.
- **Prince implicants** are expressions with least number of literals that represents all the terms given in a truth table.
- Prime implicants are examined to get essential prime implicants for a particular expression that avoids any type of duplication.
- **Essential prime implicants** are similar to prince implicants but does not contain duplicate expressions.

PROCEDURE USED FOR DETERMINING ESSENTIAL PRIME IMPLICANTS

- Consider a 4-variable simplification problem for Table 3.10. Figure 3.32 shows prime implicant determination table for the problem.

Stage 1

i) We find out all the terms that gives output 1 from truth table (Table 3.10).

ii) We put them in different groups depending on how many 1 i/p variable combinations have.

For example,

First group has no 1 in input combination.
Second group has only one 1.
Third group has two 1s.
Fourth group has three 1s.
Fifth group has four 1s.

iii) We also write decimal equivalent of each combination to their right for convenience.

Stage 2

i) We first try to combine first and second group of stage 1, on a member to member basis.

ii) The rule is to see if only one binary digit is differing between two members and we mark that position by '-'.

iii) This means corresponding variable is not required to represent those members.

Stage 3

i) We combine members of different groups of stage 2 in a similar way.

ii) Now it will have two '-' elements in each combination.

iii) This means each combination requires 2 literals to represent it.

iv) There is no stage 4 for this problem. This completes process of determination of prime implicants.

A	B	C	D	Y
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Table 3.10

Stage 1		Stage 2		Stage 3	
ABCD		*ABCD*		*ABCD*	
0000	(0)√	000-	(0,1)√	00--	(0,1,2,3)
		00-0	(0,2)√	00--	(0,2,1,3)
0001	(1)√				
0010	(2)√	00-1	(1,3)√	-01-	(2,10,3,11)
		001-	(2,3)√		
0011	(3)√	-010	(2,10)√	1-1-	(10,11,14,15)
1010	(10)√			1-1-	(10,14,11,15)
1100	(12)√	-011	(3,11)√	11--	(12,13,14,15)
		101-	(10,11)√	11--	(12,14,13,15)
		1-10	(10,14)√		
1011	(11)√	110-	(12,13)√		
1101	(13)√	11-0	(12,14)√		
1110	(14)√				
		1-11	(11,15)√		
1111	(15)√	11-1	(13,15)√		
		111-	(14,15)√		

Figure 3.32: Determination of prime implicants

SELECTION OF PRIME IMPLICANTS

• Next step is to select essential prime implicants and remove duplication among them.

• For this, we prepare a table as shown in Table 3.11 that along the row lists all the prime implicants and along columns lists all minterms.

• The cross-point of a row and column is ticked if the term is covered by corresponding prime implicant.

• For example,

Terms 0 and 1 are covered by A'B'

Terms 2 and 3 are covered by both A'B and B'C'.

Thus, the corresponding cross-points are ticked.

This way we complete the table for rest of the terms.

	0	1	2	3	10	11	12	13	14	15
$A'B'$ (0,1,2,3)	√	√	√	√						
$B'C$ (2,3,10,11)			√	√	√	√				
AC (10,11,14,15)					√	√			√	√
AB (12,13,14,15)							√	√	√	√

Table: 3.11

• Selection of essential prime implicants from this table is done in the following way:

i) We find minimum number of prime implicants that covers all the minterms.

ii) We find A'B' and AB cover terms that are not covered by others and they are essential prime implicants.

iii) B'C and AC among themselves cover 10,11 which are not covered by others. So, one of them has to be included in the list of essential prime implicants making it three. And the simplified representation of truth table given in Table3.10 is one of the following

Y=A'B'+B'C+AB or Y=A'B'+AC+AB

EXAMPLE 3.13

Give simplified logic equation of Table 3.6 by Quine-McClusky method.

Solution Tables that determine prime implicants and selects essential prime implicants are shown in Figs. 3.33a and 3.33b respectively. We find both the prime implicants are essential prime implicants. The simplified logic equation thus is expressed as

$$Y = AB + BC'$$

Note that, we got the same expression by simplification entered variable map shown in Figs. 3.22a and 3.22b.

Stage 1		Stage 2	
A B C		A B C	
0 1 0	(2)√	- 1 0	(2,6)
1 1 6	(6)√	1 1 -	(6,7)
1 1 1	(7)√		

(a)

	2	6	7
BC' (2,6)	√	√	
AB (6,7)		√	√

(b)

Figure 3.33: Simplification by Quine-McClusky method for Example 3.14

HAZARDS & HAZARD COVERS

- The unwanted switching transient that may appear at the output of a circuit are called **hazards**.
- The hazards cause the circuit to malfunction.
- The main cause of hazards is the different propagation delays at different paths.
- Here we consider, 1) Static-1 hazard 2) Static-0 hazard & 3) Dynamic hazard.

STATIC-1 HAZARD

- In a combinational circuit, if output goes momentarily 0 when it should remain a 1, the hazard is known as **static-1 hazard** (Figure: 3.34).
- Static-1 hazard occurs when
  - → Y=A+A' type of situation appears in a logic circuit.
  - → 'A' makes a transition 1→0
- Here is how static-1 hazard occurs:
  - ➢ An A+A' condition should always generate 1 at the output, i.e. static-1.
  - ➢ But the NOT gate output takes finite time to become 1 following 1→0 transition of A.
  - ➢ Thus for OR gate, there are 2 zeros appearing at its input for the small duration, resulting a 0 at its output.
  - ➢ The width of this zero is in nanosecond order and is called a **glitch**.
- In combinational circuits, static-1 hazard may not cause any serious problem.
  But in sequential circuit, static-1 hazard may cause major malfunctioning.

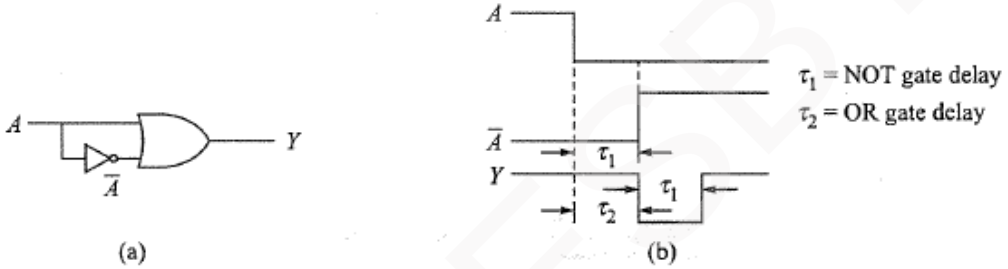


Figure 3.34: Static-1 hazard

Hazard cover means additional gates in logic circuit which prevents hazard.

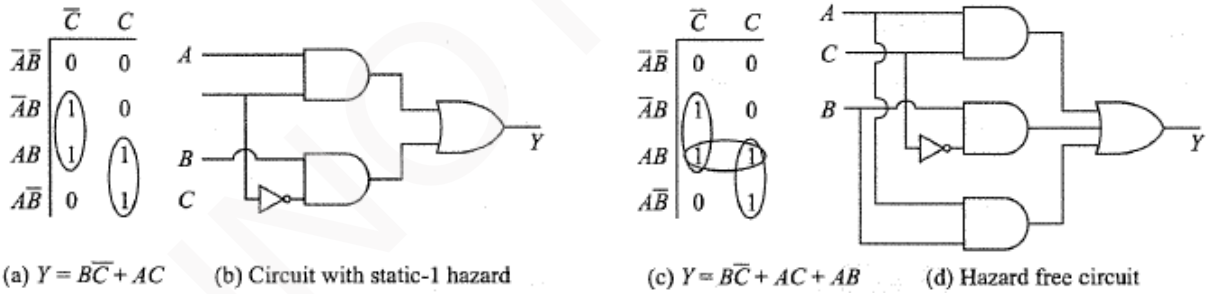


Figure 3.35: Static-1 hazard and its cover

Example for circuit with static-1 hazard

- Consider Kmap shown in Fig. 3.35a.
- Minimally, the Kmap is represented by output Y=BC'+AC.
- The corresponding circuit is shown in Fig. 3.35b.
- Assume, input A=1, B=1.
- When C makes a transition 1→0, there will be static-1 hazard occurring at output.

Example for circuit with hazard cover

- Consider another grouping for the Kmap in Fig.3.35c.
- This includes one additional term AB and now output Y=BC'+AC+AB.
- The corresponding circuit is shown in Fig. 3.35d.
- This circuit is hazard free.
- This circuit requires more hardware than minimal representation.
- The additional term AB ensures Y=1 for A=1, B=1.
- A 1→0 transition at C does not affect output.

STATIC-0 HAZARD

• In a combinational circuit, if output goes momentarily 1 when it should remain a 0, the hazard is known as **static-0 hazard** (Figure: 3.36).

• Static-0 hazard occurs when

→ Y=AA' type of situation appears in a logic circuit.

→ 'A' makes a transition 0→1.

• Here is how static-0 hazard occurs:

- An AA' condition should always generate 0 at the output, i.e. static-0.
- But the NOT gate output takes finite time to become 0 following 0→1 transition of A.
- Thus for AND gate, there are 2 ones appearing at its input for the small duration, resulting a 1 at its output.
- The width of this zero is in nanosecond order and is called a **glitch**.

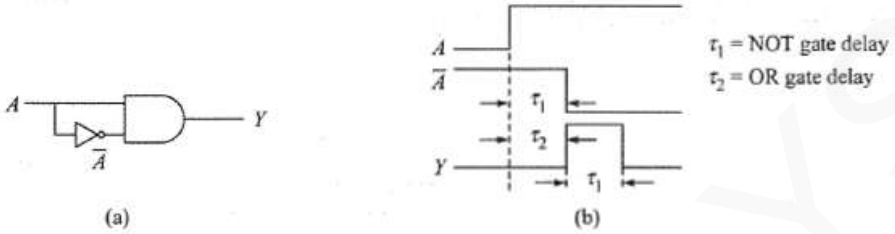


Figure 3.36: Static-0 hazard

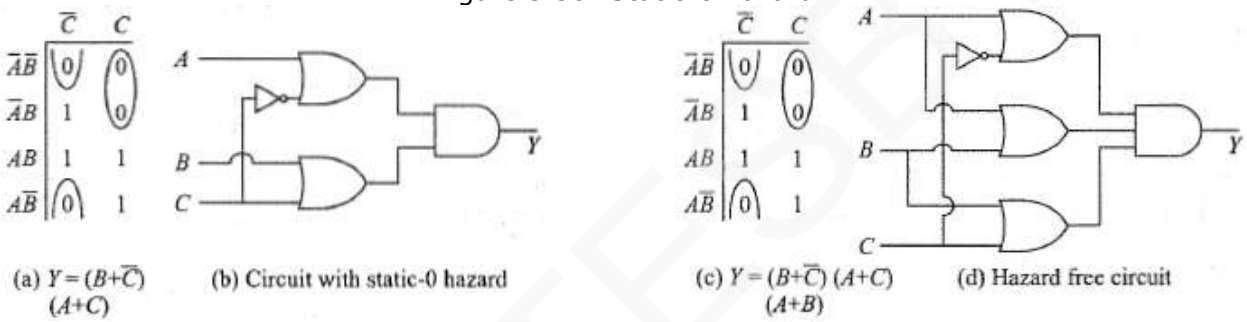


Figure 3.37: Static-0 hazard and its cover

Example for circuit with static-0 hazard

- Consider Kmap shown in Fig. 3.37a.
- Minimally, the Kmap is represented by output Y=(B+C)(A+C')
- The corresponding circuit is shown in Fig. 3.37b.
- Assume, input A=0, B=0.
- When C makes a transition 0→1, there will be static-0 hazard occurring at output.

Example for circuit with hazard cover

- Consider another grouping for the Kmap in Fig.3.37c.
- This includes one additional term (A+ B) and now output Y=(B+C)(A+C')(A+B)
- The corresponding circuit is shown in Fig. 3.37d.
- This circuit is hazard free.
- This circuit requires more hardware than minimal representation.
- The additional term (A + B) ensures Y=0 for A=0, B=0.
- A 0→1 transition at C does not affect output.

DYNAMIC HAZARD

• **Dynamic hazard** occurs when circuit output makes multiple transitions before it settles to a final value while the logic equation asks for only one transition.

• When this hazard occurs, an output transition designed as

1→0 may give 1→0→1→0

0→1 may give 0→1→0→1

• The output of logic equation in dynamic hazard degenerates into Y=A+A'.A or Y=(A+A').

• A kind of relations for certain combinations of the other input variables.

• Dynamic hazard occur in multilevel circuits having implicit static-1 and/or static-0 hazards.

• By providing covers to each one of them, dynamic hazard can be prevented.

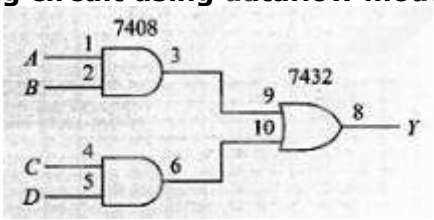
HDL IMPLEMENTATION MODEL
DATAFLOW MODELING

- 1) Keyword '**assign**' and 2) Set of operators are used to describe a circuit through its behavior or function.
- Set of operators is given in Table 3.12.
- All 'assign' statements are concurrent i.e. order in which they appear do not matter.
- All 'assign' statements are also continuous i.e. any change in a variable in the RHS will immediately effect LHS output.
- We do not explicitly need to define any gate structure using nand, nor etc.
- We do not use intermediate variables through wire. The compiler takes care of this.

Relational Operation	*Symbol*	*Bit-wise Operation*	*Symbol*
Less than	<	Bit-wise NOT	~
Less than or equal to	<=	Bit-wise AND	&
Greater than	>	Bit-wise OR	|
Equal to	==	Bit-wise Ex-OR	^
Not equal to	!=		
Logical Operation (for expressions)	**Symbol**	**Arithmetic Operation**	**Symbol**
		Binary addition	+
Logical NOT	!	Binary subtraction	–
Logical AND	&&	Binary multiplication	*
Logical OR	||	Binary division	/

Table 3.12: List of Verilog operator

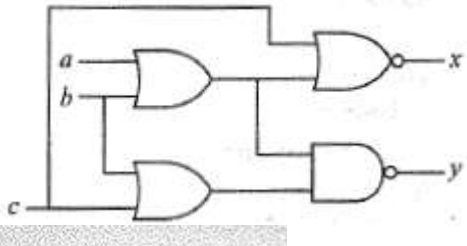
Write a verilog code for following circuit using dataflow modeling.




```
module fig2_24a(A,B,C,D,Y);
  input A,B,C,D;
  output Y;
  assign Y=(A&B)|(C&D);
  //One statement is enough

endmodule
```

Write a verilog code for following circuit using dataflow modeling.

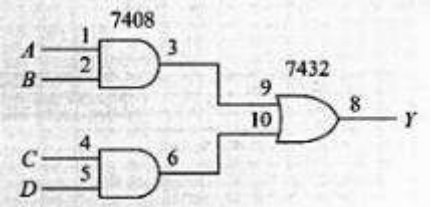



```
module testckt(a,b,c,x,y);
  input a,b,c;
  output x,y;
  assign x=~((a|b)|c); // NOR through NOT-OR
  assign y=~((a|b)&(b|c)); /* NAND by NOT-
  AND*/
endmodule
```

BEHAVIORAL MODELING

- Statements are executed sequentially following algorithmic description.
- It is ideally suited to describe a sequential logic circuit.
- It always uses keyword '**always**' followed by a sensitivity-list.
- Procedural assignment or output variables within 'always' must be register type, defined by '**reg**'.
- Unlike 'wire', 'reg' is not continuously updated
  but 'reg' is updated only after a new value is assigned to it.
- 'wire' variables can only be read and not assigned to in any procedural block.
- 'wire' variables cannot store any value.

Write a verilog code for following circuit using behavioral modeling.




```
module fig2_24a(A,B,C,D,Y);
  input A,B,C,D;
  output Y;
  reg Y;       /* Y is output after procedural assignment within always
                  block, hence as reg.*/
always @ (A or B or C or D) //A,B,C,D form sensitivity list, note keyword
                            or
  if ((A==1)&&(B == 1)) // If A,B both are 1
     Y=1;               // Assignment through equal sign not keyword assign
  else if ((C==1)&&(D == 1)) // if C,D both are 1
     Y=1;
  else                  // for all other combinations of A,B,C,D
     Y=0;
endmodule
```

EXAMPLE 3.15

Get a minimized expression for Y=F(A, B, C) =A'B'C'+ A'B'C+ A'BC+ AB'C

Solution We can solve this using Boolean Algebra, Karnaugh Map, Entered Variable Map and QM Algorithm.

In Method-1 We take help of Boolean Algebra for minimization. We see that $\overline{A}\,\overline{B}C$ can be combined with all three terms using distributive law (Eq. 3.5)

Since, in Boolean algebra $X = X + X + X$ (extending Eq. 3.7) we can write

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

From associative law (Eq. 3.3)

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

From distributive law (Eq. 3.5)

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

From Eq. 3.9, since $X + \overline{X} = 1$

$$Y = \overline{A}\,\overline{B} \cdot 1 + \overline{A}C \cdot 1 + \overline{B}C \cdot 1$$
$$= \overline{A}\,\overline{B} + \overline{A}C + \overline{B}C$$

In Method-2, we use Karnaugh Map for minimization. Fig. 3.39 shows the solution by this method.

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

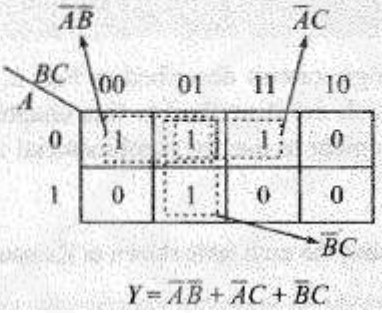


Figure 3.39: Solution using Karnaugh Map

In Method-3, we use Entered Variable Map for minimization. Figure 3.40 shows the solution by this method.

Since $1 = C + \overline{C}$, we need a separate group for $AB = 00$ as $\overline{C}$ is not explained by other two groups. We use C embedded in 1 to make other two groups bigger and reduce the number of literals, and thus minimize the expression.

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

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

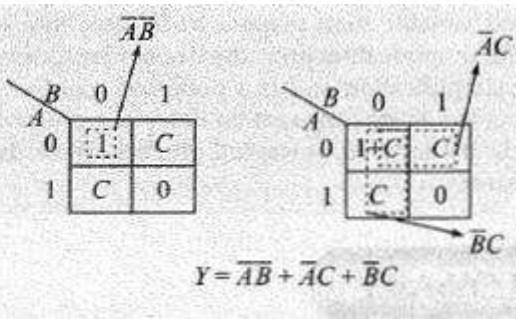


Figure 3.40: Solution using Entered Variable Map

In Method-4, we use QM algorithm for minimization. Fig. 3.41 shows prime implicants and essential prime implicants. The final solution is arrived at by combining essential prime implicants.

Stage 1			Stage 2	
ABC			*ABC*	
000	(0)	√	00–	(0, 1)
001	(1)	√	0–1	(1, 3)
			–01	(1, 5)
011	(3)	√		
101	(5)	√		

Prime implicants only from stage 2.
They are:
00–($A'B'$), 0–1 ($A'C$) and –01 ($B'C$)

	0	1	3	5
$A'B'$	√	√		
$A'C$		√	√	
$B'C$		√		√

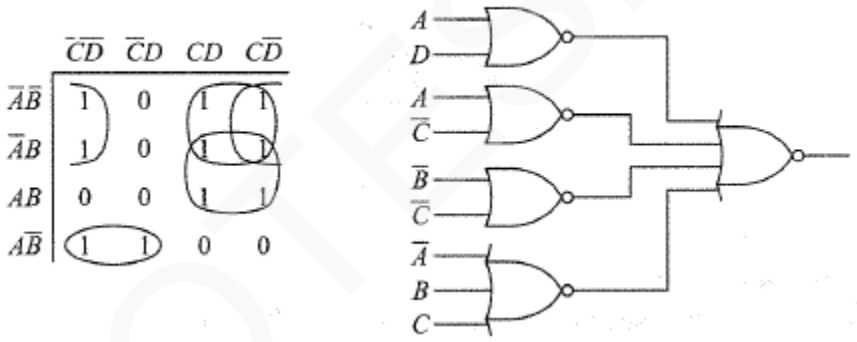
All are essential
$Y = A'B' + A'C + B'C$

Figure 3.41: Solution using QM Algorithm

EXAMPLE 3.16
What is the simplified NOR-NOR circuit for following Boolean equation:
Y=F(A,B,C,D)=Σm(0,2,3,4,6,7,8,9,14,15)

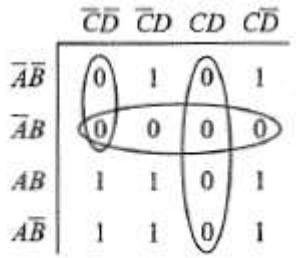
Solution:

The map and the circuit.



EXAMPLE 3.17
Simplify to give POS form by grouping zeros in Kmap for equation
Y=F(A,B,C,D)=∏M(0,3,4,5,6,7,11,15)

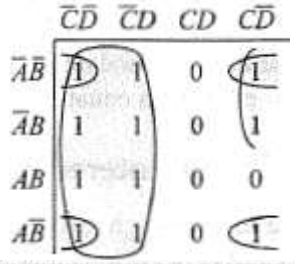
Solution:



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

EXAMPLE 3.18

Realize the truth table shown in below Kmap using data flow model



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

Solution The simplified logic equation of this is given by $Y = C' + A'D' + B'D'$. We use this in **assign** statement to get HDL description.

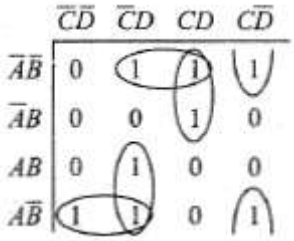
```
module fig3_20a(A,B,C,D,Y);
  input A,B,C,D;
  output Y;
  assign Y= ~C | (~A & ~D) | (~B & ~D); // ~ operator has higher precedence
endmodule
```

EXAMPLE 3.19

Draw the Kmap for below Table.

A	B	C	D	Y
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

Solution:



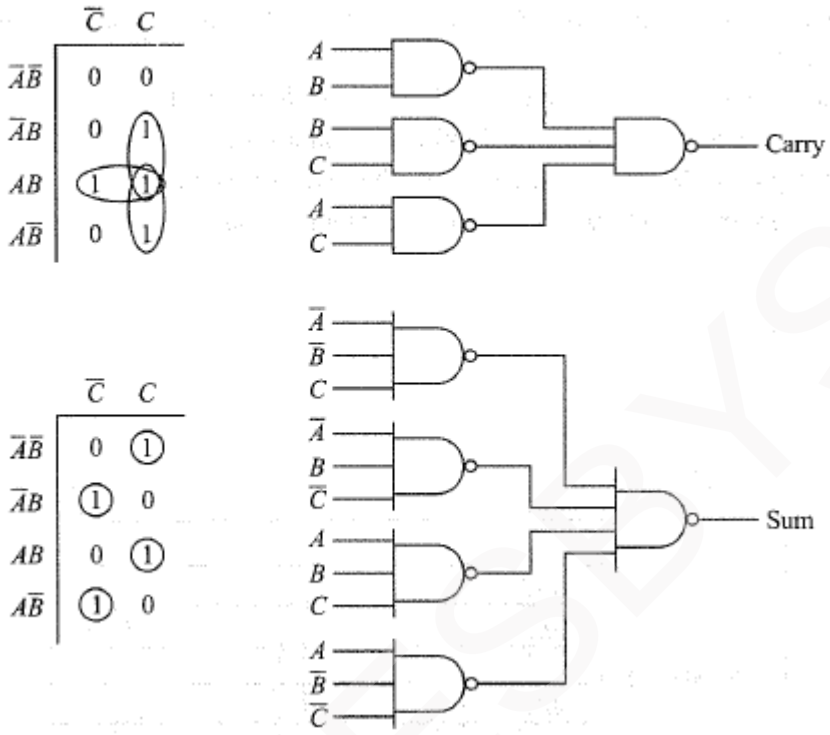
	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	0	1	1	1
$\bar{A}B$	0	0	1	0
AB	0	1	0	0
$A\bar{B}$	1	1	0	1

EXAMPLE 3.20

Table 3.16 is the truth table of full adder, a logic circuit with two outputs called the CARRY and the SUM. What is the simplified NAND-NAND circuit for the CARRY output? For the SUM output?

Solution:

The simplified NAND-NAND circuits.



MODULE 3: DATA PROCESSING CIRCUITS

MULTIPLEXER

- It is a digital circuit with many inputs but only 1 output (Figure 4.1a).
- By applying control-signals, we can steer any input to output.
- Thus, it is also called a **data-selector** and control-inputs are called **select inputs.**

4:1 Multiplexer

- 4:1 mux has
  - → 4 data inputs
  - → 1 data output and
  - → 2 select inputs (Figure 4.1b).

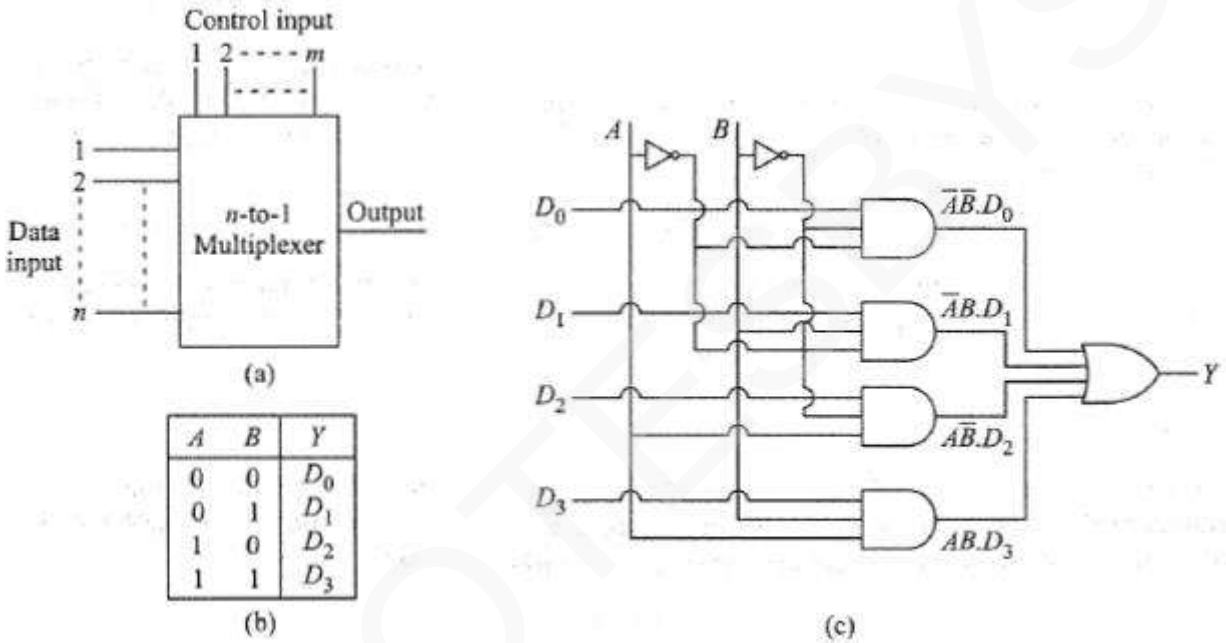


Figure 4.1: a)Multiplexer block diagram b)4:1 multiplexer truth table c)logic circuit

- Depending on select-inputs A and B, one of the 4 inputs D_0 to D_3 is steered to output Y (Figure 4.1c).
- The logic equation of the circuit is given by

  $Y=A'B'D_0+A'BD_1+AB'D_2+ABD_3$
- Here, each AND gate generates a minterm which are finally summed by OR gate.

  If A=0, B=0; $Y=0'0'D_0+0'0D_1+00'D_2+ABD_3$

  $Y=1.1.D_0+0+0+0$

  $Y=D_0$
- In other words, for AB=00,
  - → first AND gate to which D_0 is connected remains active & equal to D_0.
  - → all other AND gate are inactive.

    If D_0=0, then Y=0 and if D_0=1, then Y=1.
- Similarly, for AB=01,
  - → second AND gate will be active and
  - → all other AND gates remain inactive. Thus, output $Y=D_1$.

74150

- The 74150 is a 16-to-1 TTL multiplexer (Figure 4.3).
- 16:1 mux has
  - → 16 data inputs D_0, D_1, D_2 , , , , ,D_{15}
  - → 1 data output Y and
  - → 4 select inputs A, B, C, & D (Table 4.1).
- **STROBE'** input pin is used to enable or disable the multiplexer.
- The multiplexer is
  - → active (enabled) when the STROBE is low and
  - → inactive (disabled) when the STROBE is high.
- The STROBE is called an **active-low signal**; because it causes something to happen when it is low rather than when it is high.
- Here's how it works (Figure 4.3):
  - ➢ A low on strobe enables the mux, so that output Y equals complement of the input data bit.
    $Y=D'_n$ where n is the decimal equivalent of ABCD.
  - ➢ On the other hand, a high on strobe disables the mux and forces the output into the high state. With a high on strobe, the value of ABCD doesn't matter.

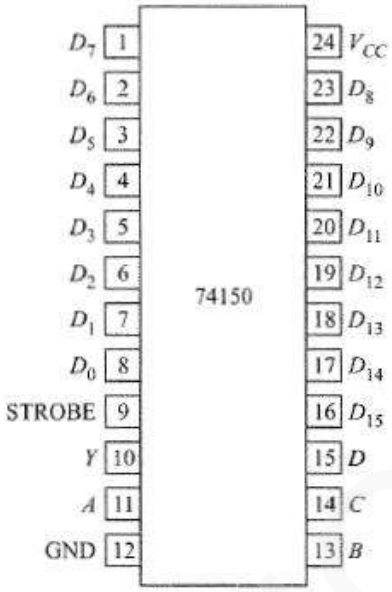


Figure 4.3: Pinout diagram of 74150

Strobe	A	B	C	D	Y
L	L	L	L	L	$\overline{D_0}$
L	L	L	L	H	$\overline{D_1}$
L	L	L	H	L	$\overline{D_2}$
L	L	L	H	H	$\overline{D_3}$
L	L	H	L	L	$\overline{D_4}$
L	L	H	L	H	$\overline{D_5}$
L	L	H	H	L	$\overline{D_6}$
L	L	H	H	H	$\overline{D_7}$
L	H	L	L	L	$\overline{D_8}$
L	H	L	L	H	$\overline{D_9}$
L	H	L	H	L	$\overline{D_{10}}$
L	H	L	H	H	$\overline{D_{11}}$
L	H	H	L	L	$\overline{D_{12}}$
L	H	H	L	H	$\overline{D_{13}}$
L	H	H	H	L	$\overline{D_{14}}$
L	H	H	H	H	$\overline{D_{15}}$
H	X	X	X	X	H

Table 4.1: 74150 Truth Table

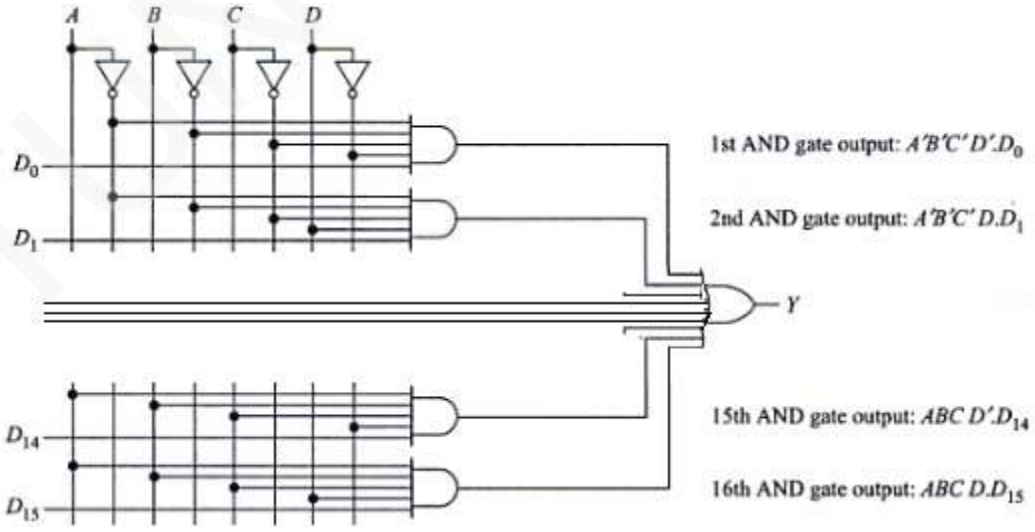


Figure 4.3: 16:1 Multiplexer

MULTIPLEXER LOGIC

• Two standard methods for implementing a truth table are SOP and POS solutions.

• The third method is the multiplexer solution. For example to use a 74150 to implement Table 4.2, complement each Y output to get the corresponding data input.

D_0=1'=0
D_1=0'=1
D_2=1'=0
….
….
D_{15}=1'=0

• D_0 is grounded, D_1 is connected to +5V, D_2 is grounded and so forth (Figure 4.4).

- When ABCD=0000, D_0 is the selected input. Since D_0 is low, Y is high.
- When ABCD=0001, D_1 is the selected input. Since D_1 is high, Y is low.
- When ABCD=0010, D_2 is the selected input. Since D_2 is low, Y is high.

• The multiplexer is

→ active (enabled) when the STROBE is low and
→ inactive (disabled) when the STROBE is high.

A	B	C	D	Y
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

Table 4.2

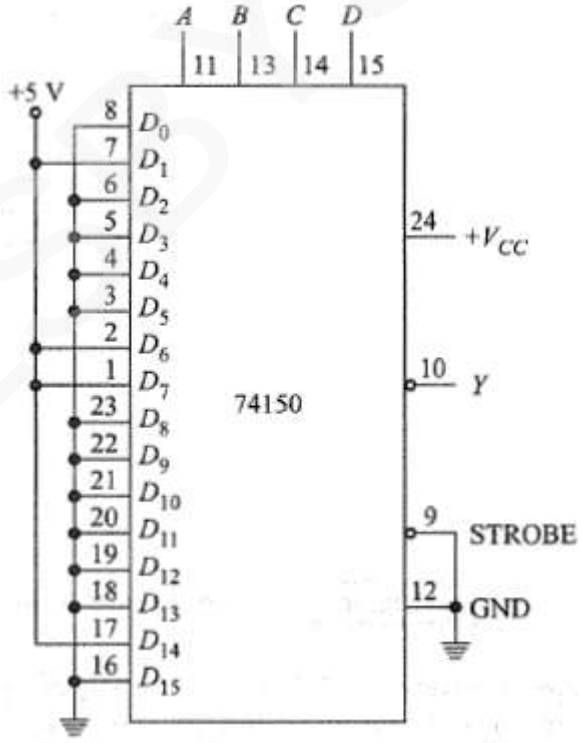


Figure 4.4:Using a 74150 for multiplexer logic

Why Multiplexer is called Universal Logic Circuit?

- Because a 2^n to 1 multiplexer can be used as a design solution for any 'n' variable truth table.
- We can implement the Truth table: 4.2 using 8:1 multiplexer in following way:
  - Let's consider A, B and C variables to be fed as select inputs.
  - The fourth variable D then has to be present as data input.
  - Here, EVM method can be used to convert 4-variable truth table into 3-variable truth table.

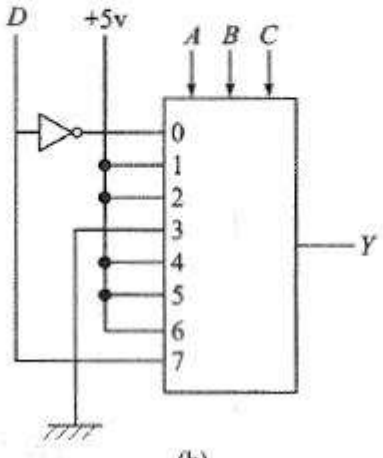
Table 4.2a:Four variable truth table

A	B	C	D	Y
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

Table 4.2b: Three variable truth table

ABC	000	001	010	011	100	101	110	111
D = 0	1	1	1	0	1	1	1	0
D = 1	0	1	1	0	1	1	1	1
Y	D′	1	1	0	1	1	1	D
8-to-1 MUX data input	$D_0 = D'$	$D_1 = 1$	$D_2 = 1$	$D_3 = 0$	$D_4 = 1$	$D_5 = 1$	$D_6 = 1$	$D_7 = D$

(a)



(b)

Figure 4.5: A four variable truth table realization using 8:1 multiplexer.

NIBBLE MULTIPLEXERS

- Nibble means 4 bits like 0000, 0001, 0010 etc.
- Nibble multiplexer is used to select one out of two input nibbles i.e. $A_3A_2A_1A_0$ or $B_3B_2B_1B_0$.
- SELECT is a control signal.
- SELECT determines which input nibble is transmitted to output (Fig: 4.6).
- Here's how it works:
  - When **SELECT=low**, the four NAND gates on the left are activated. .'. $Y_3Y_2Y_1Y_0=A_3A_2A_1A_0$
  - When **SELECT=high**, the four NAND gates on the right are activated. .'. $Y_3Y_2Y_1Y_0=B_3B_2B_1B_0$

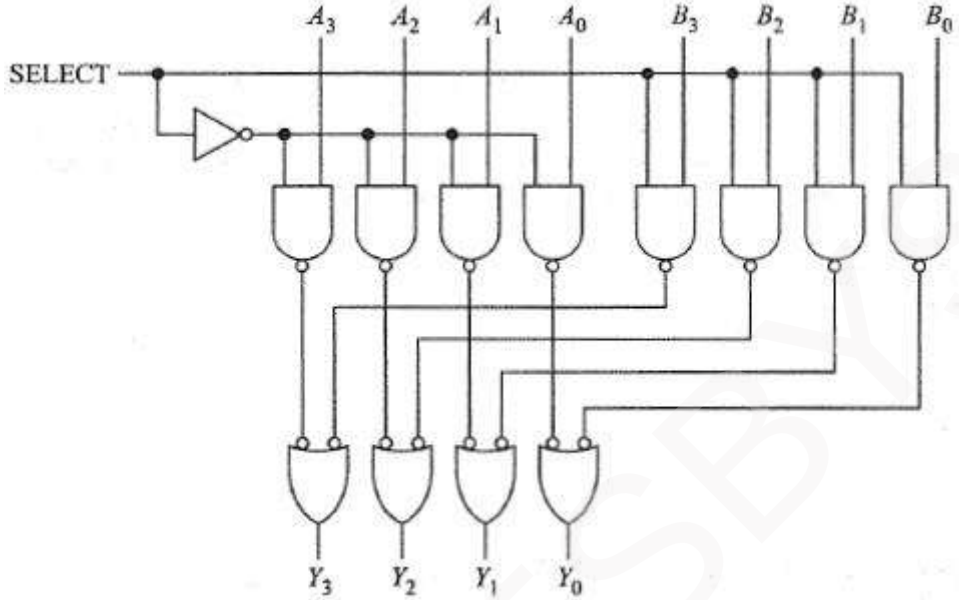


Figure: 4.6: Nibble Multiplexers

- The 74157 includes a strobe input (Figure: 4.7).
- The strobe must be low for the multiplexer to work properly.
- When the strobe is high, the multiplexer is inoperative.

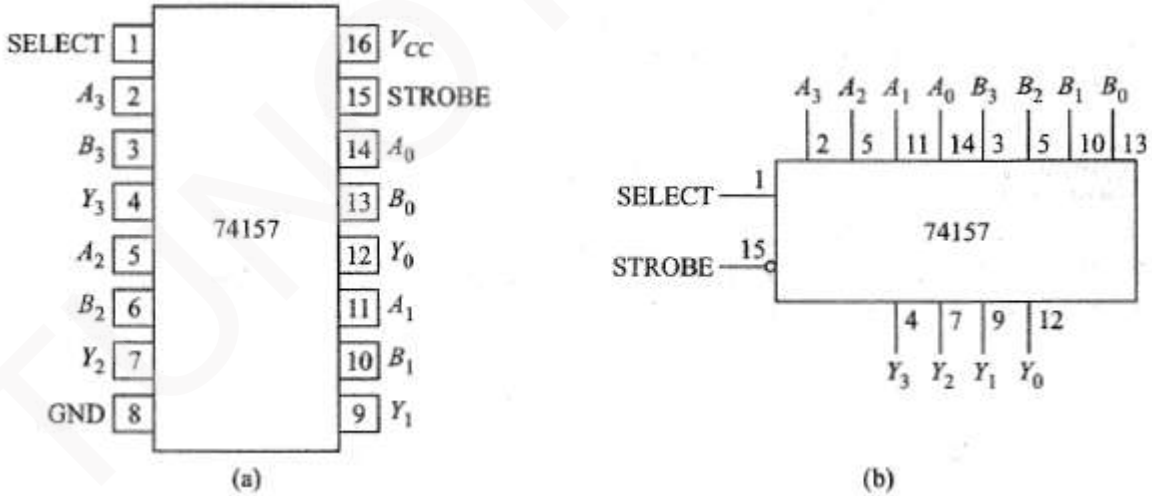


Figure: 4.7: Pinout diagram of 74157

EXAMPLE 4.1

Show how 4-to-l multiplexer can be obtained using only 2-to-l multiplexer.

Solution

Logic equation for 2-to-1 Multiplexer: $Y = A'.D_0 + A.D_1$

Logic equation for 4-to-1 Multiplexer: $Y = A'B'.D_0 + A'B.D_1 + AB'.D_2 + AB.D_3$

This can be rewritten as, $Y = A'(B'.D_0 + B.D_1) + A(B'.D_2 + B.D_3)$

Compare this with equation of 2-to-1 multiplexer. We need two 2-to-1 multiplexer to realize two bracketed terms where B serves as select input. The output of these two multiplexers can be sent to a third multiplexer as data inputs where A serves as select input and we get the 4-to-1 multiplexer. Figure 4.8a shows circuit diagram for this.

EXAMPLE 4.2

(a) Realize Y=A'B+B'C'+ABC using an 8-to-1 multiplexer. (b) Can it be realized with a 4-to-1 multiplexer?

Solution

(a) First we express Y as a function of minterms of three variables. Thus

$$Y = A'B + B'C' + ABC$$
$$Y = A'B(C' + C) + B'C'(A' + A) + ABC \text{ [As, } X + X' = 1]$$
$$Y = A'B'C' + A'BC' + A'BC + AB'C' + ABC$$

Comparing this with equation of 8 to 1 multiplexer, we find by substituting $D_0 = D_2 = D_3 = D_4 = D_7 = 1$ and $D_1 = D_5 = D_6 = 0$ we get given logic relation.

(b) Let variables A and B be used as selector in 4 to 1 multiplexer and C fed as input. The 4-to-1 multiplexer generates 4 minterms for different combinations of AB. We rewrite given logic equation in such a way that all these terms are present in the equation.

$$Y = A'B + B'C' + ABC$$
$$Y = A'B + B'C'(A' + A) + ABC \text{ [As, } X + X' = 1]$$
$$Y = A'B'.C' + A'B.1 + AB'.C' + AB.C$$

Compare above with equation of a 4-to-1 multiplexer. We see $D_0 = C'$, $D_1 = 1$, $D_2 = C'$ and $D_3 = C$ generate the given logic function.

EXAMPLE 4.3

Design a 32-to-l multiplexer using two 16-to-l multiplexers and one 2-to-l multiplexer.

Solution The circuit diagram is shown in Fig. 4.8b. A 32-to-1 multiplexer requires $\log_2 32 = 5$ select lines say, $ABCDE$. The lower 4 select lines $BCDE$ chose 16-to-1 multiplexer outputs. The 2-to-1 multiplexer chooses one of the output of two 16-to-1 multiplexers depending on what appears in the 5th select line, A.

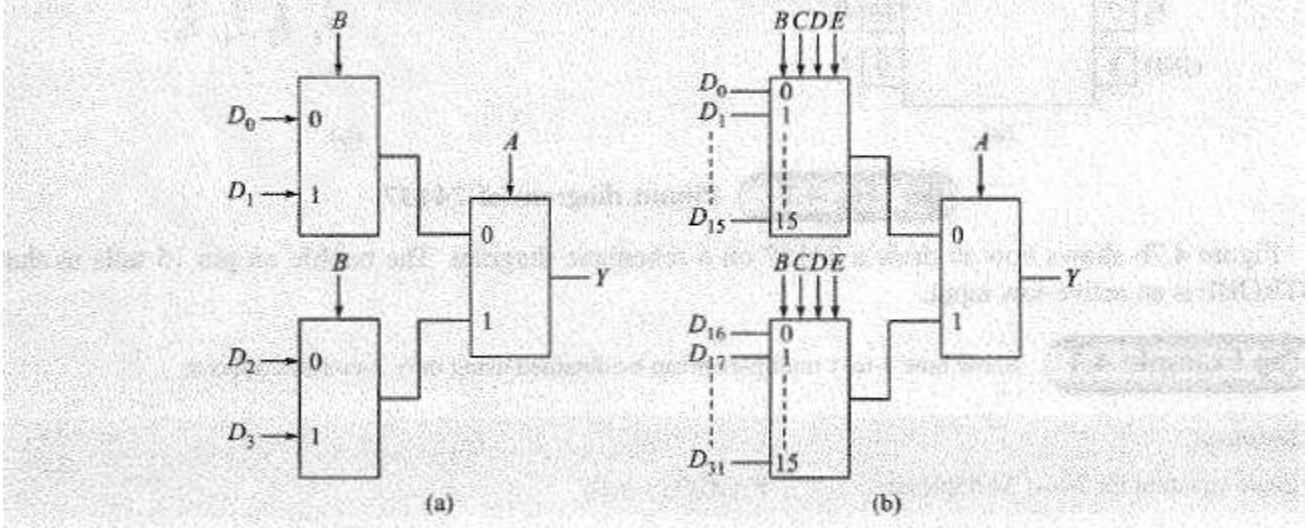
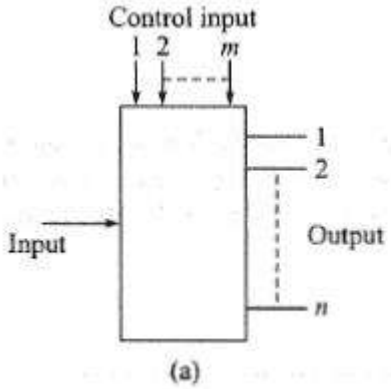
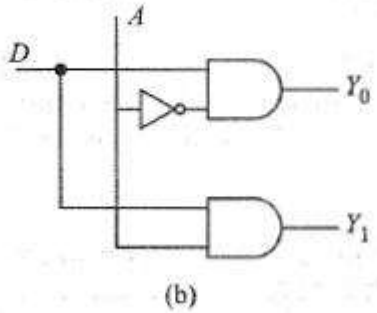


Figure 4.8: Realization of higher order multiplexers using lower orders

DEMULTIPLEXER

- It is a digital circuit with 1 input and many outputs (Figure: 4.9).
- By applying control signals, we can steer the input signal to one of the output lines.





IC No.	DEMUX Type	Decoder Type
74154	1-to-16	4-to-16
74138	1-to-8	3-to-8
74155	1-to-4	2-to-4

(c)

Figure 4.9: (a) Demultiplexer block diagram, (b) Logic circuit of 1-to-2 demultiplexer, (c) few commercially available ICs

1:16 Demultiplexer

- 1:16 Demux has
  - → 1 data input (D)
  - → 16 data output (Y_0 to Y_{15}) and
  - → 4 select inputs (A, B, C, D)
- The input data bit (D) is transmitted to one of the output lines. But which one? Again, this depends on the control-input ABCD.
- Here's how it works:
  - ➢ When ABCD=0000, the upper AND gate is enabled while other AND gates are disabled.
    Therefore, data bit D is transmitted only to the Y_0 output, giving Y_0=D.
    If D=low, Y_0=low. If D=high, Y_0=high.
  - ➢ When ABCD=1111, the bottom AND gate is enabled while other AND gates are disabled.
    Therefore, data bit D is transmitted only to the Y_{15} output, giving Y_{15}=D.

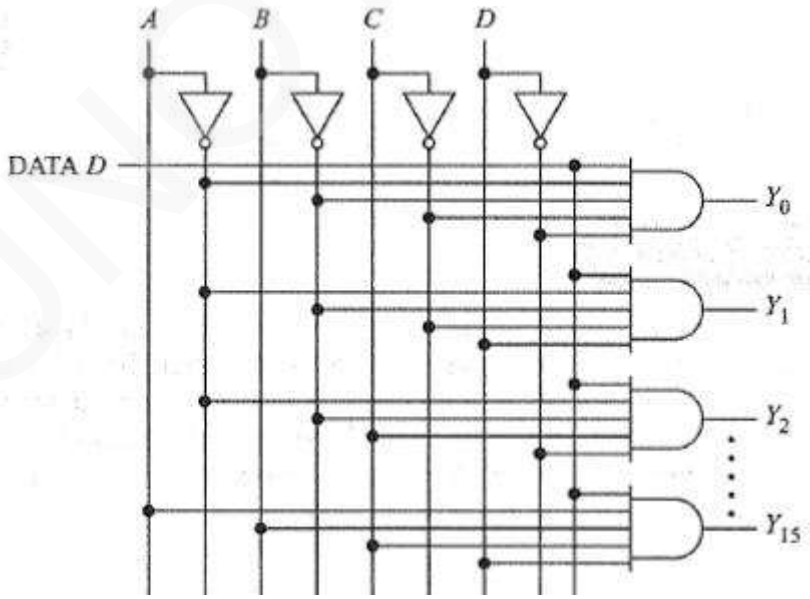


Figure 4.10: 1-to-16 demultiplexer

74154

• 74154 is a 1-to-16demultiplexer

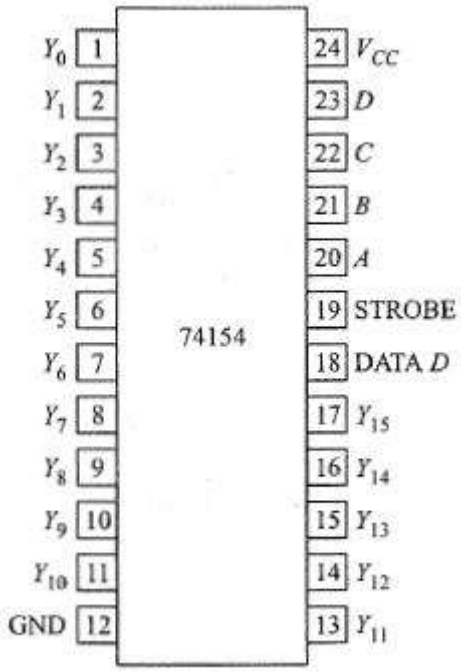


Figure 4.11: Pinout diagram of 74154

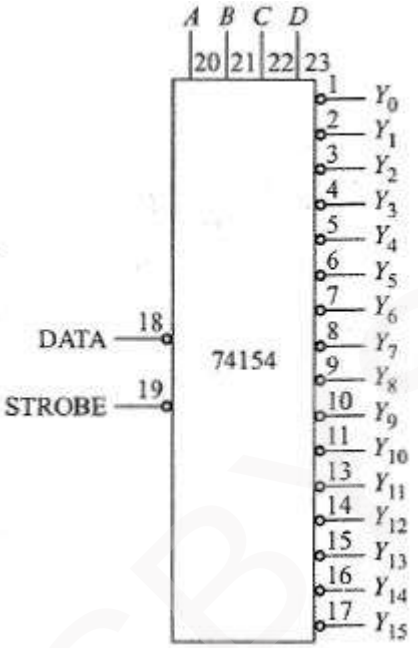


Figure 4.12: Logic diagram of 7 4154

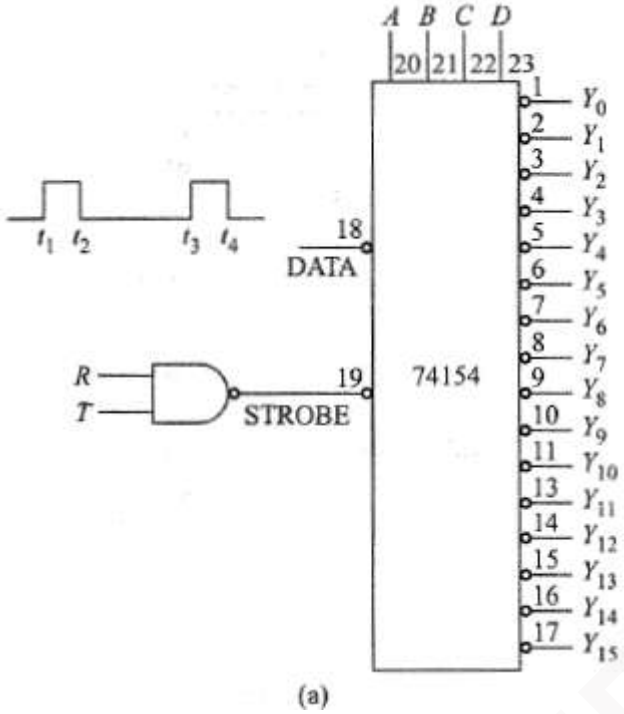
Strobe	Data	A	B	C	D	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7	Y_8	Y_9	Y_{10}	Y_{11}	Y_{12}	Y_{13}	Y_{14}	Y_{15}
L	L	L	L	L	L	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
L	L	L	L	L	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H
L	L	L	L	H	L	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H
L	L	L	L	H	H	H	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H
L	L	L	H	L	L	H	H	H	H	L	H	H	H	H	H	H	H	H	H	H	H
L	L	L	H	L	H	H	H	H	H	H	L	H	H	H	H	H	H	H	H	H	H
L	L	L	H	H	L	H	H	H	H	H	H	L	H	H	H	H	H	H	H	H	H
L	L	L	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H	H	H	H	H
L	L	H	L	L	L	H	H	H	H	H	H	H	H	L	H	H	H	H	H	H	H
L	L	H	L	L	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H	H	H
L	L	H	L	H	L	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H	H
L	L	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H
L	L	H	H	L	L	H	H	H	H	H	H	H	H	H	H	H	H	L	H	H	H
L	L	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	L	H	H
L	L	H	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	L	H
L	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	L
L	H	X	X	X	X	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
H	L	X	X	X	X	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
H	H	X	X	X	X	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H

Table 4.3: 74154 Truth table

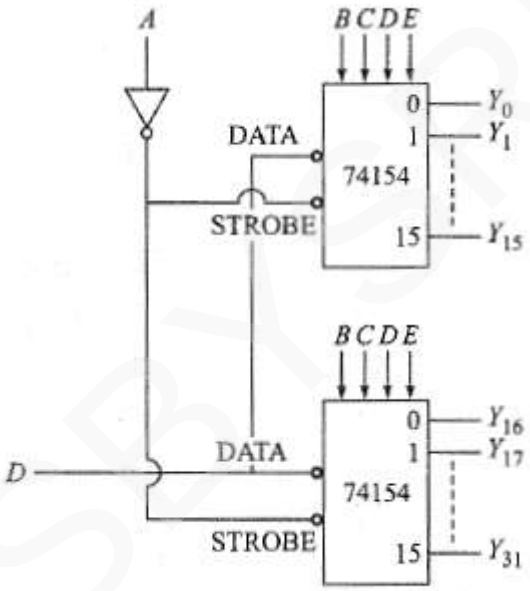
EXAMPLE 4.4

Show how two 1-to-16 demultiplexers can be connected to get a l-to-32 demultiplexer.

Solution Figure 4.13b shows the circuit diagram. A 1-to-32 demultiplexer has 5 select variables *ABCDE*. Four of them (*BCDE*) are fed to two 1-to-16 demultiplexer. And the fifth (*A*) is used to select one of these two multiplexer through strobe input. If $A = 0$, the top 714154 is chosen and *BCDE* directs data to one of the 15 outputs of that IC. If $A = 1$, the bottom IC is chosen and depending on value of *BCDE* data is directed to one of the 15 outputs this IC.



(a)



(b)

DECODERS

• It is a multiple-input, multiple-output logic circuit which converts coded inputs into coded outputs, where the input and output codes are different (Figure: 4.14).

• It is similar to a demultiplexer with one exception: there is no data input. The only inputs are the control bits.

1-of-16 Decoder

• 1-of-16 decoder is called so because only 1 of the 16 output lines is high.

• Here's how it works (Figure 4.14):

- When ABCD=0001, only the Y_1 AND gate has all inputs high, therefore only the Y_1 output is high.
- When ABCD=0100, only the Y_4 AND gate has all inputs high, therefore only the Y_4 output is high.

• The circuit is also called a **binary-to-decimal decoder**.

• The circuit is also called as a **4-line to 16-line decoder**, since it has 4 input lines & 16 output lines.

• **74154** is called a **decoder-demultiplexer**, because it can be used either as a decoder or as a demultiplexer (Figure 4.15).

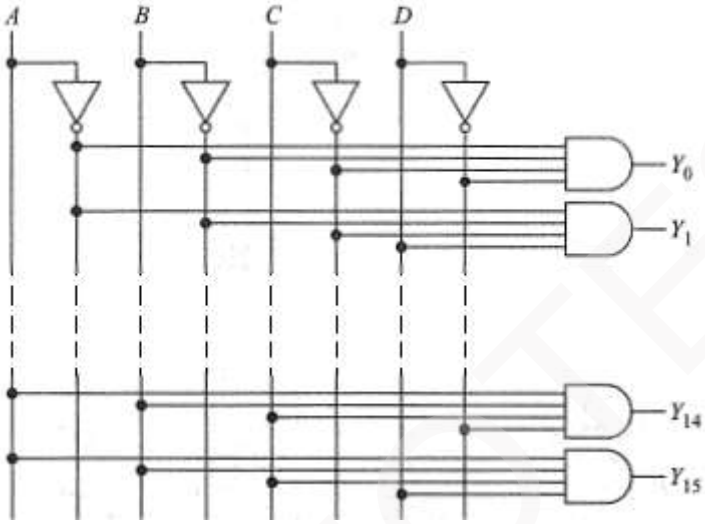


Figure 4.14: 1-of-16 decoder

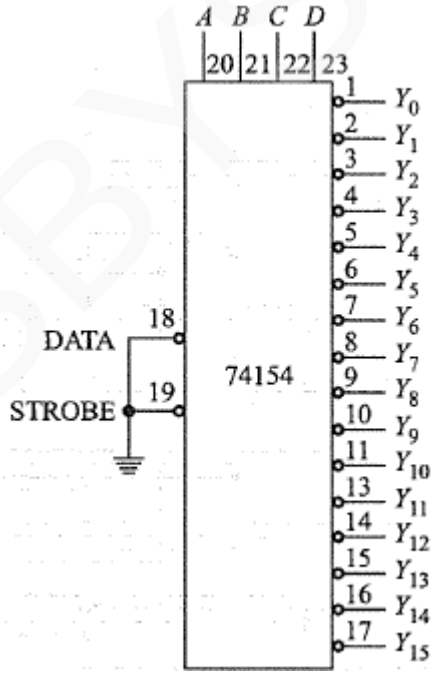


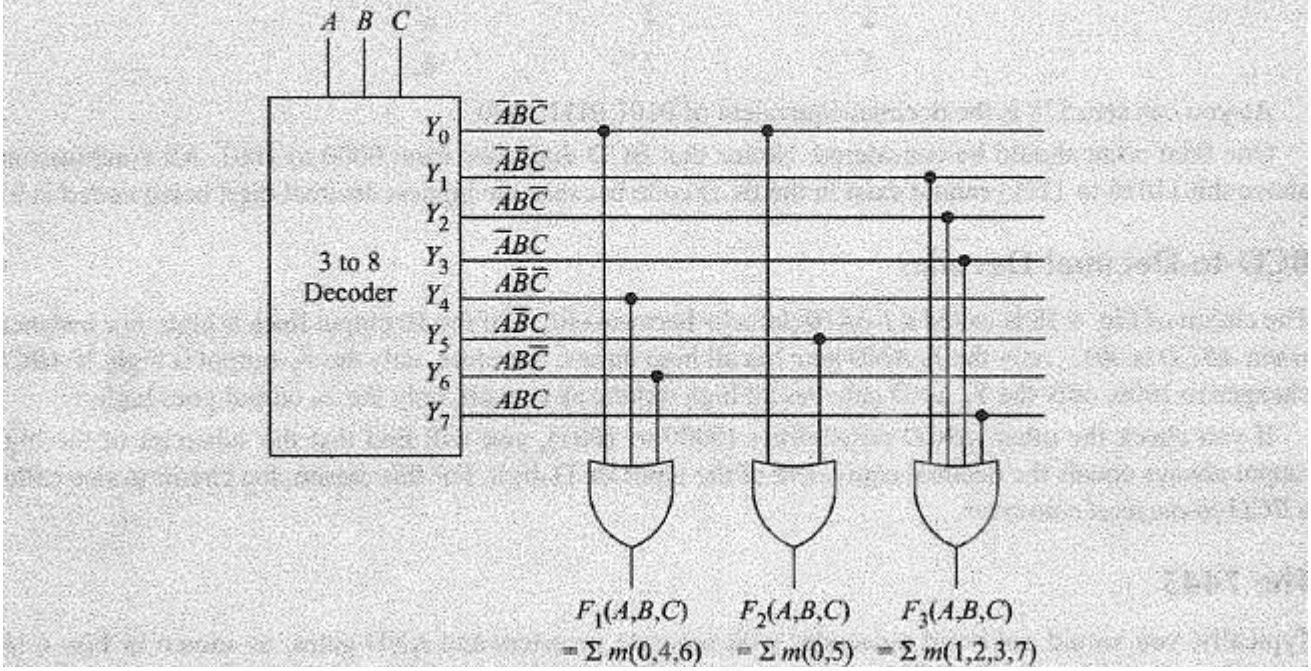
Figure 4.15: Using 74154 as decoder

Example 4.5

Show how using a 3-to-8 decoder and multi-input OR gates following Boolean expressions can be realized simultaneously.

F1(A,B,C)=∑m(0,4,6); F2(A, B, C)=∑m(0,5); F2(A,B,C)=∑m(1,2,3,7)

Solution Since at the decoder output we get all the minterms we use them as shown in Fig. 4.17 to get the required Boolean functions.



BCD-TO-DECIMAL DECODERS

- BCD is an abbreviation for binary-coded decimal.
- The BCD code expresses each digit in a decimal number by its nibble equivalent.
- BCD digits are from 0000 to 1001.
- All combinations above this (1010 to 1111) cannot exist in the BCD code because the highest decimal digit being coded is 9.
- For example:

DECIMAL	4	2	9
↓	↓	↓	↓
BCD	0100	0010	1001

BCD	0101	0111	1000
↓	↓	↓	↓
DECIMAL	5	7	8

- Application:
  - Some early computers processed BCD numbers.
  - This means that the decimal numbers were changed into BCD numbers, which the computer then added, subtracted, etc.
  - The final answer was converted from BCD back to decimal numbers.

BCD-to-Decimal Decoder

- 1-of-10 decoder is called so because only 1 of the 10 output lines is high.
- Here's how it works (Figure 4.14):
  - When ABCD=0001, only the Y_1 AND gate has all inputs high, therefore only the Y_1 output is high.
  - When ABCD=0100, only the Y_4 AND gate has all inputs high, therefore only the Y_4 output is high.
- The circuit is also called a **BCD-to-decimal converter.**
- **7445** is called a BCD-to-Decimal decoder (Figure 4.19).

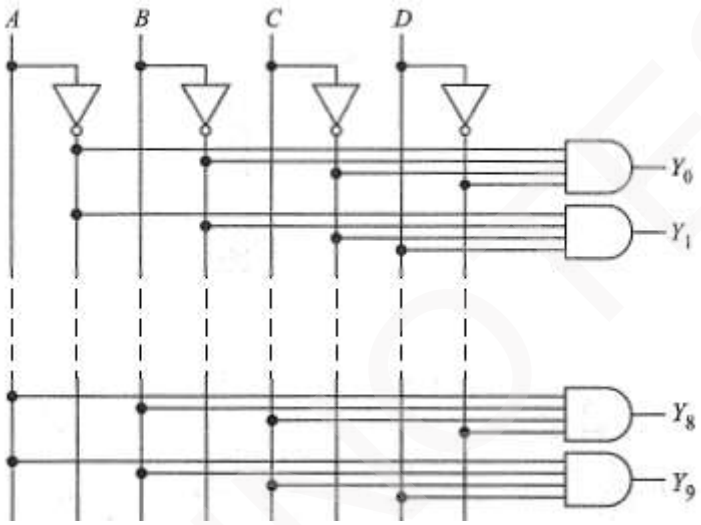


Figure 4.18: 1-of-10 decoder

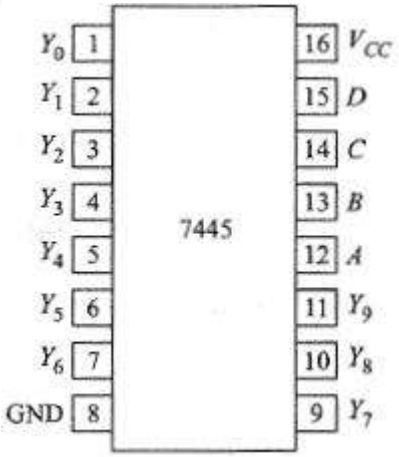


Figure 4.19: Pinout diagram of 7445

	Inputs				Outputs									
No.	A	B	C	D	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7	Y_8	Y_9
0	L	L	L	L	L	H	H	H	H	H	H	H	H	H
1	L	L	L	H	H	L	H	H	H	H	H	H	H	H
2	L	L	H	L	H	H	L	H	H	H	H	H	H	H
3	L	L	H	H	H	H	H	L	H	H	H	H	H	H
4	L	H	L	L	H	H	H	H	L	H	H	H	H	H
5	L	H	L	H	H	H	H	H	H	L	H	H	H	H
6	L	H	H	L	H	H	H	H	H	H	L	H	H	H
7	L	H	H	H	H	H	H	H	H	H	H	L	H	H
8	H	L	L	L	H	H	H	H	H	H	H	H	L	H
9	H	L	L	H	H	H	H	H	H	H	H	H	H	L

Table 4.4: 7445 Truth Table

EXAMPLE 4.6

The decoded outputs of a 7445 can be connected to light-emitting diodes (LEDs), as shown in Fig. 4.20. If each resistance is 1kQ and each LED has a forward voltage drop of2 V, how much current is there through a LED when it is conducting? (See Chapter 13 for a discussion of LEDs.)

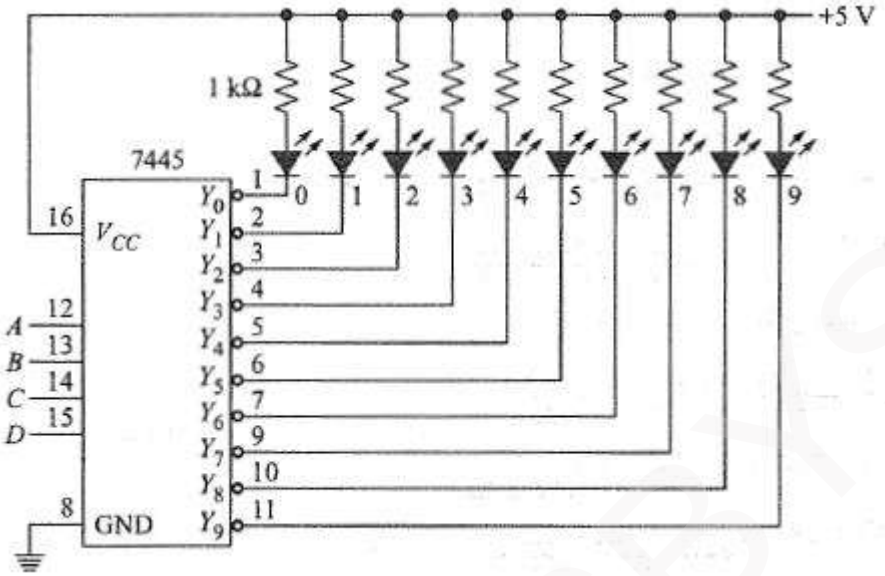


Figure: 4.20

Solution When an output is in the low state, you can approximate the output voltage as zero. Therefore, the current through a LED is

$$I = \frac{5\text{ V} - 2\text{ V}}{1\text{ k}\Omega} = 3\text{ mA}$$

EXAMPLE 4.7

The LEDs of Fig. 4.20 are numbered O through 9. Which of the LEDs is lit for each of the following conditions

a. ABCD=OIOI.

b. ABCD = 1001.

c. ABCD = 1100

Solution

a. When $ABCD = 0101$, the decoded output line is Y_5. Since Y_5 is approximately grounded, LED 5 lights up. All other LEDs remain off because the other outputs are high.

b. When $ABCD = 1001$, LED 9 is on:

c. $ABCD = 1100$ is an invalid input. Therefore, none of the LEDs is on because all output, lines are high (see Table 4.4).

SEVEN-SEGMENT DECODERS

- It has seven LEDs labeled a through g.
- By forward-biasing different LEDs, we can display the digits O through 9 (Fig. 4.21b).
- For example, to display a 7, we need to light up segments a, b and c. (Fig. 4.21a).

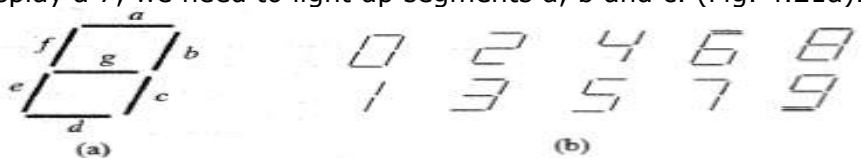


Figure 4.21: Seven-segment indicator

- Seven-segment indicators may be
  1) Common-anode type where all anodes are connected together (Fig. 4.22a) or
  2) Common-cathode type where all cathodes are connected together (Fig. 4.22b).

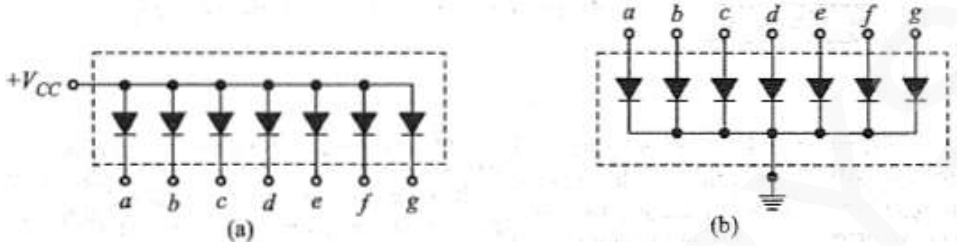


Figure 4.22: (a) Common-anode type, (b) Common-cathode type

- In the common anode type
  - A current limiting resistor is connected between each LED and ground.
  - The size of this resistor determines how much current flows through the LED.
- In the common-cathode type
  - A current limiting resistor is connected between each LED and $+V_{cc}$.

7446

- A seven-segment decoder-driver is an IC decoder that can be used to drive a seven-segment indicator (Fig. 4.23a).
- Each decoder-driver has
  → 4 input pins (the BCD input) and
  → 7 output pins (the a through g segments).
- Logic circuits inside the 7446 convert the BCD input to the required output.
  - For example, if the BCD input is 0111, the logic circuits will force LEDs a, b, and c to conduct.
  - As a result, digit 7 will appear on the seven-segment indicator.

7448

- Here, a 7448 drives a common-cathode indicator (Fig. 4.23b).
- The 7448 has its own current-limiting resistors on the chip.

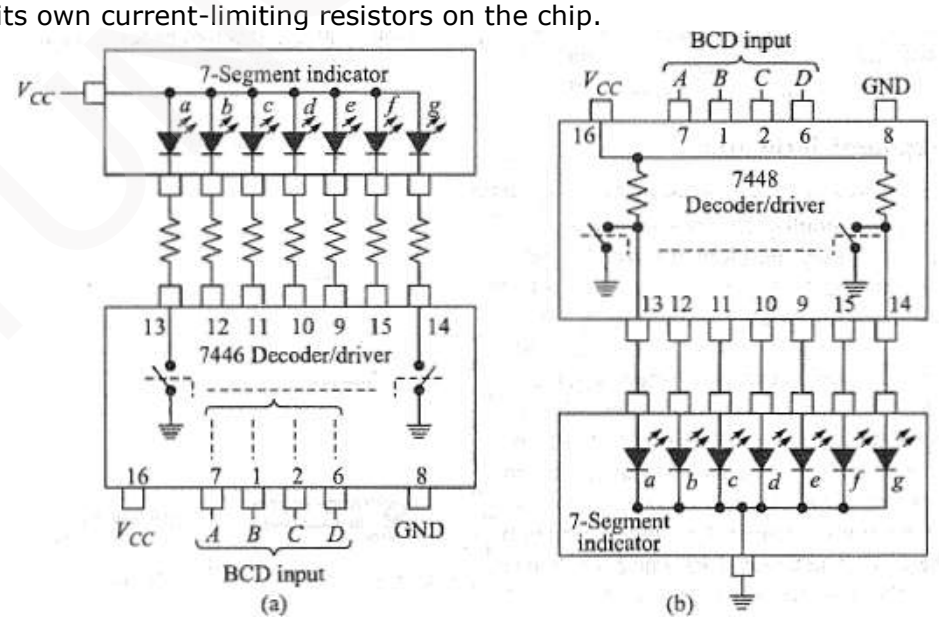


Figure 4.23: (a) 7446 decoder-driver, (b) 7448 decoder-driver

ENCODERS

- It converts an active input signal into a coded signal.
- As shown in Figure 4.24, there are 'n' input lines, only one of which is active.
- Internal logic within the encoder converts 'n' active input to a coded binary output with 'm' bits.

Decimal-to-BCD Encoders

- The switches are push-button switches like those of a pocket calculator (Figure: 4.24).
- Here's how it works (Figure 4.14):
  - When button-3 is pressed, the C and D OR gates have high inputs, .'. output ABCD=0011
    If button-5 is pressed, output ABCD=0101
    If button-9 is pressed, output ABCD=1001

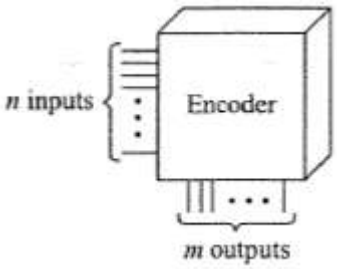


Figure: 4.24: Encoders

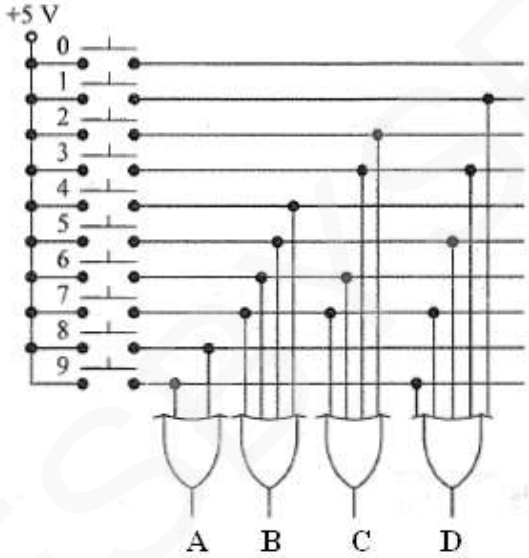


Figure 4.25: Decimal-to-BCD encoder

74147

- 74147 is called a decimal-to-BCD encoder (Figure 4.25a).
- 74147 is called a priority encoder because it gives priority to the highest-order input.

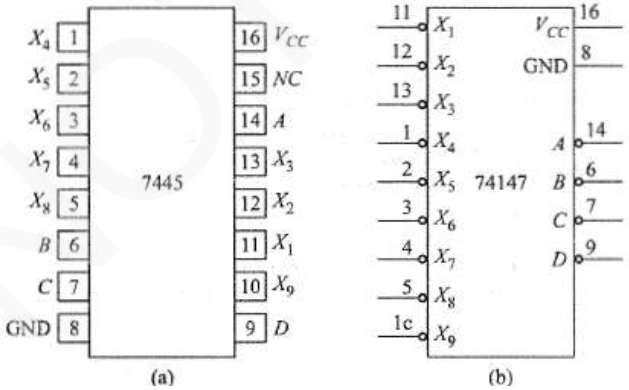


Figure 4.25: (a) Pinout diagram of 74147, (b) logic diagram

Inputs									*Outputs*			
X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	A	B	C	D
H	H	H	H	H	H	H	H	H	H	H	H	H
X	X	X	X	X	X	X	X	L	L	H	H	L
X	X	X	X	X	X	X	L	H	L	H	H	H
X	X	X	X	X	X	L	H	H	H	L	L	L
X	X	X	X	X	L	H	H	H	H	L	L	H
X	X	X	X	L	H	H	H	H	H	L	H	L
X	X	X	L	H	H	H	H	H	H	L	H	H
X	X	L	H	H	H	H	H	H	H	H	L	L
X	L	H	H	H	H	H	H	H	H	H	L	H
L	H	H	H	H	H	H	H	H	H	H	H	L

Table 4.5: 74147 Truth Table

EXAMPLE 4.8

What is the ABCD output of Fig. 4.27 when button 6 is pressed?

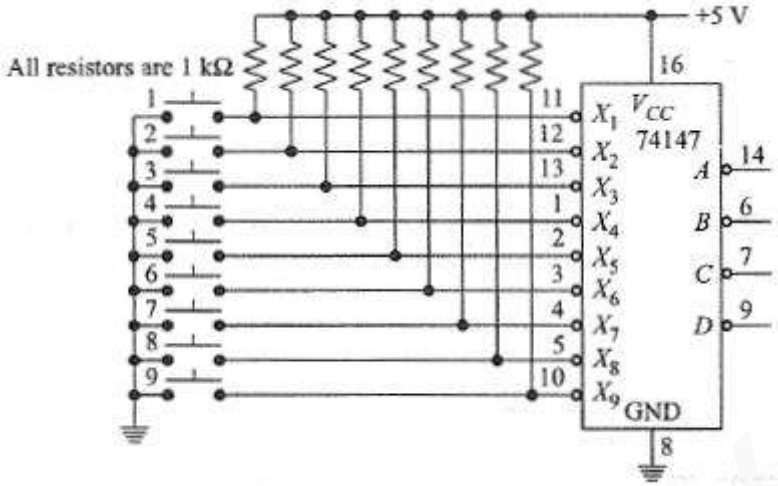


Figure 4.27

Solution When all switches are open, the X_1 to X_9 inputs are pulled up to the high state (+5 V). A glance at Table 4.5 indicates that the *ABCD* output is *HHHH* at this time.

When switch 6 is pressed, the X_6 input is grounded. Therefore, all X inputs are high, except for X_6. Table 4.5 indicates that the *ABCD* output is *HLLH*, which is equivalent to 6 when the output bits are complemented.

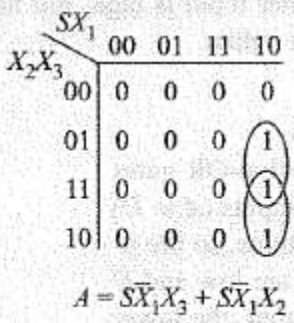
EXAMPLE 4.9

Design a priority encoder the truth table of which is shown in Fig. 4.28a. The order of priority for three inputs is X1 > X2 > X3. However, if the encoder is not enabled by Sor all the inputs are inactive the output AB= 00.

Solution Figure 4.28b and Fig. 4.28c show the Karnaugh map for output A and B respectively. Note that, we have used a different notation for input variables in these maps. Compare this with notations presented in previous chapters. You will find a variable with prime is presented by 0 and if it is not primed is represented by 1. Then taking groups of 1s we get the design equations as shown in the figure. The logic circuits for output A and B can be directly drawn from these equations.

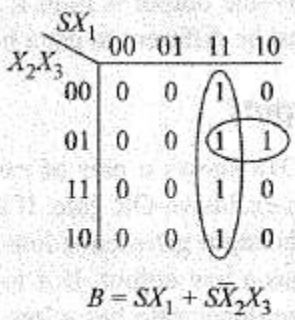
Intput				Output	
S	X_1	X_2	X_3	A	B
0	×	×	×	0	0
1	1	×	×	0	1
1	0	1	×	1	0
1	0	0	1	1	1
1	0	0	0	0	0

(a)



$A = S\bar{X}_1X_3 + S\bar{X}_1X_2$

(b)



$B = SX_1 + S\bar{X}_2X_3$

(c)

Figure 4.28: Design of a priority encoder

EXCLUSIVE OR GATES

• This has a high output only when an odd number of inputs is high (Figure: 4.29 & 4.30).

• The upper AND gate forms the product A'B, while the lower one produces AB'. Therefore, the output of the OR gate is

Y=A'B+AB'

• This gate always produces an output 1 only when n-bit input has an odd number of 1s (Table 4.6).

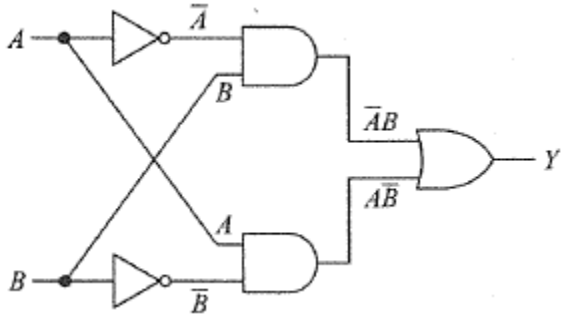


Figure 4.29: Exclusive OR gate

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

Table 4.6: Two input Exclusive OR gate

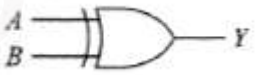


Figure 4.30: Logic symbol

PARITY GENERATORS AND CHECKERS

- Even-parity means an n-bit input has an even number of 1s.
  For e.g. 110011 has even parity because it contains four 1s.
- Odd parity means an n-bit input has an odd number of 1s.
  For e.g. 110001 have odd parity because it contains three 1s.
- In the transmitter, the circuit which generates the parity-bit is called a **parity generator**.
  In the receiver, the circuit which checks the parity is called a **parity checker**.

Parity Generation

- In a computer, a binary number may represent an instruction.
- In this case, you sometimes will see an extra bit added to the original binary number to produce a new binary number with odd parity (Figure 4.34).
- XOR gates can be used for generating the parity of a binary number.
- Here's how it works (Figure 4.33):
  - **Case 1:** Suppose $X_7X_6X_5X_4\ X_3X_2X_1X_0$=0100 0001.
    - Then, the number has even parity.
    - This means the XOR gate produces an output 0.
    - Because of the inverter, X_8=1.
    - The final 9-bit output is 1 0100 0001.
    - The final output has odd parity.
  - **Case 2:** Suppose $X_7X_6X_5X_4\ X_3X_2X_1X_0$=0110 0001.
    - Then, the number has odd parity.
    - This means the XOR gate produces an output 1.
    - Because of the inverter, X_8=0.
    - The final 9-bit output is 0 0110 0001.
    - Again, the final output has odd parity.
- The circuit is called an **odd-parity generator** because it always produces a 9-bit output number with odd parity.

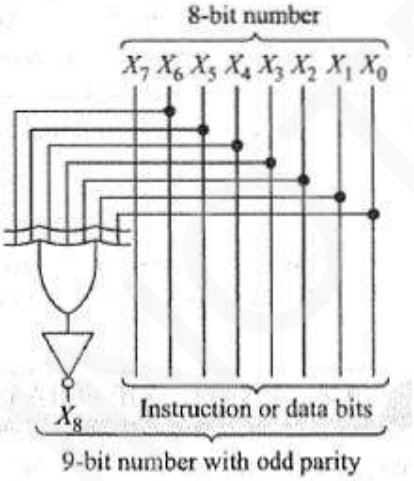


Figure 4.33: Odd parity generator

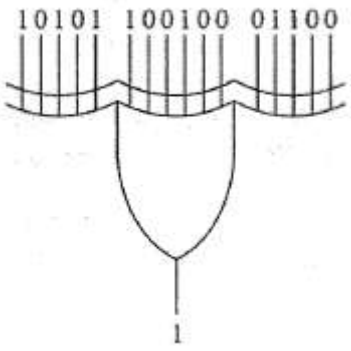


Figure 4.34: Odd parity Checker

Parity Checker

- XOR gates can be used for checking the parity of a binary number.
- XOR gates produce an output 1 when the input has an odd number of 1s.
- Here's how it works (Figure 4.34):
  - An even parity input to an exclusive OR gate produces a low output
  - An odd parity input to an exclusive OR gate produces a high output.

What is the Practical Application of Parity Generation & Checking?

- Because of noise and other disturbance, 1-bit errors may occur when binary data is transmitted over telephone lines.
- Odd parity generator & odd-parity checker can be used to check for errors.
- Basically, odd parity generator is used at the transmitting end.
  Odd-parity checker is used at the receiving end.
- If no errors occur in transmission, the received data will have odd parity.
  If 1-bit error occurs in transmission, the received data will have odd parity.

74180

- 74180 is a TTL parity generator-checker.
- The input data bits are X_7 to X_0; these bits may have even or odd parity.
- The even input (pin 3) and the odd input (pin 4) control the operation of the chip as shown in Table 4.8.
- The 74180 can be used to detect even or odd parity.
- The 74180 can also be set up to generate even or odd parity.

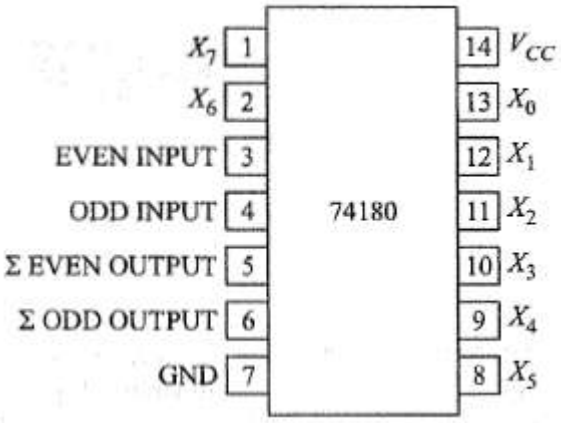


Figure 4.35: Pinout diagram of 7 4180

	Inputs		*Outputs*	
Σ of H's at X_7 to X_0	*Even*	*Odd*	*Σ even*	*Σ odd*
Even	*H*	*L*	*H*	*L*
Odd	*H*	*L*	*L*	*H*
Even	*L*	*H*	*L*	*H*
Odd	*L*	*H*	*H*	*L*
X	*H*	*H*	*L*	*L*
X	*L*	*L*	*H*	*H*

74180 Truth Table

EXAMPLE 4.10

Show how to connect a 74180 to generate a 9-bit output with odd parity.

Solution Figure 4.36 shows one solution. The ODD INPUT (pin 4) is connected to +5 V, and the EVEN INPUT (pin 3) is grounded. Suppose the input data $X_7 \ldots X_0$ has even parity. Then, the third entry of Table 4.8 tells us the Σ ODD OUTPUT (pin 6) is high. Therefore, the 9-bit number $X_8 \ldots X_0$ coming out of the circuit has odd parity.

On the other hand, suppose $X_7 \ldots X_0$ has odd parity. Then the fourth entry of Table 4.8 says that the Σ odd output is low. Again, the 9-bit number $X_8 \ldots X_0$ coming out at the bottom of Fig. 4.36 has odd parity.

The following conclusion may be drawn. Whether the input data has even or odd parity, the 9-bit number being generated in Fig. 4.36 always has odd parity.

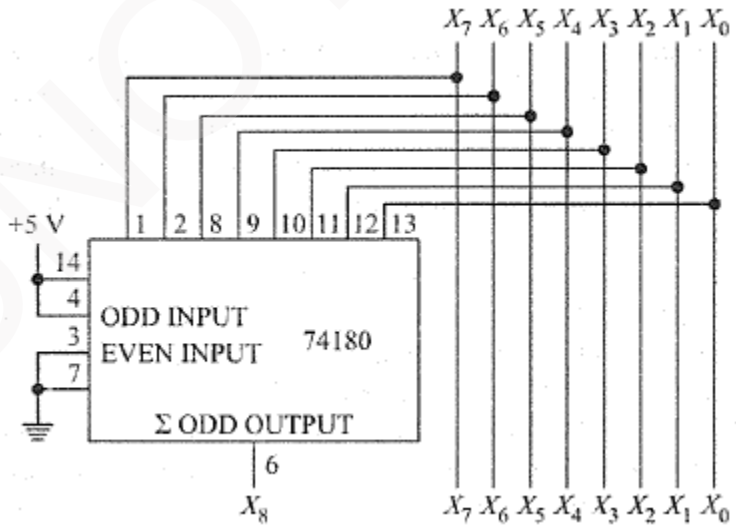


Figure 4.36: Using a 74180 to generate odd parity

MAGNITUDE COMPARATOR

- Magnitude comparator has
  - → 2 inputs
  - → 3 outputs
- Magnitude comparator
  - → compares two n-bit binary numbers, say A and B and
  - → activates one of these 3 outputs: X=Y, X>Y and X<Y (Figure: 4.37).
- The logic equations for the outputs can be written as follows:

  (X>Y): G=XY'

  (X<Y): L=X'Y

  (X=Y): E=X'Y'+XY=(XY'+X'Y)'=(G+L)'

where G→ greater than
L→ less than
E→equal to

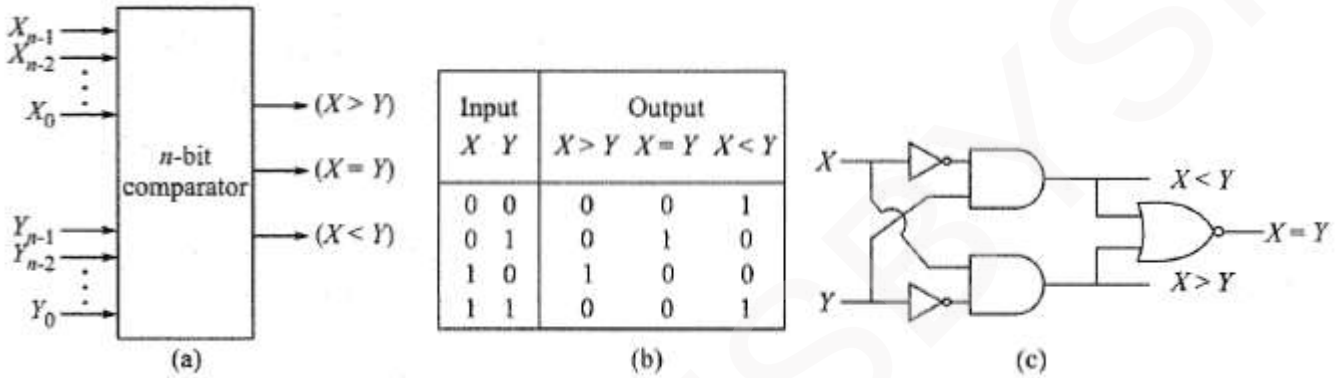


Figure 4.37: a)Block diagram of magnitude comparator b)Truth table c)Circuit for 1-bit comparator.

- How to design a 2-bit comparator?

Let's first define bit-wise greater than terms (G):	$G_1 = X_1 Y_1'$,	$G_0 = X_0 Y_0'$
Then, bit-wise less than term (L):	$L_1 = X_1' Y_1$,	$L_0 = X_0' Y_0$
Therefore, bit-wise equality term (E):	$E_1 = (G_1 + L_1)'$,	$E_0 = (G_0 + L_0)'$

- From above definitions we can easily write 2-bit comparator outputs as follows.

$(X = Y) = E_1.E_0 \quad (X > Y) = G_1 + E_1.G_0 \quad (X < Y) = L_1 + E_1.L_0$

7485

- 7485 is a 4-bit magnitude comparator.
- 7485 compares two 4-bit numbers.
- The circuit has three additional inputs in the form of

$(X = Y)_{in}$, $(X > Y)_{in}$ and $(X < Y)_{in}$

- These additional inputs are used when we need to connect more than one IC 7485 to compare numbers having more than 4-bits.
- When IC 7485 is not used in cascade, we keep

$(X = Y)_{in} = 1, (X > Y)_{in} = 0$ and $(X < Y)_{in} = 0$.

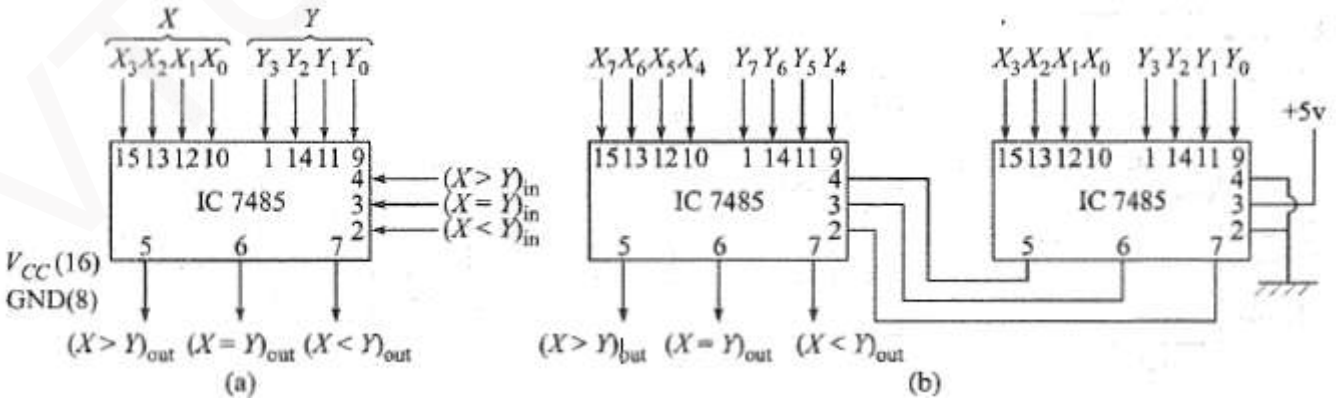
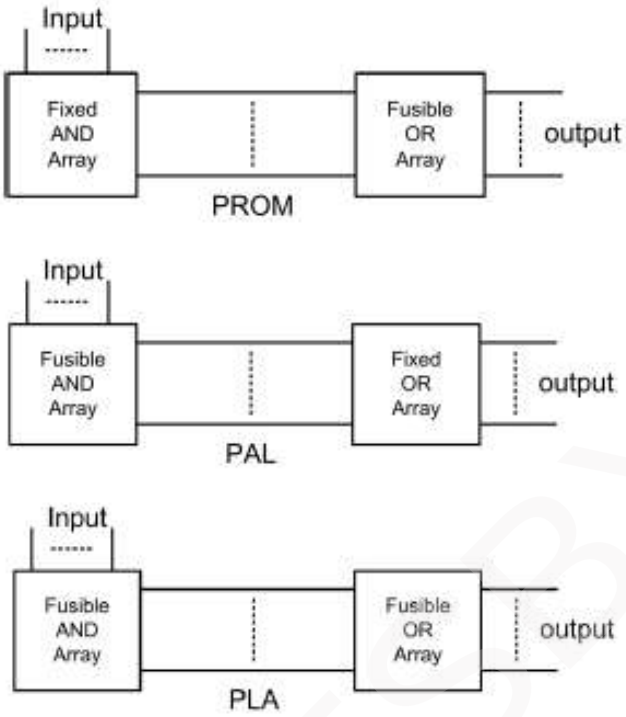


Figure 4.38: 8-bit comparator from two 4-bit comparators

READ-ONLY MEMORY

- It is an IC that can store thousands of binary numbers representing instructions and data.
- We can use a ROM instead of SOP circuits to generate any boolean function.
- The block diagrams of the three PLDs (Programmable Logic Device) are as shown below:



PROGRAMMABLE ROMS

- This allows the user instead of the manufacturer to store the data.
- An instrument called a **PROM programmer** stores the words by "burning in".
- Programming like this is permanent because the data cannot be erased after it has been burned in.

Q: Why PROM is a Universal Logic Solution?

Ans: Because the AND gates generate all the fundamental products. The user can then OR these products as needed to generate any boolean output.

PROGRAMMABLE ARRAY LOGIC (PAL)

• This is a programmable array of logic gates on a single chip.

• PALs are another design solution, similar to a sum-of-products solution, product-of-sums solution, and multiplexer logic.

• This is different from a PROM because it has a programmable AND array and a fixed OR array.

• With a PROM programmer, we can burn in the desired fundamental products, which are then ORed by the fixed output connections.

• As shown in Figure 4.43, there 2 types of links: 1) Fusible links & 2) Fixed links.

1) x's on the input side is fusible links,
2) Solid black bullets(•) on the output side are fixed connections.

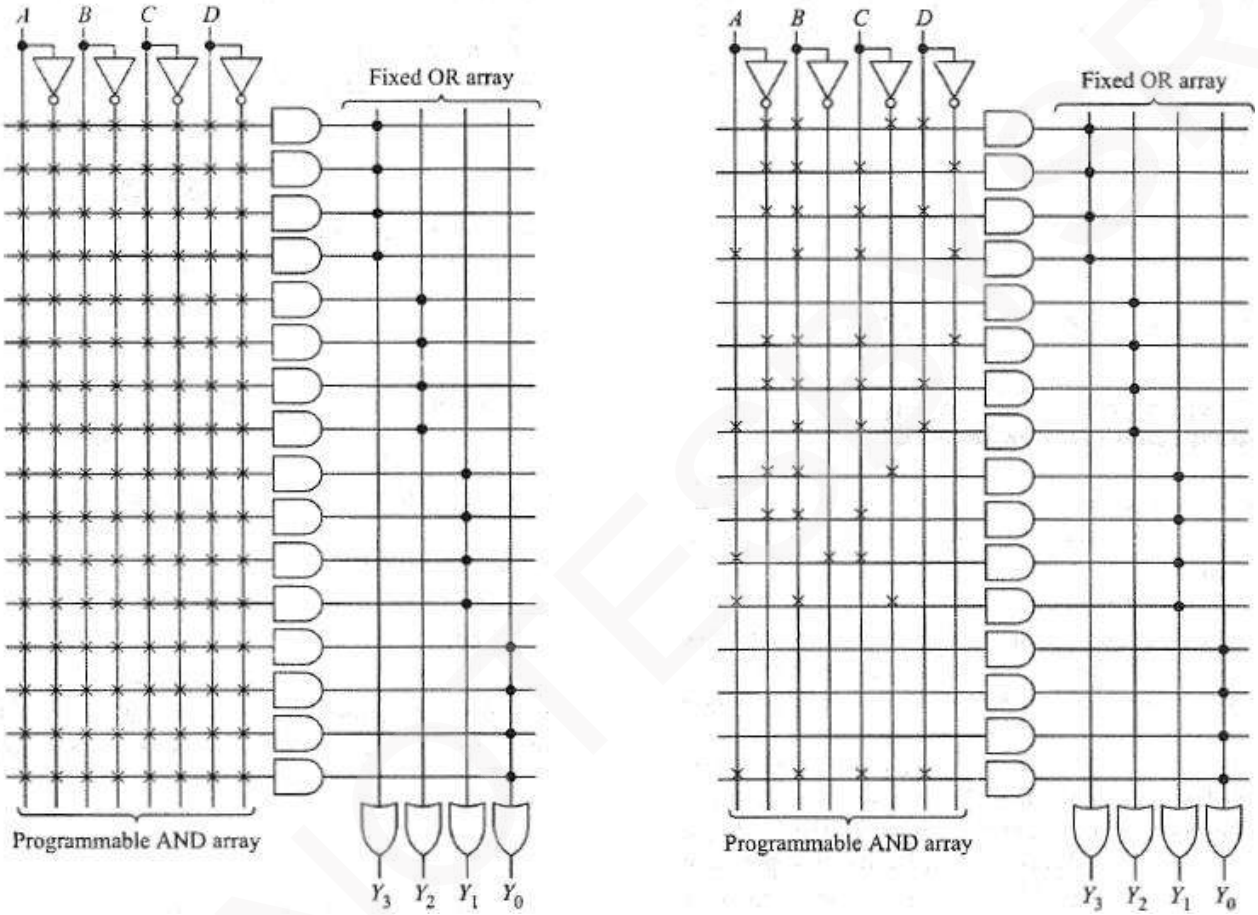


Figure 4.43: Structure of PAL

• Suppose we want to generate the following boolean functions. Show how we can program a PAL.

$$Y_3 = \bar{A}B\bar{C}D + \bar{A}BC\bar{D} + \bar{A}BCD + ABC\bar{D}$$
$$Y_2 = \bar{A}BC\bar{D} + \bar{A}BCD + ABCD$$
$$Y_1 = \bar{A}B\bar{C} + \bar{A}BC + A\bar{B}C + AB\bar{C}$$
$$Y_0 = ABCD$$

PROGRAMMABLE LOGIC ARRAYS (PLA)

• In a PROM, the input AND gate array is fixed and cannot be altered, while the output OR gate array is fusible linked, and can thus be programmed.

In PAL, the output OR gate array is fixed while the input AND gate array is fusible linked and thus programmable.

• Advantage: The PLA is much more versatile than the PROM or the PAL, since both its AND gate array and its OR gate are fusible linked and programmable.

• Drawback: PLA is also more complicated to utilize since the number of fusible links are doubled.

• In this, the input signals are presented to an array of AND gates while the outputs are taken from an array of OR gates.

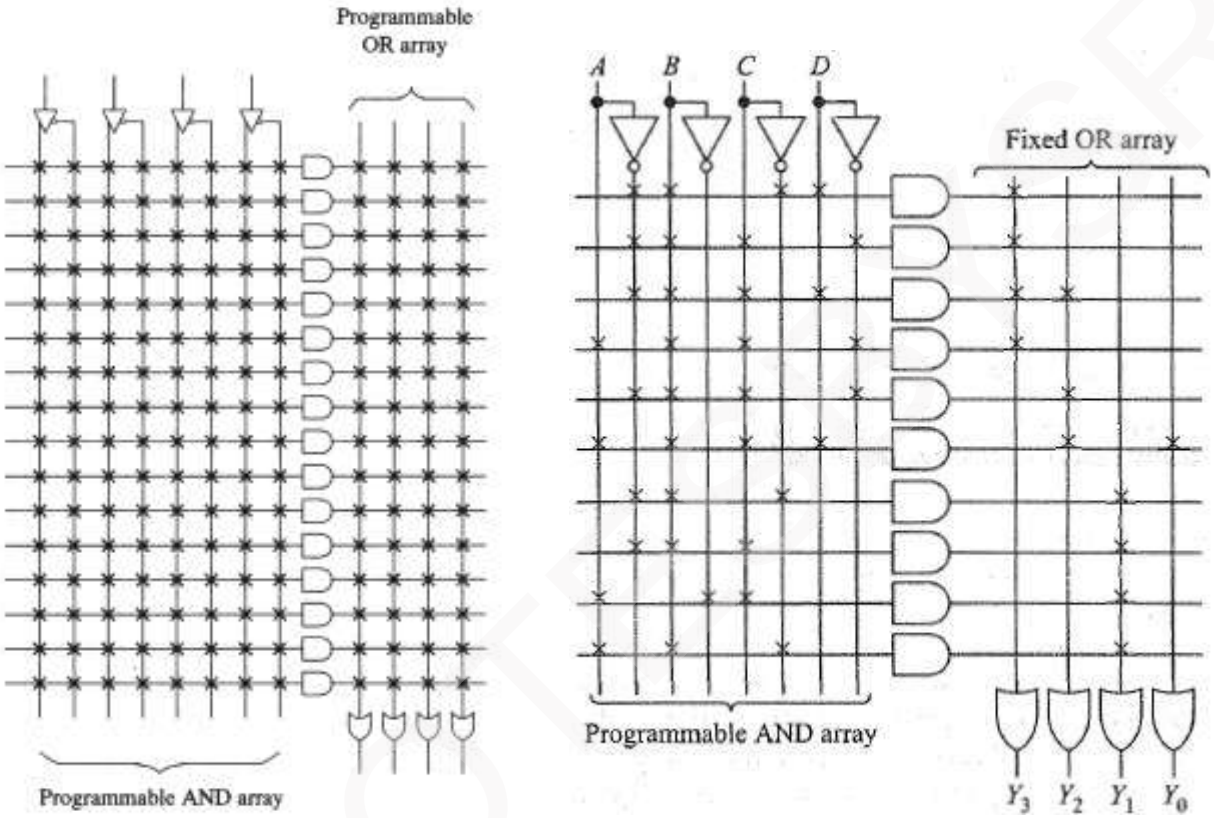
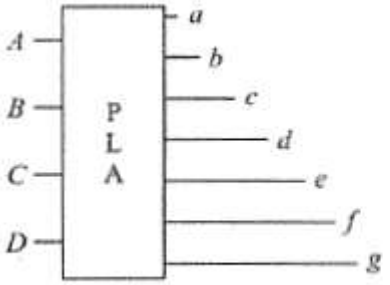


Figure 4.45: Structure of PLA

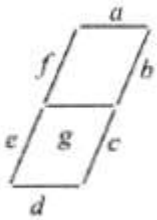
• Suppose we want to generate the following boolean functions. Show how we can program a PLA.

$$Y_3 = \bar{A}B\bar{C}D + \bar{A}BC\bar{D} + \bar{A}BCD + ABC\bar{D}$$
$$Y_2 = \bar{A}BC\bar{D} + \bar{A}BCD + ABCD$$
$$Y_1 = \bar{A}B\bar{C} + \bar{A}BC + A\bar{B}C + AB\bar{C}$$
$$Y_0 = ABCD$$

• Show how we can use a PLA to recognize each of the 10 decimal digits represented in binary form and to correctly drive a 7-segment display.



(a)



(c)

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

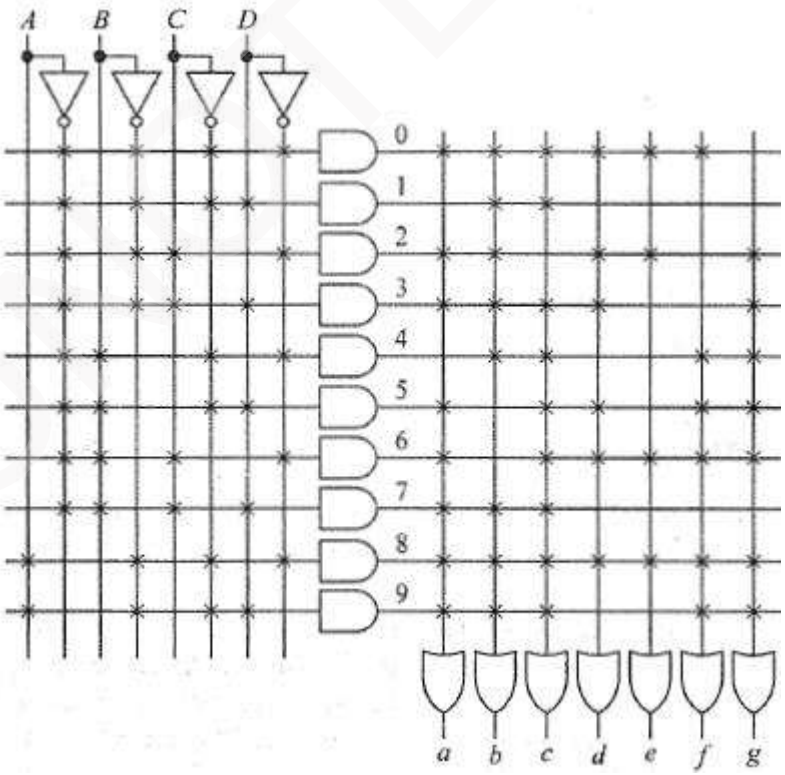


Figure 4.47: 7-segment decoder using PLA

Write a verilog code to realize a 2:1 multiplexer for the given circuit

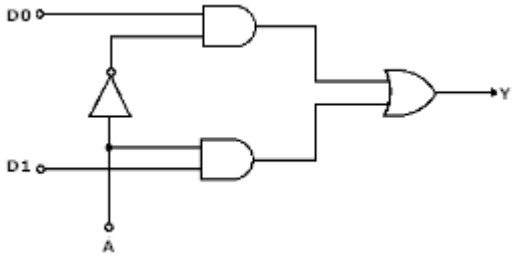


Figure 4.48:2 to 1 multiplexer

```
module mux2to1(A,D0,D1,Y);
input A,D0,D1
output Y;
assign Y=(~A&D0)|(A&D1);
endmodule
```

```
modue mux2to1(A,D0,D1,Y);
input A,D0,D1
output Y;
assign Y=A? D1: D0; //conditional assignment
endmodule
```

```
module mux2to1(A,D0,D1,Y);
input A,D0,D1
output Y;
reg Y;
always @ (A or D0 or D1)
if (A==1)
   Y=D1;
else
   Y=D0;
endmodule
```

```
module mux2to1(A,D0,D1,Y);
input A,D0,D1
output Y;
reg Y;
always @ (A or D0 or D1)
   case (A)
       0:Y=D0;
       1:Y=D1;
   endcase
endmodule
```

Write a verilog code to realize a 4:1 multiplexer using dataflow modeling

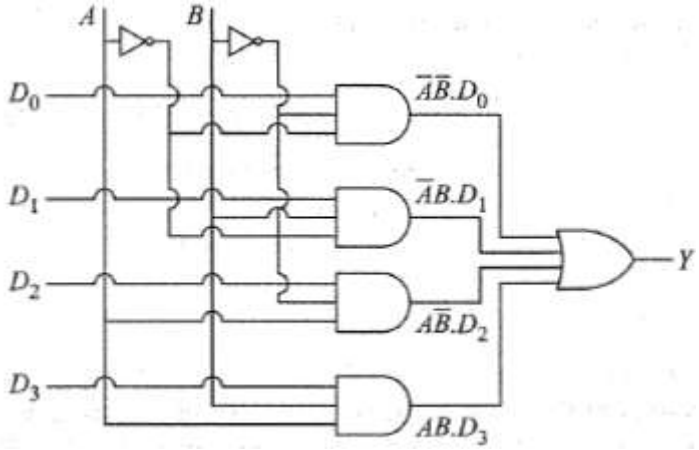


Figure: 4:1 multiplexer

```
module mux4to1(A,B,D0,D1,D2,D3,Y);
  input A,B,D0,D1,D2,D3;
  output Y; /* Circuit shown
  in Fig. 4.1(c)*/
  assign  Y = A ?( B ? D3 : D2):(B ? D1 : D0);
endmodule
```

Write a verilog code to realize a 4:1 multiplexer using behavioral modeling

```
module mux4to1(A,B,D0,D1,D2,D3,Y);
  input A,B,D0,D1,D2,D3;
  output Y;
  reg Y;
  always @ (A or B or D0 or D1 or D2
  or D3)
    case ({A,B}) /*Concatenation of
    A and B, A is MSB*/
     0: Y=D0; /*Two binary digit can
        generate*/
     1: Y=D1; /*four different values
        0,1,2,3 for*/
     2: Y=D2; /*binary combination
        00,01,10*/
     3: Y=D3; //and 11 respectively
  endcase
endmodule
```

EXAMPLE 4.11

Show how data processing circuits can be used to compare two 2-bit numbers, A_1A_0 and B_1B_0 to generate two outputs, A>B and A= B.

Solution We can use multiplexers, decoder or simply a 4-bit comparator. The truth table of the above problem is shown in Fig. 4.49.

In Method-1, we use two 16 to 1 multiplexers to realize $A > B$ and $A = B$ as shown in Fig. 4.50. The numbers A_1A_0 and B_1B_0 are used as selection inputs as shown. For every selection of input, the corresponding data input goes to the output. The input assignment comes straight from the truth table in Fig. 4.49 for the two cases.

A_1A_0	B_1B_0	A>B	A=B
00	00	0	1
00	01	0	0
00	10	0	0
00	11	0	0
01	00	1	0
01	01	0	1
01	10	0	0
01	11	0	0
10	00	1	0
10	01	1	0
10	10	0	1
10	11	0	0
11	00	1	0
11	01	1	0
11	10	1	0
11	11	0	1

Truth table

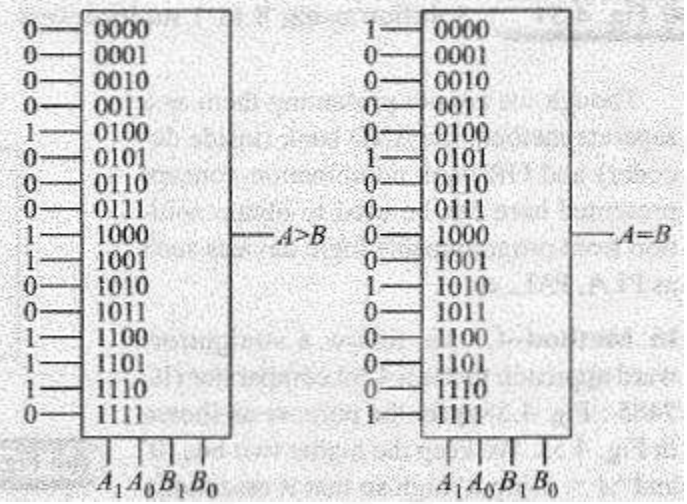


Figure 4.50:Solution using 16 to 1 multiplexers

In Method-2, we use two 8 to 1 multiplexers to realize $A > B$ and $A = B$ as shown in Fig. 4.51. The numbers A_1A_0 and B_1 are used as selection inputs while B_0 is part of the data input. We form pair of combinations of the truth table for constant $A_1A_0B_1$ and B_0 variable. This helps to find out how output varies with B_0.

In Method-3, we use one 4 to 16 decoder and two multi-input OR gates to realize $A > B$ and $A = B$ as shown in Fig. 4.52. We sum selected minterms, as required from the truth table, from the set of all the minterms generated by the decoder.

$A_1A_0B_1$ B_0	A>B	A=B
0 0 0 0	0 (0)	1 (B_0')
0 0 0 1	0 (0)	0 (0)
0 0 1 0	0 (0)	0 (0)
0 0 1 1	0 (0)	0 (0)
0 1 0 0	1 (B_0')	0 (B_0)
0 1 0 1	0 (0)	0 (0)
0 1 1 0	0 (0)	0 (0)
0 1 1 1	0 (0)	0 (0)
1 0 0 0	1 (1)	0 (0)
1 0 0 1	1 (1)	0 (0)
1 0 1 0	0 (0)	1 (B_0')
1 0 1 1	0 (0)	0 (0)
1 1 0 0	1 (1)	0 (0)
1 1 0 1	1 (1)	0 (0)
1 1 1 0	1 (B_0')	0 (B_0)
1 1 1 1	0 (0)	1 (B_0)

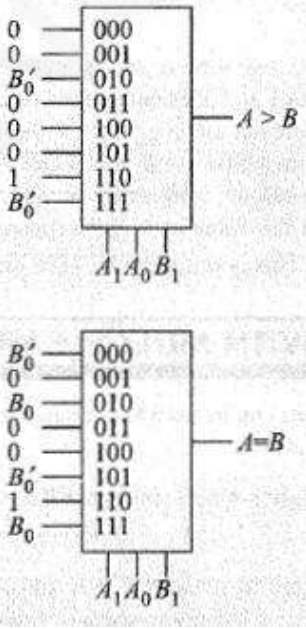


Figure 4.51:Solution using 8:1 multiplexers

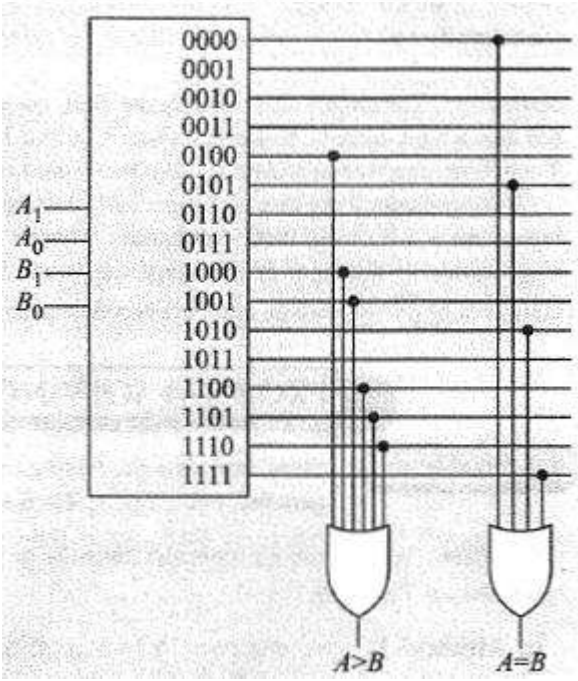


Figure 4.52:Solution using 4:16 decoder

Though we are not presenting them as a separate method, the AND bank (inside decoder) and OR bank combination concept presented here can be used to obtain solution from programmable logic devices such as PLA, PAL, etc.

In Method-4, we follow a straightforward approach to use a 4-bit comparator (IC 7485 : Fig. 4.38a) for the purpose as shown in Fig. 4.53. We keep the higher two bits '0' and '$A = B$ input' high so that it essentially becomes a 2-bit comparator generating all three outputs $A > B$, $A = B$ and $A < B$ of which only first two are useful here.

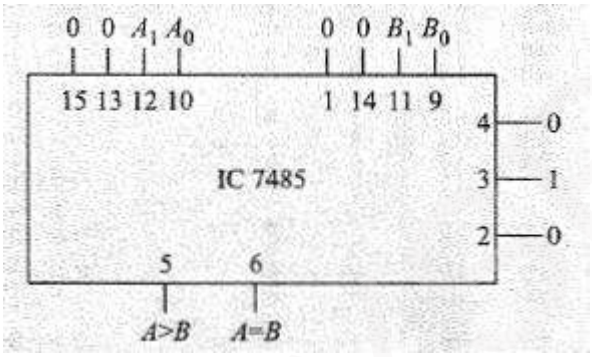


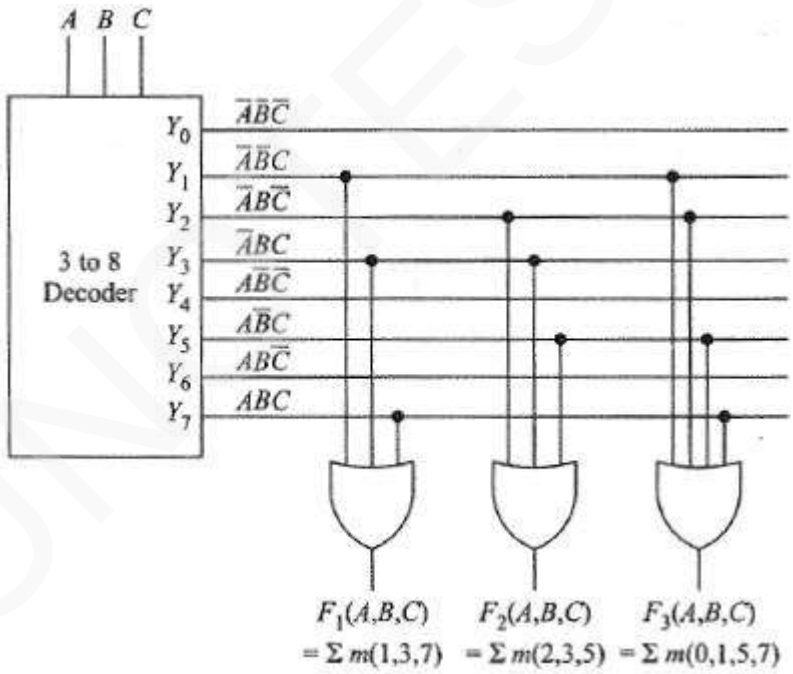
Figure 4.53: Solution using 4-bit comparator

EXAMPLE 4.12

Design a circuit that realizes following three functions using a decoder and three OR gates.

F1(A,B,C) = ∑m(l,3,7),
F2(A,B,C) = ∑m(2,3,5) and
F3(A,B,C) = ∑m(O, 1,5,7)

Solution:

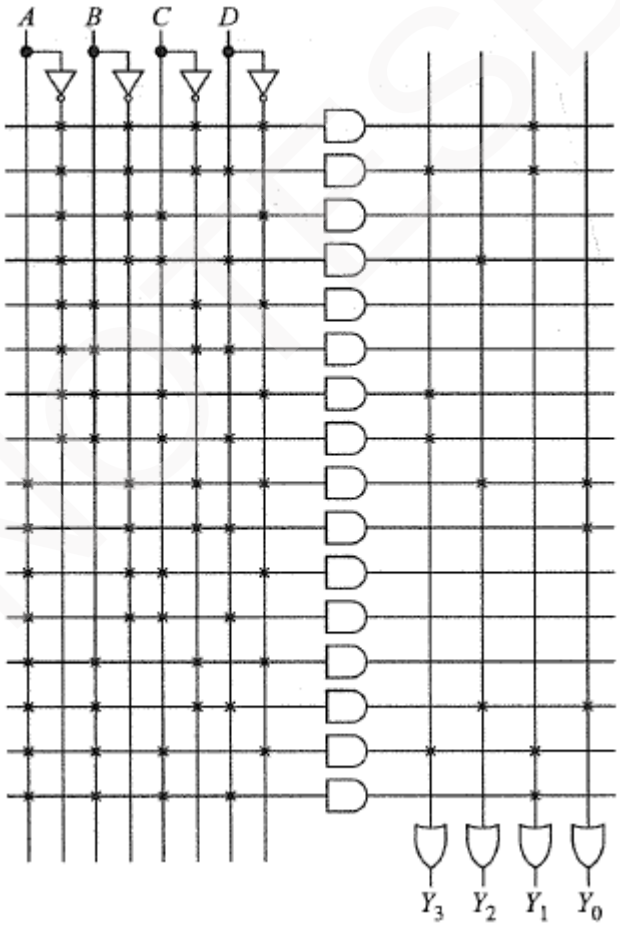


EXAMPLE 4.13

Draw a 4-input and 4-output PAL circuit that has the truth table of Table 4.11.

Solution:

A	B	C	D	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	1	0
0	0	0	1	1	0	1	0
0	0	1	0	0	0	0	0
0	0	1	1	0	1	0	0
0	1	0	0	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	0	1	0	0	0
0	1	1	1	1	0	0	0
1	0	0	0	0	1	0	1
1	0	0	1	0	0	0	1
1	0	1	0	0	0	0	0
1	0	1	1	0	0	0	0
1	1	0	0	0	0	0	0
1	1	0	1	0	1	0	1
1	1	1	0	1	0	1	0
1	1	1	1	0	0	1	0



EXAMPLE 4.14

Draw a 3-input and 3-output PLA circuit to generate the following boolean functions

f(a,b,c) = a'b' + abc

g(a,b,c) = a'b'c' + ab + bc

h(a,b,c) = c

Solution:

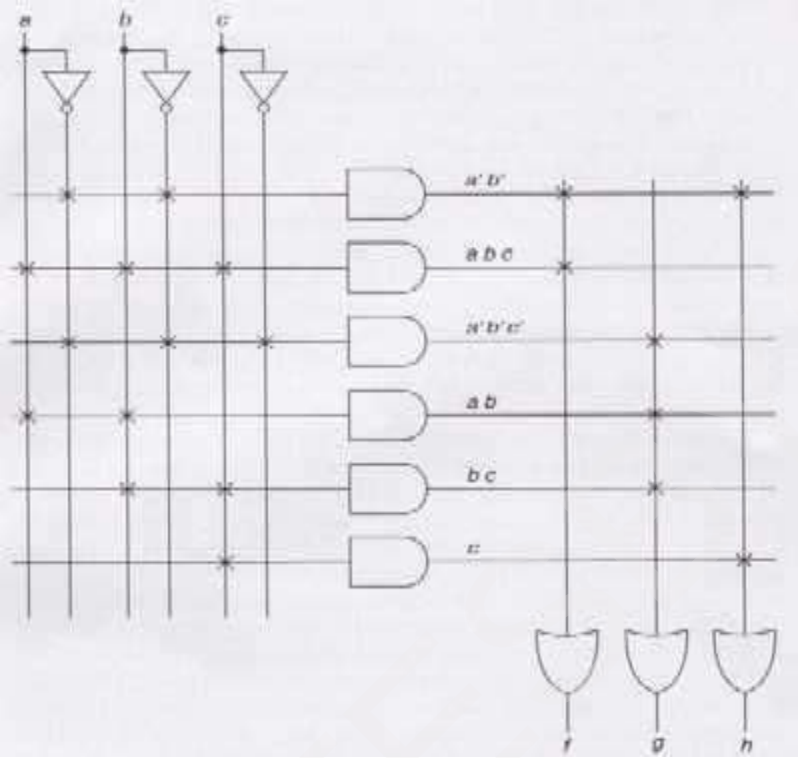


Figure 4.46: Example of programming a PLA

ARITHMETIC BUILDING BLOCKS

• These building blocks are the half-adder, the full-adder, and the controller inverter.

HALF-ADDER

• It is a logic circuit with two inputs and two outputs (Fig. 6.2).

• It adds 2 bits at a time, producing a sum and a carry output.

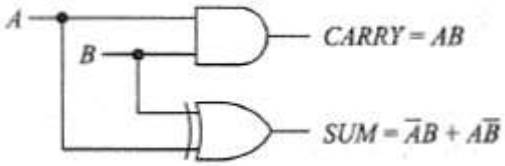


Figure 6.2: Half-adder

A	B	CARRY	SUM
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Table 6.2 Half-adder Truth Table

FULL-ADDER

• It is a logic circuit with three inputs and two outputs (Fig. 6.3).

• The circuit adds 3 bits at a time, giving a sum and a carry output

A	B	C	CARRY	SUM
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Table 6.3 Half-adder Truth Table

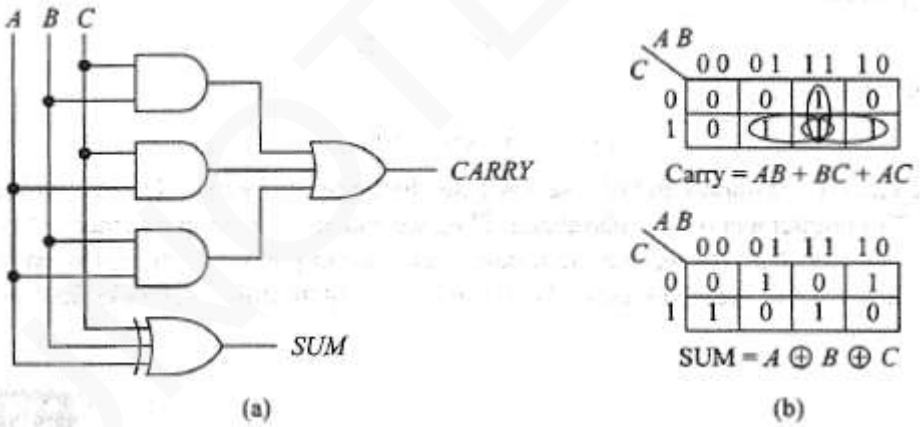


Figure 6.3: (a) Full-adder, (b) Karnaugh map of Table 6.3

CONTROLLED INVERTER

- When INVERT is low, it transmits the 8-bit input to the output (Figure 6.5).
  When INVERT is high, it transmits the 1's complement.

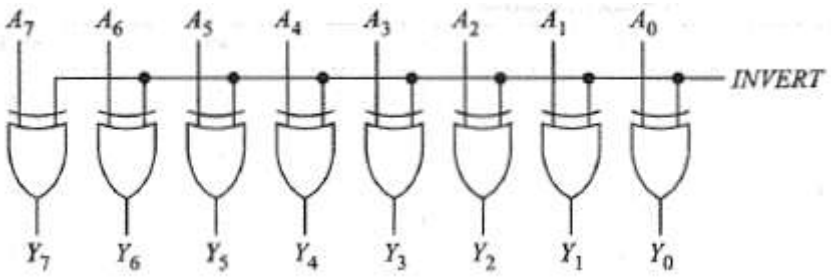


Figure 6.5: Controlled inverter

- For example, if the input number is

$$A_7 \cdots A_0 = 0110\ \ 1110$$

- A low INVERT produces

$$Y_7 \cdots Y_0 = 0110\ \ 1110$$

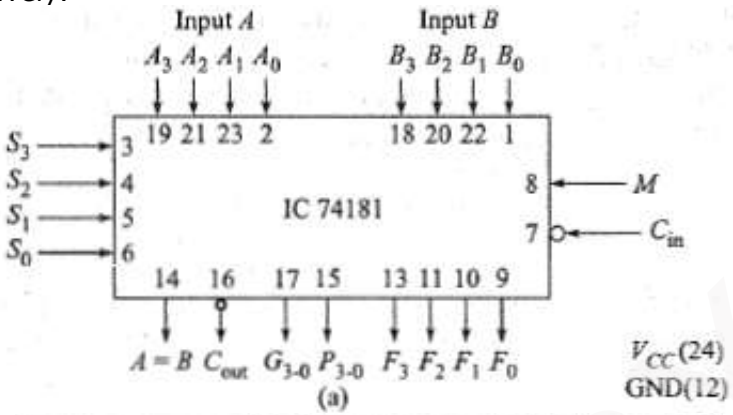
- A high INVERT produces

$$Y_7 \cdots Y_0 = 1001\ \ 0001$$

- The controlled inverter can be used for performing subtraction of 2's complement numbers.
- During a subtraction,
  - We first need to take the 2's complement of the subtrahend.
  - Then, we can add the complemented subtrahend to obtain the answer.
- With a controlled inverter, we can produce the 1's complement.

ARITHMETIC LOGIC UNIT (ALU)

- ALU is multifunctional device that can perform both arithmetic and logic function.
- ALU is an integral part of CPU of a computer.
- Other than normal addition & subtraction, it can also perform increment, decrement operations.
- As logic unit, it performs usual AND, OR, NOT, EX-OR and many other complex logic functions.
- It also has PRESET and CLEAR options which are used for setting (=1) & clearing (=0) all the function outputs respectively.



$S_3 S_2 S_1 S_0$	$M = 1$ (Logic Function)	$M = 0$ (Arithmetic Function) $C_{in} = 1$ (For $C_{in} = 0$, add 1 to F)
0 0 0 0	$F = A'$	$F = A$
0 0 0 1	$F = (A+B)'$	$F = A + B$
0 0 1 0	$F = A'B$	$F = A + B'$
0 0 1 1	$F = 0$	$F =$ minus 1
0 1 0 0	$F = (AB)'$	$F = A$ plus (AB')
0 1 0 1	$F = B'$	$F = (A + B)$ plus (AB')
0 1 1 0	$F = A \oplus B$	$F = A$ minus B minus 1
0 1 1 1	$F = AB'$	$F = AB'$ minus 1
1 0 0 0	$F = A' + B$	$F = A$ plus (AB)
1 0 0 1	$F = (A \oplus B)'$	$F = A$ plus B
1 0 1 0	$F = B$	$F = (A + B')$ plus (AB)
1 0 1 1	$F = AB$	$F = AB$ minus 1
1 1 0 0	$F = 1$	$F = A$ plus A
1 1 0 1	$F = A + B'$	$F = (A + B)$ plus A
1 1 1 0	$F = A + B$	$F = (A + B')$ plus A
1 1 1 1	$F = A$	$F = A$ minus 1

(b)

Figure 6.9: (a) Functional representation of ALU IC 74181, (b) Its truth table

IC 74181

- IC 74181 is a 4-bit ALU that can generate 16 different kinds of outputs (Fig. 6.9).
- Normally, a mode selector input (M) decides whether ALU performs
  1) Logic operation (M=1) or
  2) Arithmetic operation (M=0).
- In each mode, different functions are selected by four selection inputs S_3, S_2, S_1 and S_0.

Logic Operation	**Arithmetic Operation**
Logic operations are done bit-wise by → making M = 1 & → choosing appropriate select inputs.	Arithmetic operations are done by → making M = 0 → choosing appropriate select inputs & → choosing appropriate C_{in} (active low), if any.
For ex, ➢ Assume we want to perform AND operation between two 4-bit numbers: 1101 and 0111. ➢ Enter input A_3 .. A_0= 1101 and B_3 .. B_0 = 0111. ➢ Make S_3 .. S_0 = 1011. ➢ The output is shown as F_3 . .F_0 = 0011.	For ex, ➢ Assume we want to add decimal numbers 6 & 4. ➢ Enter input A_3 .. A_0= 0110 and B_3 .. B_0 = 0100. ➢ Make S_3 .. S_0 = 1001 and C_{in} = 1 (active low) ➢ The output is shown as F_3 . .F_0 = 1010 (which is decimal equivalent of 10)

Example 6.1

Show how a half-adder can be realized.

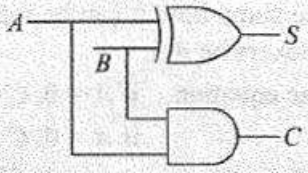
Solution The half-adder truth table and logic equations are reproduced from Section 6.7 in Fig. 6.10a.

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

(a)

$C = AB$

$S = A \oplus B$
$= AB' + A'B$

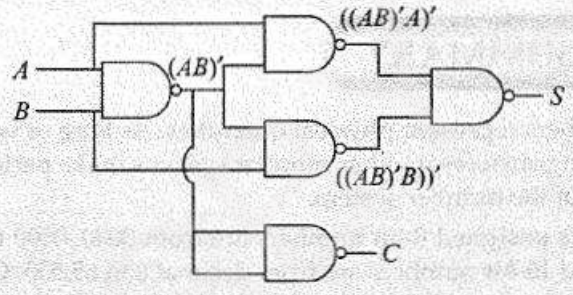


(b)

$C = A \cdot B = ((A \cdot B)')'$: double complement

$S = A \cdot B' + A'B$
$= (A \cdot B' + A'B)''$: double complement
$= ((A \cdot B')' \cdot (A'B)')'$: from Eq. 2.1
$= ((A' + B) \cdot (A + B')'$: from Eq. 2.2
$= (A'B' + AB)'$: since $X'X = 0$
$= (A'B')' \cdot (AB)'$: from Eq. 2.1
$= (A + B) \cdot (AB)'$: from Eq. 2.2
$= (AB)'A + (AB)'B$
$= ((AB)'A + (AB)'B)''$: double complement
$= (((AB)'A)' \cdot ((AB)'B))'$: from Eq. 2.1

(c)



(d)

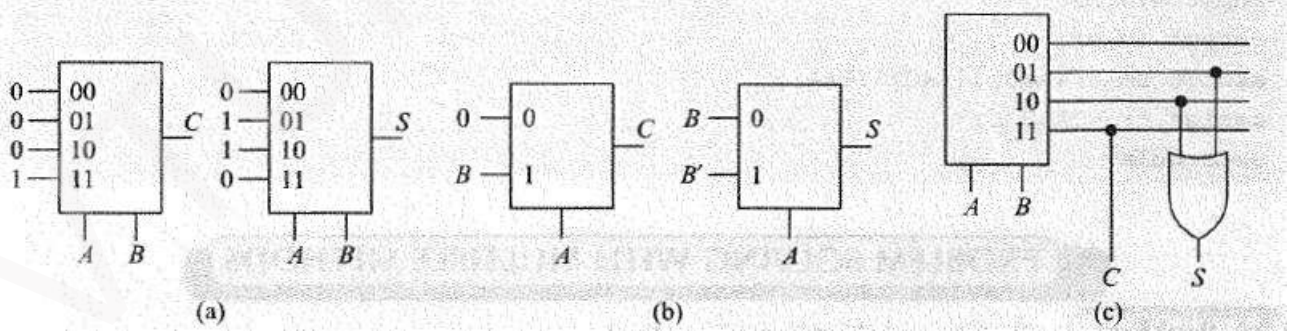
Figure 6.10: (a) Truth table and logic equation for half-adder, (b) Realization using AND and exclusive-OR gate, (c) Derivation of AND and exclusive-OR relation, (d) Realization using only NAND gates

In Method-1, the logic relations can directly be realized using AND and exclusive-OR gate as shown in Fig. 6.10b.

In Method-2, we show how it can be realized using only one type of basic gates, say NAND gate. The derivation is shown in Fig. 6.10c and the realization is shown in Fig. 6.10d.

It is left as an exercise to find how it can be realized using only NOR gates.

In Method-3, we show how it can be realized using two 4 to 1 multiplexers. We make use of the truth table to assign data inputs to the multiplexers while A and B are used as select inputs. The realization is shown in Fig. 6.11a.



(a) (b) (c)

Figure 6.11: Realization using (a) 4 to 1 multiplexers, (b) 2 to 1 multiplexers, (c) Decoder, OR gate

In Method-4, we show how it can be realized using two 2 to 1 multiplexers. Let the only selected input to the multiplexers be A.

We note from the equation, if $A = 0, C = 0$ and if $A = 1, C = B$

if $A = 0, C = B$ and if $A = 1, C = B'$

The realization from these is shown in Fig. 6.11b where B is used in the data input.

In Method-5, we show how it can be realized using a 2 to 4 decoder and OR gate. The decoder generates all the four minterms $A'B'$, $A'B$, AB' and AB. Carry output is generated directly from AB. Sum output is generated OR-ing $A'B$ and AB'. The realization is shown in Fig. 6.11c.

Example 6.2

Suppose that FD34H is the input to a 16-bit controlled inverter. What is the inverted output in hexadecimal notation? In binary?

Solution:

FD34H → 1111 1101 0011 0100

(FD34H)'→ 0000 0010 1100 1011 (in binary notation)

→ 02CBH (in hexadecimal notation)

Example 6.3

Show how 7 can be subtracted from 13 using IC 74181 (ALU).

Solution:

Substitute M = 0 and S_3 .. S_0 = 0110, C_{in}= 0, A_3 .. A_0= 1101 and B_3 .. B_0 = 0111.

MODULE 3 (CONT.): FLIP-FLOPS

INTRODUCTION

• The outputs of the digital circuits such as multiplexer, decoder & encoder are dependent entirely on their inputs i.e. If an input changes state, output may also change state.
• However, there are requirements for a digital circuit whose output will remain unchanged, once set, even if there is a change in input level(s).
Such a device can be used to store a binary number.
A **flip-flop** is one such circuit.
• Flip-flops are used in the construction of registers and counters.
• In a sequential logic circuit, flip-flops serve as key memory-elements.
• Analysis of sequential-circuits are done through
→ truth tables or
→ characteristic equations of flip-flops.
• The analysis-result is normally presented through
→ state table or
→ state transition diagram.

BISTABLE ELECTRONIC CIRCUIT

• Any device that has 2 stable states is said to be bistable.
• For example: 1) Toggle Switch and 2) Flip-Flop.

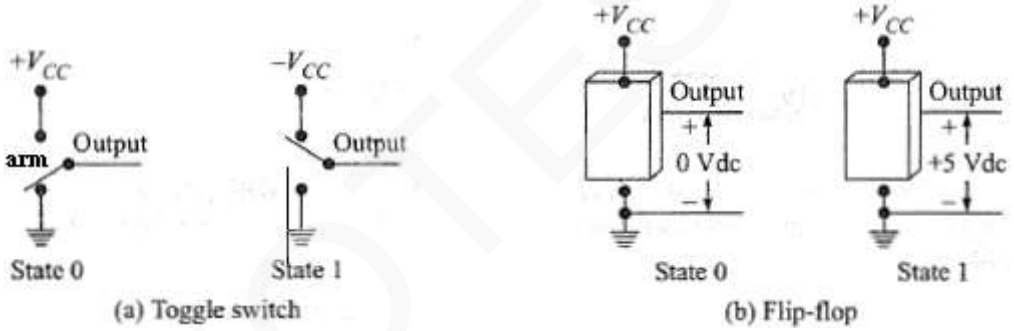


Figure 8.1: Bistable device

1) Toggle Switch

- A toggle switch has 2 stable states.
- As shown in Fig. 8.1a, the arm is either up or down, depending on the position of the switch.
- The switch is said to have memory since it will remain as set until someone changes its position.
- In general, the switch can be regarded as a memory device.

2) Flip-Flop

- A flip-flop is a bistable circuit that has 2 stable states i.e. 0=0V or 1=5V.
- As shown in Fig. 8.1b, output of flip-flop is either 0 or +5V dc.
- The flip-flop also has memory since its output will remain as set until something is done to change it.
- In general, the flip-flop can be regarded as a memory device.
- In fact, the flip-flop can be used to store one binary digit (bit).
- For example,
  1) When the flip-flop has its output set at 0V dc, it can be regarded as storing a logic 0.
  2) When the flip-flop has its output set at +5V dc, it can be regarded as storing a logic 1.
- The flip-flop is often called a **latch**, since it will hold (or latch) in one of the 2 stable states.

BASIC IDEA OF FLIP-FLOPS

• As shown in Fig. 8.2a, a flip-flop can be constructed by connecting 2 inverters in series.

• The line connecting the output of inverter-B (INV-B) back to the input of inverter-A (INV-A) is referred to as the *feedback-line*.

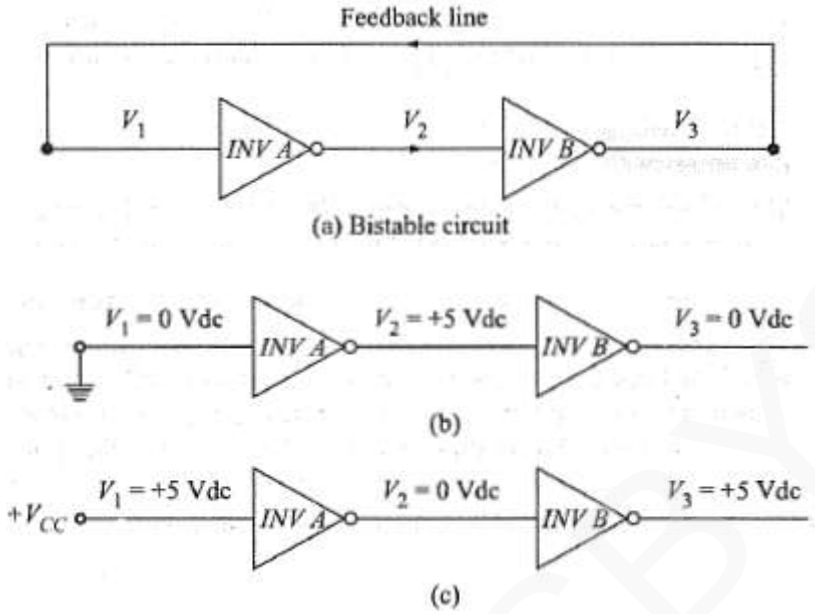


Figure 8.2: Bistable circuit

• Consider V_1 as the input and V_3 as the output.

• There are 2 signals in a digital system: 1) L = 0 = 0V dc and 2) H = 1 = +5V dc.

L=0=0V dc	H=1=+5V dc
➢ As shown in Fig. 8.2b, if V_1 is 0V dc, then V_3 will also be 0V dc. ➢ When the input of INV-A is grounded, the output of INV-B will go low. ➢ This is 1st stable state: V_3 = 0V dc.	➢ As shown in Fig. 8.2c, if V_1 is +5V dc, then V_3 will also be +5V dc. ➢ When the input of INV-A is +5V dc, the output of INV-B will go high. ➢ This is 2nd stable state: V_3 = +5V dc.

CONSTRUCTING RS FLIP-FLOP

• Two different methods for constructing an RS flip-flop: 1) NOR-Gate latch 2) NAND-Gate latch.

NOR-GATE LATCH

• As shown in Fig 8.3, two 2-input NOR gates are connected to form a flip-flop.

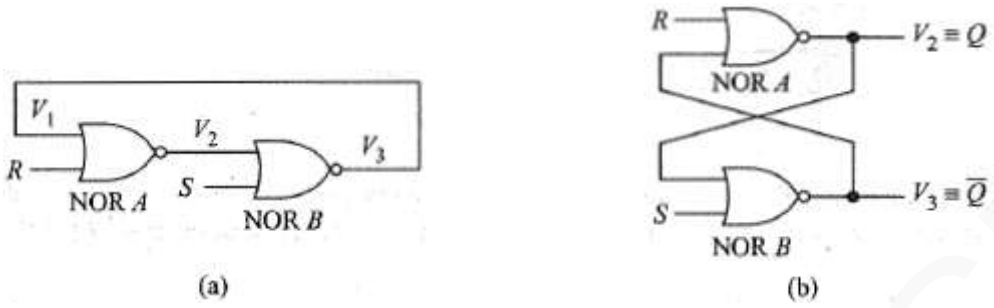
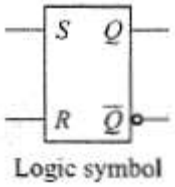


Figure 8.3: NOR-gate flip-flop

• As shown in Fig 8.6a, the flip-flop has
→ two inputs: R and S.
→ two outputs: Q and Q'. Regardless of the value of Q, its complement is Q'.



R	S	Q	Action
0	0	Last state	No change
0	1	1	SET
1	0	0	RESET
1	1	?	Forbidden

(b) Truth table

Figure 8.6: RS flip-flop

• Here's how it works (Fig 8.6b):

1) R=0 and S=0

➢ Since a 0 at the input of a NOR gate has no effect on its output, the flip-flop simply remains in its present state i.e. Q remains unchanged.

2) R=0 and S=1.

➢ This condition forces the output of NOR gate-B low.
➢ Since both inputs to NOR gate-A are now low, the output must be high.
➢ Thus, a 1 at the S input is said to SET the flip-flop.
➢ Finally, the flip-flop switches to the stable state where Q = 1.

3) R=1 and S=0.

➢ This condition forces the output of NOR gate-A low.
➢ Since both inputs to NOR gate-B are now low, the output must be high.
➢ Thus, a 1 at the R input is said to RESET the flip-flop.
➢ Finally, the flip-flop switches to the stable state where Q = 0 (or Q' = 1).

4) R=1 and S=1.

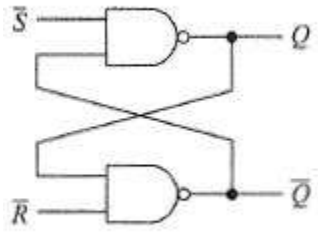
➢ This condition is forbidden, as it forces the outputs of both NOR gates to the low state.

Q: Why R=1, S=1 is forbidden?

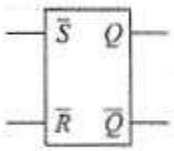
➢ Incidentally, if this condition is for some reason, imposed and the next input is R = 0, S = 0 then the resulting state Q depends on propagation delays of two NOR gates.
➢ If delay of gate-A is less, i.e. it acts faster;
Then Q = 1;
Otherwise Q =0.
➢ Such dependence makes the job of a design engineer difficult, as any replacement of a NOR gate will make Q unpredictable.

NAND-GATE LATCH

- This NAND-gate latch is also called as **R'S' flip-flop**.
- Recall: A low on any input to a NAND gate will force its output high.
- Here's how it works (Fig 8.7c):
    1) If both R' and S' are high, the flip-flop will remain in its previous state.
    2) A low on the S' input will set the latch (Q = 1 and Q' = 0).
    3) A low on the R' input will reset the latch (Q = 0).
    4) Setting both R' and S' low simultaneously is forbidden since this forces both Q and Q' high.



(a) NAND gate latch



Logic symbol

(b)

$\overline{R}$	$\overline{S}$	Q
1	1	Last state
1	0	1
0	1	0
0	0	? (Forbidden)

(c) Truth table

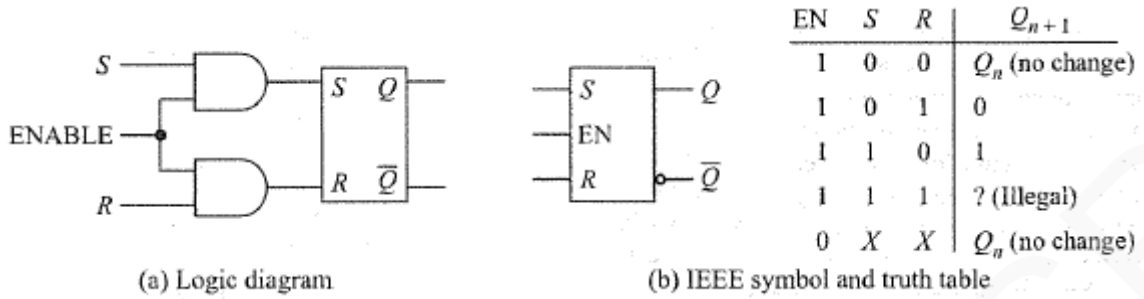
Figure 8.7: R'S' flip-flop

GATED (OR CLOCKED) FLIP-FLOPS

• Here, we discuss about 1) Clocked RS flip-flop 2) Clocked D Flip-flop.

CLOCKED RS FLIP-FLOPS

• As shown in Fig 8.11a, a flip-flop can be enabled or disabled by the adding 2 AND gates at the R and S inputs.



EN	S	R	Q_{n+1}
1	0	0	Q_n (no change)
1	0	1	0
1	1	0	1
1	1	1	? (Illegal)
0	X	X	Q_n (no change)

(a) Logic diagram (b) IEEE symbol and truth table

Figure 8.11: Clocked RS flip-flop

• Here's how it works:

- When **ENABLE=low**, the AND gate outputs must be low.
  And changes in neither R nor S will have any effect on the flip-flop output Q.
  The latch is said to be **disabled**.
- When **ENABLE=high**, information at R and S inputs will be transmitted directly to the outputs.
  The latch is said to be **enabled**.
- The output will change, in response to input changes, as long as the ENABLE is high.
- When the ENABLE input goes low, the output will retain the old information that was present on the input when the high-to-low transition took place.
- In this way, it is possible to
  → strobe the flip-flop in order to store information (set it or reset it) at any time and
  → then hold the stored information for any desired period of time.

• This type of flip-flop is called a **gated or clocked RS flip-flop**.

• As shown in Fig 8.12, there are 3 inputs: R, S, and EN (ENABLE).

• The truth-table output is not simply Q, but Q_{n+1}.This is because, we must consider 2 different instants in time:

1) The time before the ENABLE goes low Q_n.
2) The time just after ENABLE goes low Q_{n+1}.

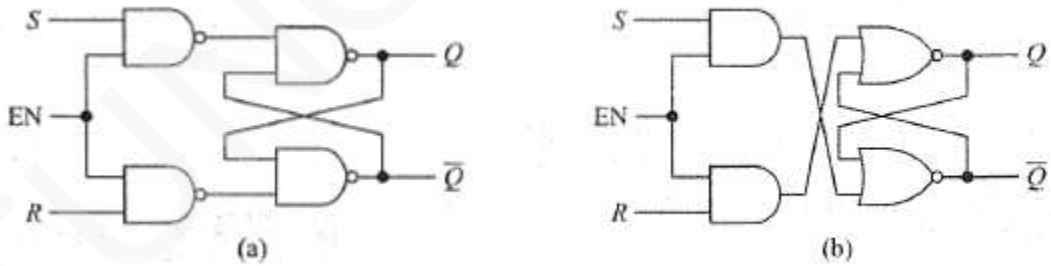


Figure 8.12: Two different realizations for a clocked RS flip-flop

• The simple latch-type flip-flops are completely transparent; that is, the output Q immediately follows any change of state at the input (R or S).

• **Drawbacks**:

1) The RS flip-flop has two data inputs, R and S. Generation of 2 signals to drive a flip-flop is a disadvantage in many applications.
2) Furthermore, the forbidden condition of both R and S high may occur inadvertently.

Solution: Use D flip-flop which needs only a single data input.

CLOCKED D FLIP-FLOPS

• A D flip-flop is a bi-stable circuit whose D input is transferred to the output when EN is high.
• The flip-flop is
→ disabled when EN is low.
→ transparent when EN is high.

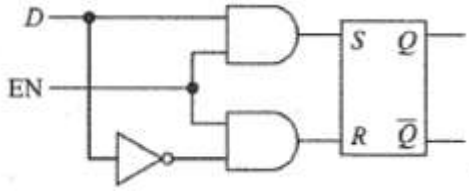
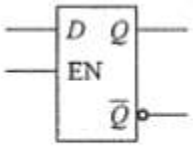


Figure 8.14: A D Flip-flop

• Here is how it works (Fig 8.14):
1) When **EN=low**, both AND gates are disabled. Therefore, Q is independent of value of D.
2) When **EN=high**, both AND gates are enabled. Therefore, Q is forced to equal value of D.
3) When EN again goes low, Q retains or stores the last value of D.
• This kind of D flip-flop is often called a **D latch**.



(a) Logic symbol

EN	D	Q_{n+1}
0	X	Q_n (last state)
1	0	0
1	1	1

(b) Truth table

Figure 8.15: D Flip-flop logic symbol & truth table

• As shown in truth table (Fig 8.15b):
1) When **EN= low**, D is a don't care (X); Q will remain latched in its last state.
2) When **EN= high**, Q takes on the last value of D.

Idea of Data Storage

- As shown in Fig 8.16, four D latches are driven by the same clock signal.
- When the clock goes high, input data is loaded into the flip-flops and appears at the output. When the clock goes low, the output retains the data.
- For example,

Suppose that the data input is

$$D_3D_2D_1D_0 = 0111$$

When clock goes high, this word is loaded into the D latches, resulting in an output of

$$Q_3Q_2Q_1Q_0 = 0111$$

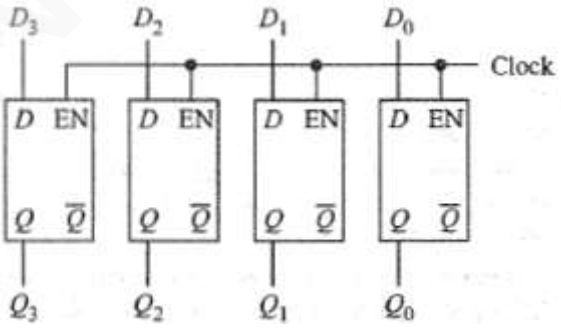


Figure 8.16: Storing a 4-bit word

• The gated or clocked D flip-flop is considered semitransparent. That is, the output Q will change state immediately provided that the EN input is high.
• **Drawbacks of Clocked RS/D Flip-Flops:**
- If any of these flip-flops are used in a synchronous system, care must be taken to ensure that all flip-flop inputs change state in synchronism with the clock.

Solution: The edge-triggered flip-flop overcomes this problem.

EDGE-TRIGGERED FLIP-FLOPS

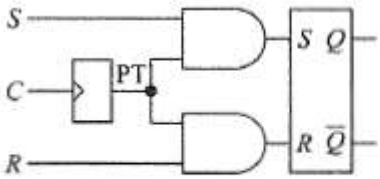
• Here, we discuss about
1) Edge-Triggered RS Flip-flop (Positive & Negative).
2) Edge-Triggered D Flip-flop.
3) Edge-Triggered JK Flip-Flop.

POSITIVE-EDGE-TRIGGERED RS FLIP-FLOPS

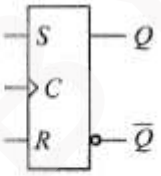
• As shown in Fig. 8.18a:
- The clock (C) is applied to a positive pulse-forming circuit.
- The PTs developed are then applied to a gated RS flip-flop.

• Each PT of the clock produces a very narrow PT that is applied to the AND gates.
• The AND gates are active only when the PT is high (perhaps 25 ns). Thus, Q can change state only during this short time period (Fig. 8.18d).
• In this manner, Q changes state in synchronism with the PTs of the clock.
• Advantages:
1) This flip-flop is easy to use in any synchronous system.
2) This flip-flop is transparent only during PTs; it is not transparent for the remainder of time.

(The small triangle inside the symbol (dynamic input indicator) indicates that Q can change state only with PTs of the clock C).



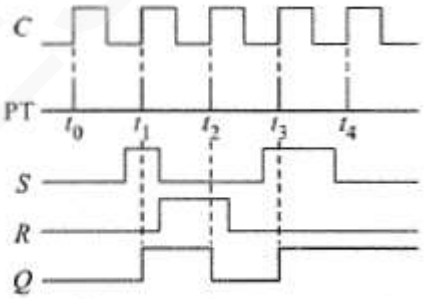
(a) Logic diagram



(b) IEEE symbol

C	S	R	Q_{n+1}	Action
↓	0	0	Q_n	No change
↓	0	1	0	RESET
↓	1	0	1	SET
↓	1	1	?	Illegal

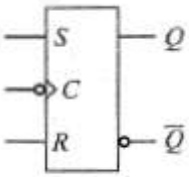
(c) Truth table



(d) Positive-edge-triggered *RS* flip-flop

Figure 8.18: Positive-edge-triggered RS flip-flop

NEGATIVE-EDGE-TRIGGERED RS FLIP-FLOPS



(a) IEEE symbol

C	S	R	Q_{n+1}	Action
↓	0	0	Q_n	No change
↓	0	1	0	RESET
↓	1	0	1	SET
↓	1	1	?	Illegal

(b) Truth table

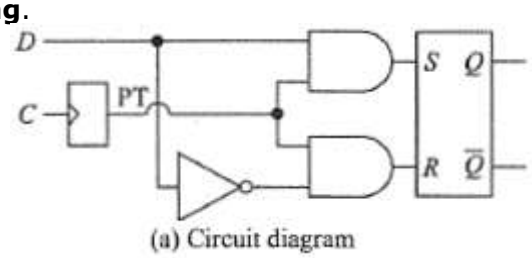
Figure 8.19: Negative-edge-triggered RS flip-flop

• As shown in the truth table (Fig. 8.19b), Q changes state according to the R and S inputs, but only during NTs of the clock.
• This flip-flop behaves exactly like the positive-edge-triggered RS flip-flop, except that changes in output Q are synchronized with NTs of the clock (C).

(On the IEEE symbol, the small bubble on the clock input C means active-low. This bubble, along with the dynamic input indicator, means negative-edge triggering).

EDGE-TRIGGERED D FLIP-FLOPS

- This flip-flop samples the data-bit at a unique point in time.
- As shown in Fig. 8.21a, the narrow positive pulse (PT) enables the AND gates for an instant.
- The effect is to activate the AND gates during the PT of C. This is equivalent to sampling the value of D for an instant.
- At this unique point in time, D & its complement hit the flip-flop inputs. This forces Q to set or reset.
- Again, this operation is called **edge triggering** because the flip-flop responds only when the clock is in transition between its 2 voltage states.
- The triggering occurs on the positive-going edge of the clock. That's why it is referred to as **positive-edge triggering**.



(a) Circuit diagram

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

(b) Truth table

Figure 8.21: Positive-edge-triggered D flip-flop

- As shown in truth table (Fig. 8.21b).
  - When the **clock=low**, D is a don't care(X) and Q is latched in its last state.
  - When the **clock=high**,
    - → data-bit is loaded into the flip-flop and
    - → Q takes on value of D. (PT designated by the up arrow)

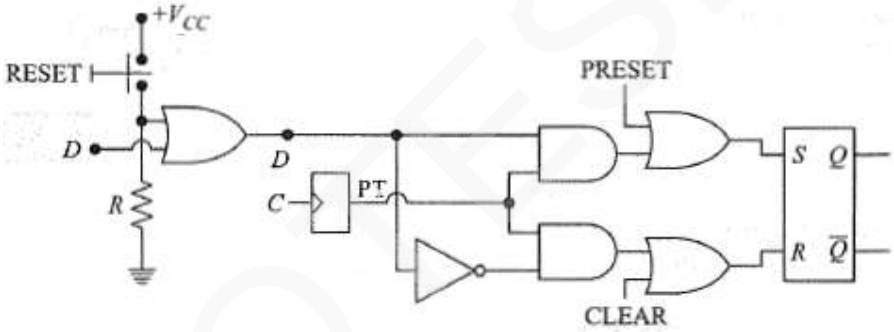


Figure 8.22: PRESET and CLEAR functions

- When power is first applied, flip-flops will be in random states.
- As shown in Fig. 8.22, the operator has to press a RESET button to start the computer.
- This sends a CLEAR or RESET signal to all flip-flops.
- Depressing the RESET button will set Q to 1.
- Q will remain high as long as the button is held closed.
- Furthermore, the OR gates allow us to slip in a high PRESET or a high CLEAR when desired.
- In some digital systems, it's necessary to preset (synonymous with set) certain flip-flops.
- A high PRESET forces Q to equal 1;
  A high CLEAR resets Q to 0.
- The PRESET and CLEAR are called **asynchronous inputs** because
  they activate the flip-flop independently of the clock.
- On the other hand, the D input is a **synchronous input** because
  it has an effect only with PTs of the clock.

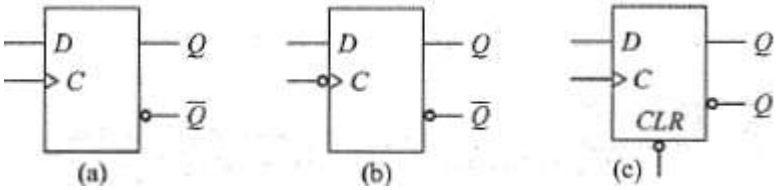


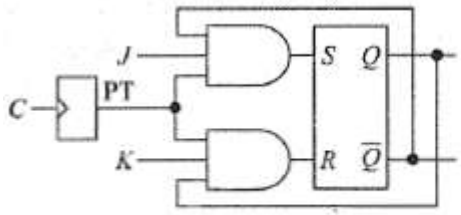
Figure 8.23: D flip-flop symbols: (a) Positive-edge-triggered, (b) Negative-edge-triggered, (c) Positive-edge-triggered with active low clear

EDGE TRIGGERED JK FLIP-FL.OPS

• **Drawbacks of edge-triggered RS Flip-Flop:**
- Setting R=S=1 with an edge-triggered RS flip-flop forces both Q & Q' to the same logic level.
- This is an illegal condition, and it is not possible to predict the final state of Q.

Solution: The JK flip-flop overcomes this problem.

POSITIVE-EDGE-TRIGGERED J K FLIP-FLOPS



(a) One way to implement a *JK* flip-flop

C	J	K	Q_{n+1}	Action
↑	0	0	Q_n (last state)	No change
↑	0	1	0	RESET
↑	1	0	1	SET
↑	1	1	$\overline{Q}_n$ (toggle)	Toggle

(b) Truth table

Figure 8.24: A positive-edge-triggered JK flip-flop

• As shown in Fig 8.24a, this circuit will be sensitive to PTs of the clock.
• The basic circuit is identical to the positive-edge-triggered RS flip-flop, with 2 important additions:
1. The Q output is connected back to the input of the lower AND gate.
2. The Q' output is connected back to the input of the upper AND gate.

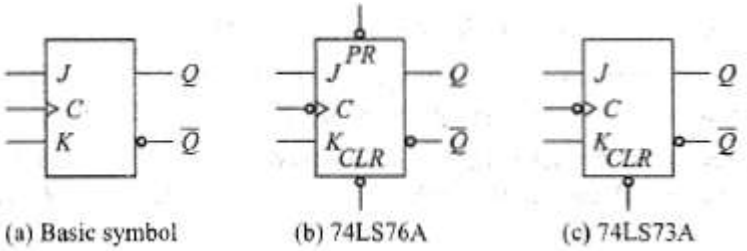
• Here's how it works (Fig 8.24b):

1) J=0 and K=0
- This means Q retains its last value.
- For J = 0 and K = 0, both AND gates are disabled. .'. clock pulses have no effect.

2) J=0 and K=1
- This means that the next PT of the clock resets the flip-flop.
- For J = 0 and K = 1, the upper gate is disabled.
- So, there's no way to set the flip-flop. But you can reset the flip flop as follows:
  i) When Q is high, lower gate passes a RESET pulse on the next positive clock edge PT.
  ii) This forces Q to become low (2nd entry in the truth table).

3) J=1 and K=0
- This means that the next PT of the clock sets the flip-flop.
- For J = 1 and K = 0, the lower gate is disabled.
- So, there's no way to reset the flip-flop. But you can set the flip flop as follows:
  i) When Q' is high; the upper gate passes a SET pulse on the next PT.
  ii) This forces Q to become high (3rd entry in the truth table).

4) J=1 and K=1
- This mean the flip-flop will toggle (switch to the opposite state) on the next positive clock edge.
- For J = 1 and K = 1, either lower gate or upper gate can be disabled.
- So, it's possible to set or reset the flip-flop as follows:
  i) When Q is high, the lower gate passes a RESET pulse on the next PT.
  ii) When Q is low, the upper gate passes a SET pulse on the next PT.
  iii) Either way, Q changes to the complement of the last state (4th entry in truth table).



(a) Basic symbol (b) 74LS76A (c) 74LS73A

Figure 8.25: JK flip-flop symbols

MODULE 4: FLIP-FLOPS (CONT.)

FLIP-FLOP TIMING

• **Switching time t_p** represents the amount of time it takes for the output of a gate or flip-flop to change states after the input changes.

- For example:
  D flip-flop has a switching time t_p=10 ns → it takes about 10 ns for Q to change states.
- Switching time is the main cause of propagation delay.

• **Setup time t_{setup}** is minimum amount of time that data-bit must be present before clock edge hits.

- For example:
  D flip-flop has a setup time of 15 ns → the data-bit must be at the D input at least 15 ns before the clock edge arrives.
- Stray capacitance at the D input makes it necessary for data-bit D to be at the input before the clock edge arrives.

• **Hold time t_{hold}** is minimum amount of time that data-bit D must be present after the PT of the clock.

- For example:
  For t_{setup} = 15 ns and t_{hold} = 5 ns → the data-bit has to be
  → at the D input at least 15 ns before the clock edge arrives and
  → held at least 5 ns after the clock PT.

EDGE TRIGGERING THROUGH INPUT LOCK OUT

- As shown in Fig. 8.27, this circuit requires 3 NAND latches. (Total 6 NAND gates).
  1) One NAND gate (number-3) has 3 inputs
  2) The remaining five NAND gates (number-1,2,4,5,6) have 2 inputs.
- The output latch behaves like an SR flip flop where no change in output occurs if S =1, R=1.
- Here is how it works:
  1) If **Clock=0**, then the NAND logic makes both S = 1, R = 1 and thus there could be no change in the output.
  2) If **Clock=1**, then SR can always change if other inputs of NAND gates 2 and 3 change and thus the output is essentially level triggered.

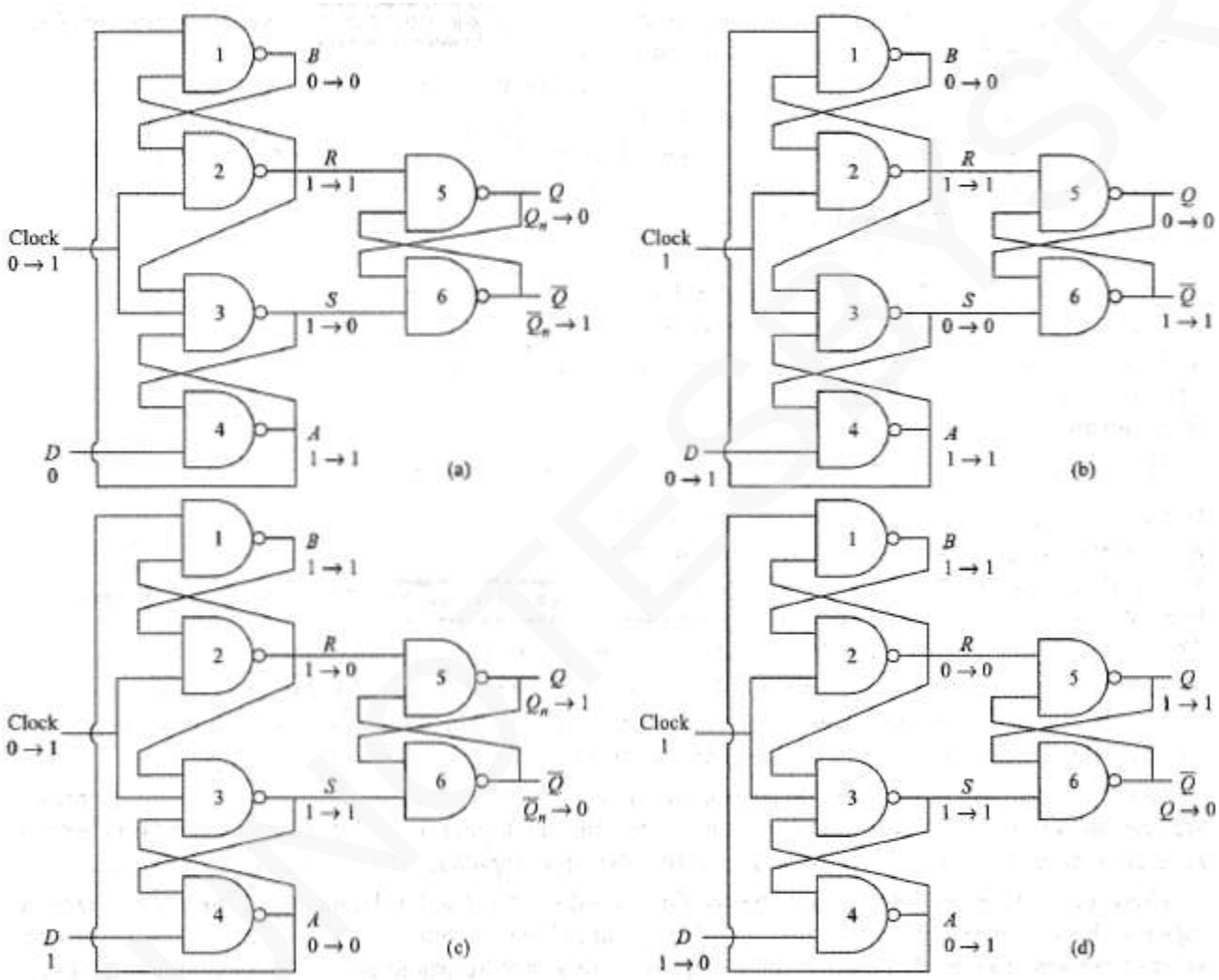


Figure 8.27: Positive edge triggering of D type flip-flop through input lock out

- Consider the case when Clock=0 and D=0 (Fig. 8.27a).
  - For a NAND gate, since 0 is the forcing input, the intermediate outputs are S=1, R=1 and A=1 which make B=0.
  - Now, clock makes a transition from 0-->1.
  - D=0 forces A=1 and B=0 keeps R=1.
  - Thus, after this transition, S=0, R=1, A=1 and B=0.
  - This makes Q=0 irrespective of the previous state.
  - Finally, the value at D, i.e. 0 is transferred to Q after the clock trigger.

JK MASTER-SLAVE FLIP-FLOPS

- Basically, it performs following 2 tasks:
  1) The master is set according to J and K while the clock is high (Fig 8.28).
  2) The contents of the master are then shifted into the slave (Q changes state) when the clock goes low.
- Here, 1) Master is positive-level-triggered.
  2) Slave is negative-level-triggered.
- Therefore, the master responds to its J and K inputs before the slave.

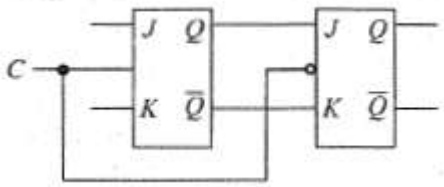
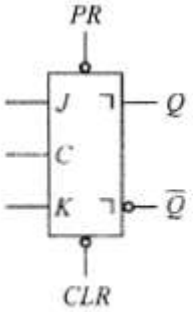


Figure 8.28: Master-slave flip-flop



(a) Symbol

C	J	K	Q_{n+1}	Action
⎍	L	L	Q_n	No change
⎍	L	H	L	RESET
⎍	H	L	H	SET
⎍	H	H	$\overline{Q}_n$	Toggle

(b) Truth table

Figure 8.29: 7476 JK master flip-flop

- Here's how it works (Fig 8.29):

1) J=1 and K=0

- The master sets on the positive clock-transition (PCT).
- The high Q output of the master goes to the J input of the slave.
- So, the slave sets on the negative clock-transition (NCT).
- Thus, the slave has copied the action of the master.

2) J=0 and K=1

- The master resets on the PCT.
- The high Q' output of the master goes to the K input of the slave.
- So, the slave resets on the NCT.
- Thus, the slave has copied the action of the master.

3) J=1 and K=1,

- Here, i) Master toggles on the PCT.
  ii) Slave then toggles on the NCT.
- Again, the slave copies the action of the master.
  i) If the master sets, the slave sets.
  ii) If the master resets, the slave resets.

4) J=0 and K=0

- The flip-flop is disabled and Q remains unchanged.

- The clock (C) is not edge-triggered.
- In fact, the master does change state when C goes high.
- When the **clock=high**, any change in J or K will immediately affect the master flip-flop.
- This particular flip-flop might be referred to as **pulse-triggered** (to distinguish it from the edge-triggered flip-flops).

SWITCH CONTACT BOUNCE CIRCUITS

- As shown in Fig 8.31a:
  1) When the switch is open, the voltage at point A is +5V dc.
  2) When the switch is closed, the voltage at point A is 0V dc.

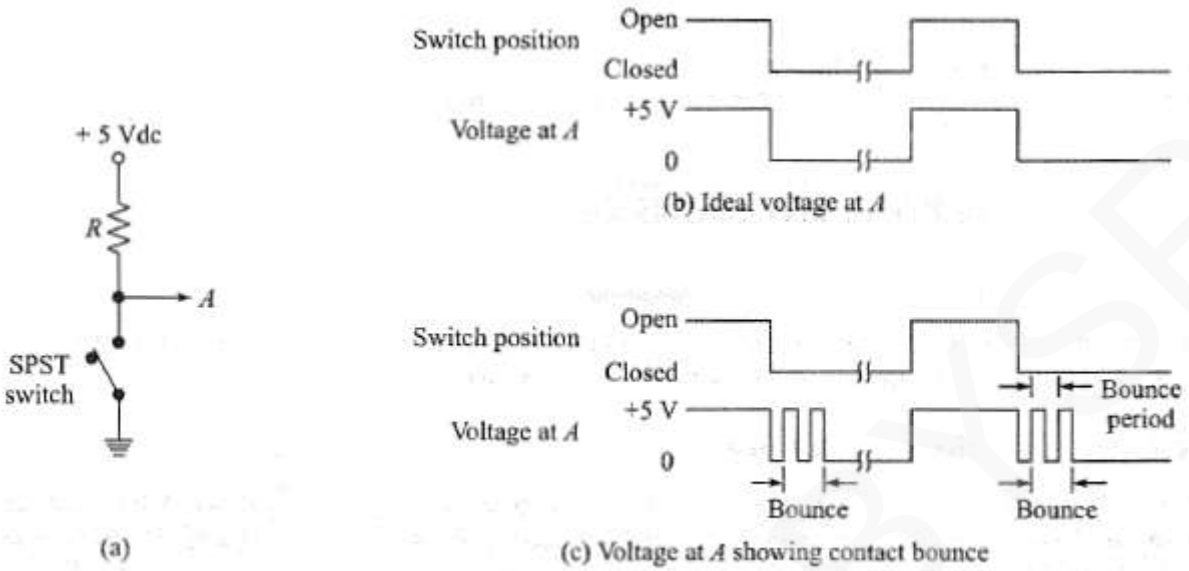


Figure 8.31: Switch Contact Bounce Circuit

- **Contact Bounce Problem:**
  - Any mechanical switching device consists of a moving contact.
  - The switching device is restrained by some sort of a spring system.
  - As a result, when the arm is moved from one stable position to the other, the arm bounces.
  - This phenomenon is known as **contact bounce problem** (Fig 8.31c).

Solution: RS latch debounce circuit overcomes this problem.

RS LATCH DEBOUNCE CIRCUIT

• The RS latch will remove any contact bounce due to the switch.

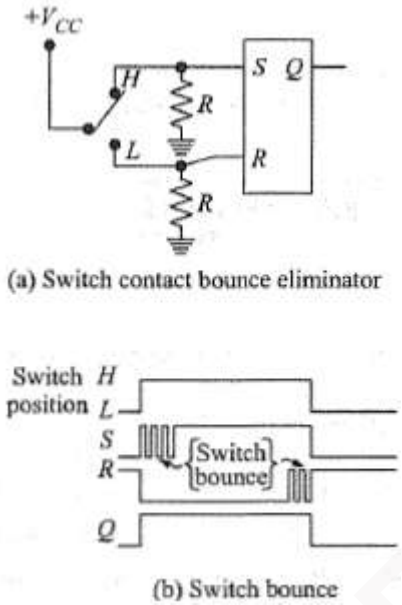


Figure 8.32: Debounce circuit

• As shown in Fig 8.32a, the output (Q) is used to generate the desired switch signal.
• When the switch is moved to position H, R = 0 and S = 1.
• Bouncing occurs at the S input due to the switch.
• The flip-flop "sees" this as a series of high and low inputs, settling with a high level.
• The flip-flop will immediately be set with Q = 1 at the first high level on S.
• When the switch bounces, losing contact, the input signals are R = S = 0.
Therefore, the flip-flop remains set (Q = 1).
• When the switch regains contact, the input signals are R = 0 and S = 1.
This causes an attempt to again set the flip-flop.
But since the flip-flop is already set, no changes occur at Q (Fig 8.32b).
• Thus, the flip-flop responds only to the first, high level at its S input. This results in a "clean" low-to-high signal at its output (Q).

VARIOUS REPRESENTATIONS OF FLIP-FLOPS
CHARACTERISTIC EQUATIONS OF FLIP-FLOPS

- The characteristic equations of flip-flops are useful in analyzing circuits made of them.
- Next output Q_{n+1} is expressed as a function of
  1) Present output Q_n of flip-flops &
  2) Input to flip-flops.
- Truth table of each flip-flop is mapped into K-map.
- K-Map is used to get the optimized expression (Fig 8.33).

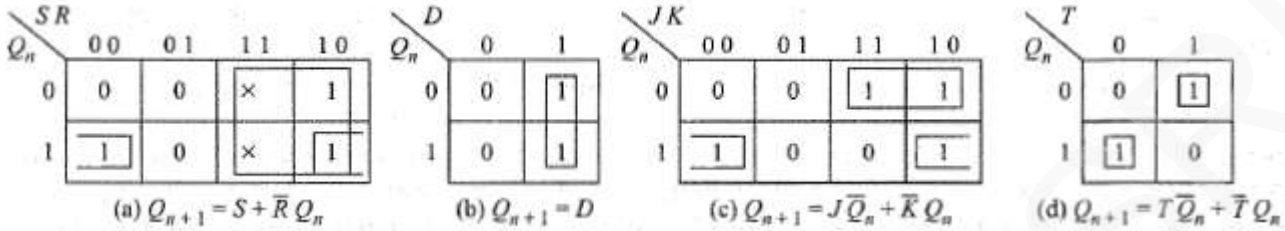


Figure 8.33: Characteristic equations of (a) SR flip-flop, (b) D flip-flop, (c) JK flip-flop,(d) T flip-flop

- The equation for 4 different flip-flops can be summarized as

SR flip-flop:	$Q_{n+1} = S + R'Q_n$
JK flip-flop:	$Q_{n+1} = JQ_n' + K'Q_n$
D flip-flop:	$Q_{n+1} = D$
T flip-flop:	$Q_{n+1} = TQ_n' + T'Q_n$

FLIP-FLOPS AS FINITE STATE MACHINE

- In a sequential logic circuit, the value of all the memory-elements at a given time define the state of that circuit at that time (FSM → Finite State Machine).
- FSM concept can be used for understanding progress of sequential logic with time.
- In FSM, functional behavior of the circuit is explained using finite number of states.
- State transition diagram is a very convenient tool to describe an FSM.

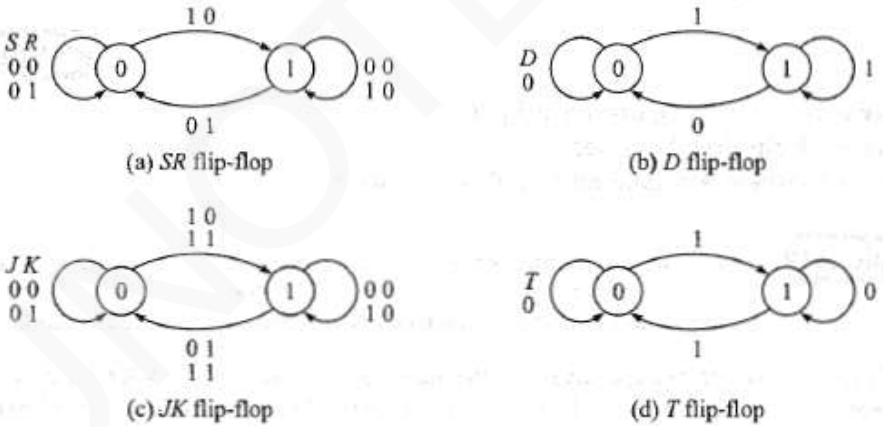


Figure 8.34: State transition diagram of (a) SR flip-flop, (b) D flip-flop, (c) JK flip-flop, (d) T flip-flop

- Let us see how state transition diagram for SR flip-flop is developed from its truth table or characteristic equation.
- Each flip-flop can be at either of 0 or 1 state defined by its stored value at any given time.
- Application of input may change the stored value, i.e. state of the flip-flop. (This is shown by directional arrow and the corresponding input is written alongside).
- For example (Fig 8.34a):
  - If SR flip-flop stores 0, then
    1) For SR = 00 or 01, the stored value does not change.
    2) For SR= 10, flip-flop output changes from 0 to 1.
  - If SR flip-flop stores 1, then
    1) For SR = 00 or 10, the stored value does not change.
    2) For SR= 01, flip-flop output changes from 1 to 0.

FLIP-FLOP EXCITATION TABLE

- Excitation table of a flip-flop is looking at its truth table in a reverse way.
- Here, flip-flop input
  - → is presented as a dependent function of transition Q_n --> Q_{n+1} and
  - → comes later in the table (Fig 8.35).

$Q_n \rightarrow Q_{n+1}$		S	R	J	K	D	T
0	0	0	×	0	×	0	0
0	1	1	0	1	×	1	1
1	0	0	1	×	1	0	1
1	1	×	0	×	0	1	0

Figure 8.35: Excitation table of flip-flops

- As shown in Excitation table;
  1) If present state(Q_n) is 0, then SR = 0X does not alter its value (1st entry in table).
  2) For SR = 10, transition occurs from State 0 to 1 (2nd entry in table).
  3) For SR = 01, transition occurs from State 1 to 0 (3rd entry in table).
  4) If present state(Q_n) is 1, then SR = X0 does not alter its value (4th entry in table).

ANALYSIS OF SEQUENTIAL-CIRCUITS

- A sequential logic circuit contains
  - → flip-flops as memory-elements &
  - → logic gates as combinatorial circuit elements.
- Analysis of a circuit helps to explain its performance.

Example 8.11:

Consider, the sequential-circuit shown in Fig. 8.37. It has only input CLK in the form of fixed frequency binary pulses that triggers both the flip-flops. An output X is generated from flip flop outputs as shown. Analysis of this circuit will give how flip-flop values (or states) and more importantly output X change with input CLK. The steps are as follows.

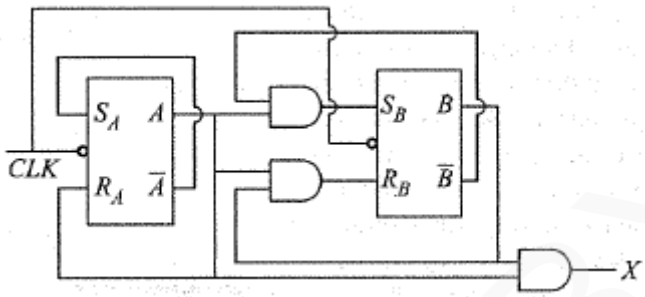


Figure 8.37: A sequential logic circuit for analysis purpose

Note from the circuit diagram flip-flop input relations: $S_A = A_n'$, $R_A = A_n$ and $S_B = A_nB_n'$, $R_B = A_nB_n$.

Next, using characteristic equation of SR flip-flop we can write,

for flip-flop A

$$\begin{aligned} A_{n+1} &= S_A + R_A'A_n \\ &= A_n' + A_n'A_n \text{ (Substituting } S_A = A_n' \text{ and } R_A = A_n) \\ &= A_n' \end{aligned}$$

and for flip-flop B

$$\begin{aligned} B_{n+1} &= S_B + R_B'B_n \\ &= A_nB_n' + (A_nB_n)'B_n \text{ (Substituting } S_B = A_nB_n' \text{ and } R_B = A_nB_n) \\ &= A_nB_n' + (A_n' + B_n')B_n \text{ (Following De Morgan's Theorem)} \\ &= A_nB_n' + A_n'B_n \\ &= A_n \oplus B_n \end{aligned}$$

Now the output from the given circuit, $X_n = A_nB_n$

- The equation shows that present values (given by time index n) of A and B flip-flop (also called states of the sequential-circuit) determine present output and next value (given by time index n + 1) of flip-flop (or the circuit).
- Thus, if present state is $B_n = 0$, $A_n = 0$ then present output is

  $X_n = A_nB_n = 0.0 = 0.$

  And at the end of first clock cycle, the next state is

  $B_{n+1} = 0 \ominus 0 = 0$

  $A_{n+1} = 0' = 1.$
- In next clock cycle, present state is next state of previous cycle or $B_n = 0$, $A_n = 1$.
- The output now is generated as

  $X_n = 0.1 = 0$ and

  Next state is determined as

  $B_{n+1} = 0 \ominus 1 = 1$, $A_{n+1} = 1' = 0.$
- Continuing this exercise, we arrive at state table as shown in Table 8.1.

Present State		*Present Input*				*Next State*		*Present Output*
B_n	A_n	$S_B = A_n B_n'$	$R_B = A_n B_n$	$S_A = A_n'$	$R_B = A_n$	$B_{n+1} = A_n \oplus B_n$	$A_{n+1} = A_n'$	$X = A_n B_n$
0	0	0	0	1	0	0	1	0
0	1	1	0	0	1	1	0	0
1	0	0	0	1	0	1	1	0
1	1	0	1	0	1	0	0	1
0	0	0	0	1	0	0	1	0
0	1	...	...				...	Repeats

Table 8.1: State Analysis Table for Analysis Example

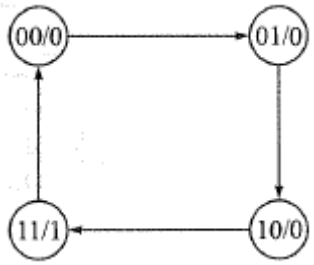


Figure 8.38: State transition diagram of the sequential-circuit given in Fig: 8.37

- The circuit thus behaves like a counter that
  → counts number of clock pulses that has arrived at its input and
  → signals when there is a count of four (Fig 8.38).
- The values within the circle follow syntax: $\mathbf{B_nA_n/X_n}$.
  1) Flip-flop outputs defining current state is shown to the left of '/' and
  2) Current output appears at right.
- There are two different models by which a synchronous sequential logic circuit can be designed.
  1) In **Mealy Model**, output is dependent both on current state and input to the circuit.
  2) In **Moore Model**, output is dependent only on current state of the circuit.

CONVERSION OF FLIP FLOPS: A SYNTHESIS EXAMPLE

• We show how to convert an SR flip-flop to a JK flip-flop.

Step 1:

➢ We look into JK flip-flop truth table and specifically note, Q_n-->Q_{n+1} transitions for a given combination of inputs JK and present state Q_n.

➢ Since the synthesis element is SR flip-flop, we shall refer to its excitation table to identify SR input combination for a required $Q_n \rightarrow Q_{n+1}$ transition.

➢ Table 8.2 shows truth table of JK flip-flop as well as necessary SR inputs for Q_n-->Q_{n+1} transitions.

• Such tables are also known as **state table**.

J_n	K_n	Q_n	$\rightarrow Q_{n+1}$	S_n	R_n
0	0	0	0	0	×
0	0	1	1	×	0
0	1	0	0	0	×
0	1	1	0	0	1
1	0	0	1	1	0
1	0	1	1	×	0
1	1	0	1	1	0
1	1	1	0	0	1

Table 8.2: State Synthesis Table for SR to JK Flip-Flop Conversion

Step 2:

➢ Write SR inputs as a function of JK inputs and present state Q_n.

➢ K-Map derived from Table 8.2 for SR inputs are shown in Fig. 8.42 along with their design equations.

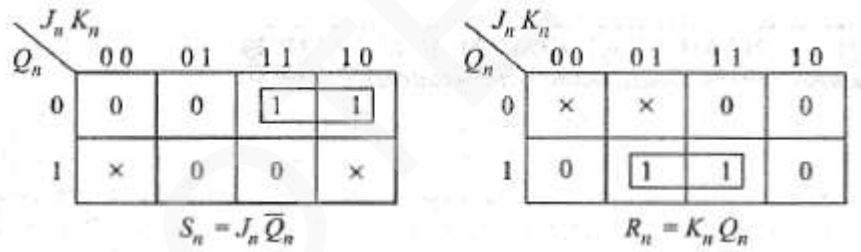


Figure 8.42: K-Map and Design equations for SR inputs

Step 3:

➢ The final synthesized circuit developed from these equations, are shown in Fig. 8.43.

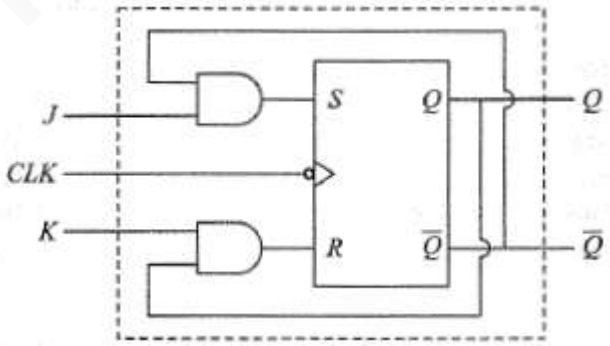


Figure 8.43:Conversion of SR flip-flop to JK flip-flop.

Example 8.13

Show how a D flip-flop can be converted to SR flip-flop.

Solution Note characteristic equation of two flip-flops.

For *SR* flip-flop: $Q_{n+1} = S + R'Q_n$ and for *D* flip-flop: $Q_{n+1} = D$

Thus with $D = S + R'Q_n$ we get circuit shown in Fig. 8.44 which behaves like an *SR* flip-flop but made from a *D* flip-flop and basic logic gates. Method as shown in Section 8.11 also gives same solution.

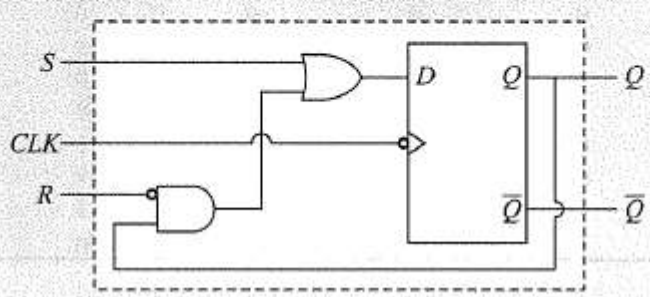


Figure 8.44: D flip-flop converted to SR flip-flop

HDL IMPLEMENTATION OF FLIP–FLOP

A verilog code for D latch.

```
module DLatch(D,EN,Q);
input D,EN;
output Q;
reg Q;
always @ (EN or D)
   if (EN) Q=D;

   //from characteristic equation
endmodule
```

A verilog code for SR latch.

```
module SRLatch(S,R,EN,Q);
input S,R,EN;
output Q;
reg Q;
always @ (EN or S or R)
   if (EN) Q=S|(~R&Q);
//from characteristic equation

endmodule
```

A verilog code for a D flip-flop with positive edge trigger.

```
module DFFpos(D,C,Q);
input D,C; //C is clock
output Q;
reg Q;
always@(posedge C)
  Q=D;
endmodule
```

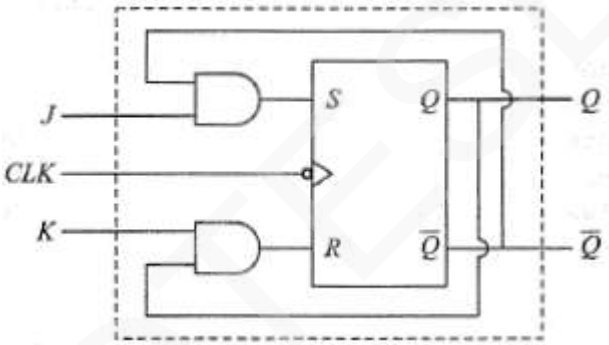
Write a verilog code for a D flip-flop with negative edge trigger.

```
module DFFneg(D,C,Q);
input D,C; //C is clock
output Q;
reg Q;
always@(negedge C)
  Q=D;
endmodule
```

Write a verilog code for a D flip-flop with positive edge trigger with reset (CLR).

```
module DFFpos_clr(D,C,CLR,Q);
input D,C,CLR; //C is clock
output Q;
reg Q;
always@(posedgeCornegedgeCLR)
   if (~CLR) Q=1'b0;
   //Q stores 1 binary bit 0
  else Q=D;
endmodule
```

Write a verilog code that converts a D flip-flop to an SR flip-flop.



Solution The code is given as follows. See how combinatorial logic part of the circuit is expressed by **assign** statement.

```
module SRFFneg(S,R,C,Q);
input S,R,C; //C is clock
output Q;
wire DSR;
assign DSR = S|(~R&Q); //combinatorial logic shown in fig.8.45
DFFneg  D1(DSR,C,Q);  //instantiates negative edge triggered D FF
endmodule

module DFFneg(D,C,Q);
input D,C; //C is clock
output Q;
reg Q;
always @ (negedge C)
    Q=D;
endmodule
```

Example 8.14

Explain the use of following Verilog code in test bench preparation of sequential logic circuit.

```
Initial
    begin
        clk = 1'b0;
        repeat (20)
        #50 clk = ~clk;
    end
```

Solution The keyword **initial** says following code is run for once. The variable 'clk' is of 1 binary digit and is initialized with 0 at time = 0. Keyword **repeat** ensures repetition of following statement 20 times. In that statement, variable clk is complemented after a delay of 50 ns. Thus, clk toggles between 1 and 0 every 50 ns and for 20 times generating 10 cycles of 50 + 50 = 100 ns duration each. In a test bench, clk can be fed as clock input to simulate a sequential circuit for a finite duration. The number of clock pulse generated can be changed by changing number after **repeat** and clock period can be changed by changing delay after # sign.

Example 8.15

Analyze the circuit shown in Fig. 8.46 and find the output Y. Consider that the flip-flops are initially reset.

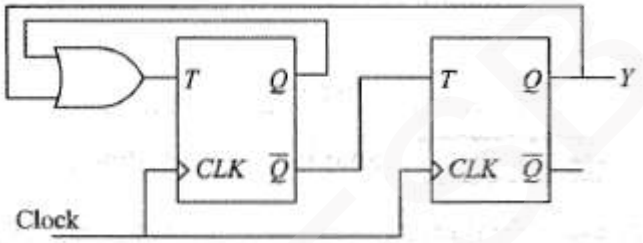


Figure 8.46: A T flip-flop based circuit for analysis purpose

Solution We follow three different methods to analyze the circuit and identify the performance of *Y*.

In Method-1, we use state table approach. We make use of the fact that a *T* flip-flop does not change its state if $T = 0$ but it toggles when $T = 1$ at the clock trigger.

Let us name the first flip-flop as *X* and its input and output as T_X and *X* respectively. Similarly, let the second flip-flop be named *Y* and its input is T_Y while its output is already assigned as *Y*. Then, the state table is shown in Fig. 8.47. We find that the circuits move from states 00, 01, 10, 00, ... repetitively and the output *Y* goes HIGH once in three cycles and remains HIGH for one clock period.

Clock cycle	X_n	Y_n	$T_X = X_n + Y_n$	$T_Y = X'_n$	X_{n+1}	Y_{n+1}
0	0	0	0	1	0	1
1	0	1	1	1	1	0
2	1	0	1	0	0	0
3	0	0	... repeats ...			

Figure 8.47: State table to analyze circuit diagram of Fig 8.46

In Method-2, we make use of the characteristic equation of *T* flip-flop.

we know that $Q_{n+1} = TQ'_n + T'Q_n$

By following similar X and Y naming of two flip-flops as in Method-1, we find that

For X flip-flop,
$$\text{input: } T_X = X_n + Y_n$$
$$\text{output: } X_{n+1} = (X_n + Y_n)X_n' + (X_n + Y_n)'X_n$$
$$= Y_n X_n' + X_n' Y_n' X_n$$
$$= Y_n X_n'$$

For Y flip-flop,
$$\text{input: } T_Y = X_n'$$
$$\text{output: } Y_{n+1} = X_n' Y_n' + (X_n')' Y_n$$
$$= X_n' Y_n' + X_n Y_n$$

The final solution is shown in Fig. 8.48.

X_n	Y_n	$X_{n+1} = Y_n X_n'$	$Y_{n+1} = X_n' Y_n' + X_n Y_n$
0	0	0	1
0	1	1	0
1	0	0	0
0	0	... repeats	...

Figure 8.48: Solution using Method-2

In Method-3, we make use of the timing diagram as shown in Fig. 8.49. We note that the flip-flops are positive edge triggered. The T input just before the positive edge decides output of the flip-flop in next cycle.

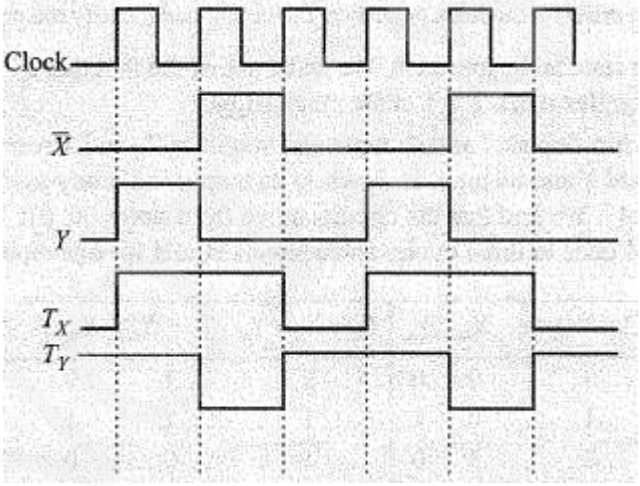


Figure 8.49: Solution using Method-3

We start with initial $XY = 00$. Then we draw T_X by ORing X and Y waveforms and T_Y by inverting Y waveform. T_X and T_Y before positive edge decide value of X and Y respectively in next clock cycle (from T flip-flop truth table).

MODULE 4 (CONT.): REGISTERS

REGISTERS

- A register is a group of flip-flops used to momentarily store binary-information (e.g. 1101).
- Each flip-flop can store either 0 or 1.
- The flip-flops used to construct registers are usually edge-triggered JK, SR or D types.
- The register can also be used
  - → to accept input-data from an alphanumeric keyboard and then present this data at the input of a microprocessor-chip.
  - → to momentarily store binary-data at the output of a decoder.
  - → to perform various arithmetic operations. For ex: multiplication & division.
  - → to count no. of pulses entering into a system as up-counter, down-counter or ring-counter
  - → as serial-adder, sequence-generator/detector.
- UART(Universal Asynchronous Receiver Transmitter) is a chip used to exchange data in a microprocessor-system.

TYPES OF REGISTERS

- Shift-register is a group of flip-flops connected in such a way that a binary-number can be shifted into or out of the flip-flops.
- The bits in a binary-number can be moved from one place to another in following 2 ways:

  1) Serial Shifting

  ➢ Data-bits are shifted one after the other in a serial fashion with 1 bit shifted at each clock-transition.

  ➢ Therefore, n clock-transitions are needed to shift an n-bit binary number. (Figure: 9.1).

  2) Parallel Shifting

  ➢ Data-bits are shifted simultaneously with a single clock-transition.

  ➢ Therefore, 1 clock-transition is needed to shift an n-bit binary number.
- Shift-register types are
  - → Serial in-Serial out e.g. 7491, 8 bits
  - → Serial in-Parallel out e.g. 74164, 8 bits
  - → Parallel in-Serial out e.g. 74165, 8 bits
  - → Parallel in-Parallel out e.g. 74198, 8 bits

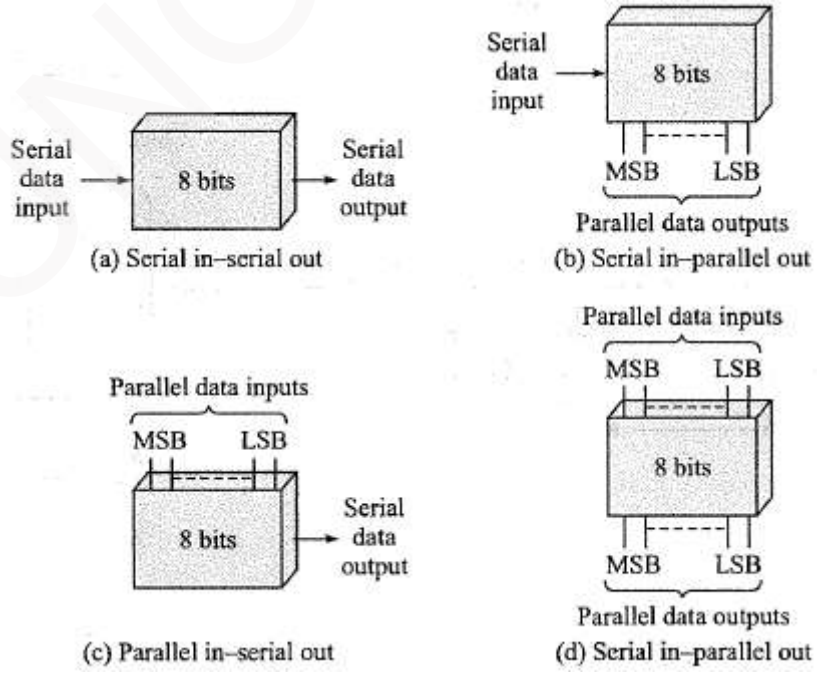


Figure: 9.1: Shift-register types

SERIAL IN-SERIAL OUT (SISO)

- Here, data can be shifted either into or out of the register in serially
- At each clock-transition, one bit is shifted (either left or right).
- Here we consider 2 cases: 1) Shifting data into register & 2) Shifting data out of register.

1) SHIFTING BINARY-DATA INTO THE REGISTER

- A common clock drives all the flip-flops (Figure 9.2).
- The output of one flip-flop is connected to input of the next.
- At every clock transition, data stored in one flip-flop is transferred to the next flip-flop.
- Data-transfer takes place like this

  D->Q
  Q->R
  R->S and
  S->T where D=serial data-input

- Suppose that 4-bit number 0100 has to be loaded into the register.
  - The bits are all shifted one flip-flop to the right.
  - When clock signals are applied, following events happen:

    Before Time A, all the flip-flops are initially cleared, QRST=0000.
    At clock edge A, serial data-in D=0, i.e. DQRS=0000. So after NT at A, QRST=0000.
    At clock edge B, serial data-in D=0, i.e. DQRS=0000. So after NT at B, QRST=0000.
    At clock edge C, serial data-in D=1, i.e. DQRS=1000. So after NT at C, QRST=1000.
    At clock edge D, serial data-in D=0, i.e. DQRS=0100. So after NT at D, QRST=0100.

- A shift-register made up of JK flip-flops has

  1) Non-inverting output Q of one flip-flop connected to J input of next flip-flop and
  2) Inverting output Q' of one flip-flop connected to K input of next flip-flop.

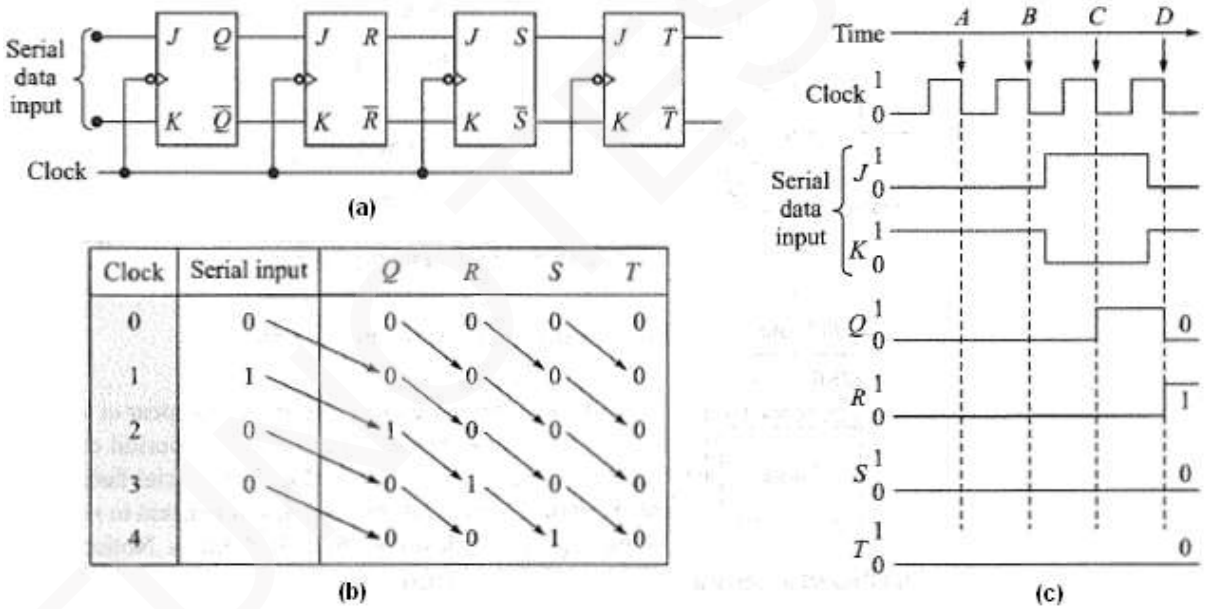


Figure 9.2: 4-bit serial input shift-register

2) SHIFTING BINARY-DATA OUT OF THE REGISTER

- Suppose that the register has the 4-bit number QRST=1010 stored in it.
  - We want to remove binary-data out of the register.
  - When clock signals are applied, following events happen (Figure 9.4):

    Before Time A, the register holds the bits QRST=1010.
    At clock edge A, serial data-in D=0, i.e. DQRS=0101. So after NT at A, QRST=0101.
    At clock edge B, serial data-in D=0, i.e. DQRS=0010. So after NT at B, QRST=0010.
    At clock edge C, serial data-in D=0, i.e. DQRS=0001. So after NT at C, QRST=0001.
    At clock edge D, serial data-in D=0, i.e. DQRS=0000. So after NT at D, QRST=0000.

- Thus, the binary-data stored is removed after 4 clock cycles.

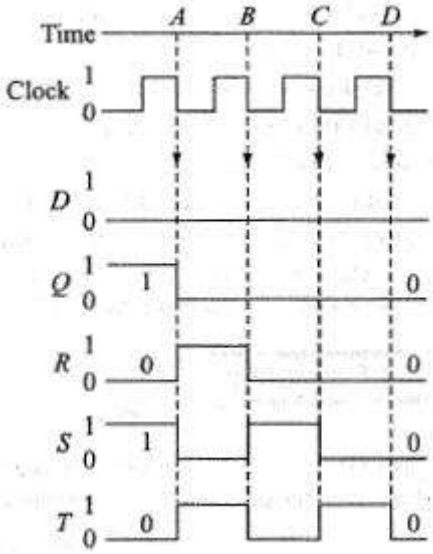
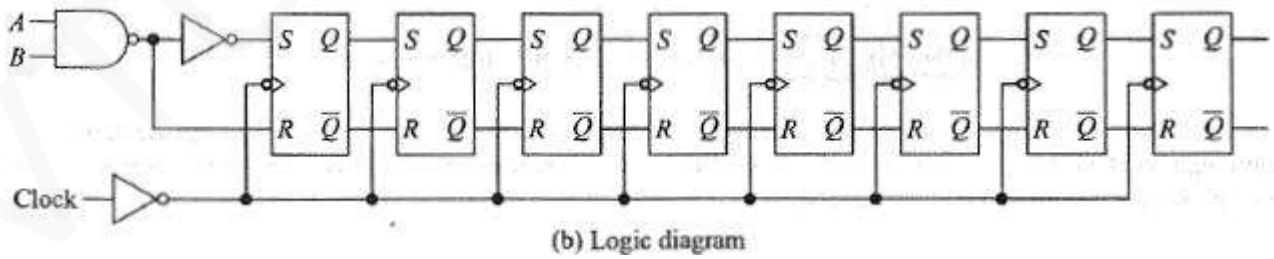


Figure 9.4: Waveform for shifting 1010 out of register

7491 8-BIT SHIFT-REGISTER

- As shown in logic diagram (Figure 9.5), 7491 register has eight RS flip-flops.
- The RS flip-flops are connected to provide
  - → serial input &
  - → serial output.
- The data will be shifted on the positive clock-transition.
- The inverter connected between R and S on the first flip-flop means that this circuit functions as a D-type flip-flop.
- The data input is applied at either A or B input of NAND gate.
- **How data-bit is sent to R input of first FF?**
  - ➢ At input 'A', data level (0 or 1) is complemented by the NAND gate and
  - ➢ Then, the data level is applied to the R input of the first flip-flop.
- **How data-bit is sent to S input of first FF?**
  - ➢ At input 'A', the data level is complemented by the NAND gate;
  - ➢Then, the data level is complemented again by the inverter and
  - ➢ Finally, the data level is applied to the S input of the first flip-flop.
- On a positive clock-transition,
  1) A '1' at input 'A' will set the flip-flop i.e. this 1 is shifted into the first flip-flop.
  2) A '0' at input 'A' will reset the flip-flop i.e. this 0 is shifted into the first flip-flop.
- If gating is desired, the NAND gate with inputs A and B simply provides a gating function for the input data-stream.

  If gating is not desired, simply short A & B together and apply the input data-stream to this connection.

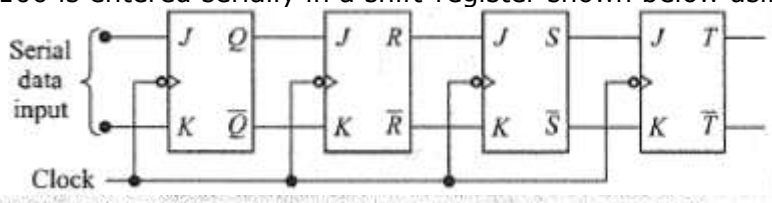


(b) Logic diagram

Figure 9.5:74191 8-bit shift-register

Example 9.1

Show how a number 0100 is entered serially in a shift-register shown below using state table.



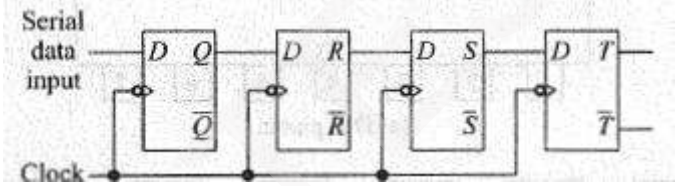
Solution Figure 9.3 presents the state table. The timing diagram corresponding to this is discussed in this section. Note how the data flow across the flip-flops is highlighted by arrow direction.

Clock	Serial input	Q	R	S	T
0	0	0	0	0	0
1	1	0	0	0	0
2	0	1	0	0	0
3	0	0	1	0	0
4		0	0	1	0

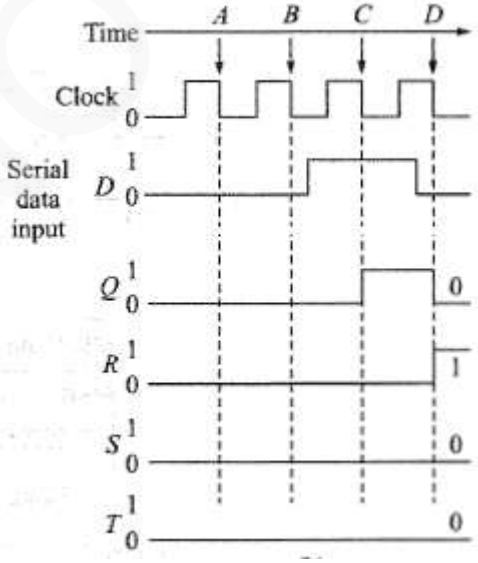
Figure 9.3: Detail transfer through serial input in a shift-register

Example 9.2

Draw the waveforms to shift the number 0100 into the shift-register shown below.



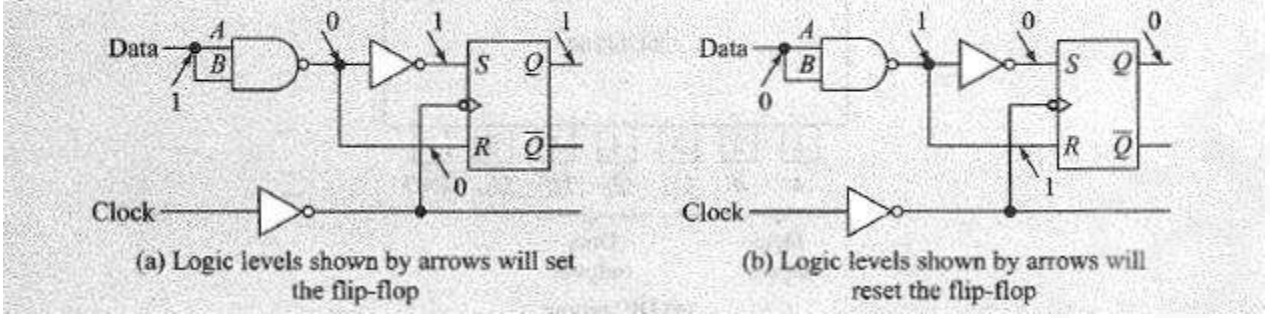
Solution:



Example 9.3

Examine the logic levels at the input of a 74LS91 and show how a 1 and then a O are shifted into the register.

Solution The input logic and the first flip-flop are redrawn in Fig. 9.6a, and a 1 is applied at the data input *A*. The *R* input is 0, the *S* input is 1, and the flip-flop will clearly be set when the clock goes high. In other words, the 1 at the data input will shift into the flip-flop. In Fig. 9.6b, a 0 is applied at the data input *A*. The *R* input is 1, the *S* input is 0, and the flip-flop will be reset when the clock goes high. The input 0 is thus shifted into the flip-flop.



(a) Logic levels shown by arrows will set the flip-flop

(b) Logic levels shown by arrows will reset the flip-flop

Figure 9.6

Example 9.4

How long will it take to shift an 8-bit number into a 54164 shift-register if the clock is set at 10 MHz? When must the input data be stable? When can it be changed?

Solution A minimum of eight clock periods will be required since the data is entered serially. One clock period is 100 ns, so it will require 800 ns minimum.

The data must be stable from 30 ns before a positive clock transition until the positive transition occurs. This leaves 70 ns during which the data may be changing (see Fig. 9.8).

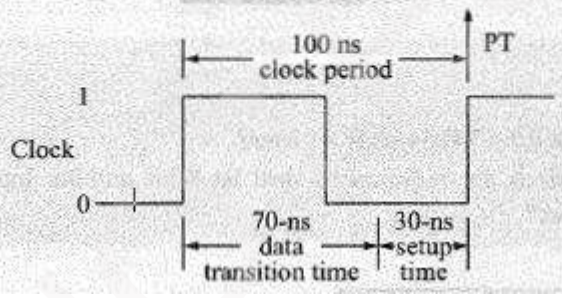


Figure 9.8

SERIAL IN-PARALLEL OUT (SIPO)

- Here, binary-data is shifted into register serially, but shifted out in parallel fashion.
- The 74164 is an 8-bit SIPO shift-register.
- It is constructed by using eight RS flip-flops (Figure 9.7).
- The data will be shifted on the clock-transition.
- 74164 is similar to 7491 with two exceptions:
  1) All 8 bits of any number stored in the register are available simultaneously as an output.
  2) Each flip-flop has an asynchronous clear-input.
- **Clear** is asynchronous input i.e. it can be activated at any time.
  A low level at the clear-input will immediately reset all flip-flops low (0000 0000)
- It has a setup time of 30 ns and hold time of 0.0 ns.
- Suppose that the serial-data is connected to A, then B can be used as a control-line.
- Here's how the control-line works:

  B is held high
  - The NAND gate is enabled and the serial input-data passes through the NAND gate inverted.
  - The input-data is shifted serially into the register (Figure 9.9).

  B is held low
  - The NAND gate is disabled and the serial input-data cannot passes through the NAND gate.
  - The next positive clock-transition will shift a 0 into the first flip-flop.
  - After 8 clock pulses, the register will be full of zeros ($Q_AQ_BQ_CQ_D$ $Q_EQ_FQ_GQ_H$=0000 0000).

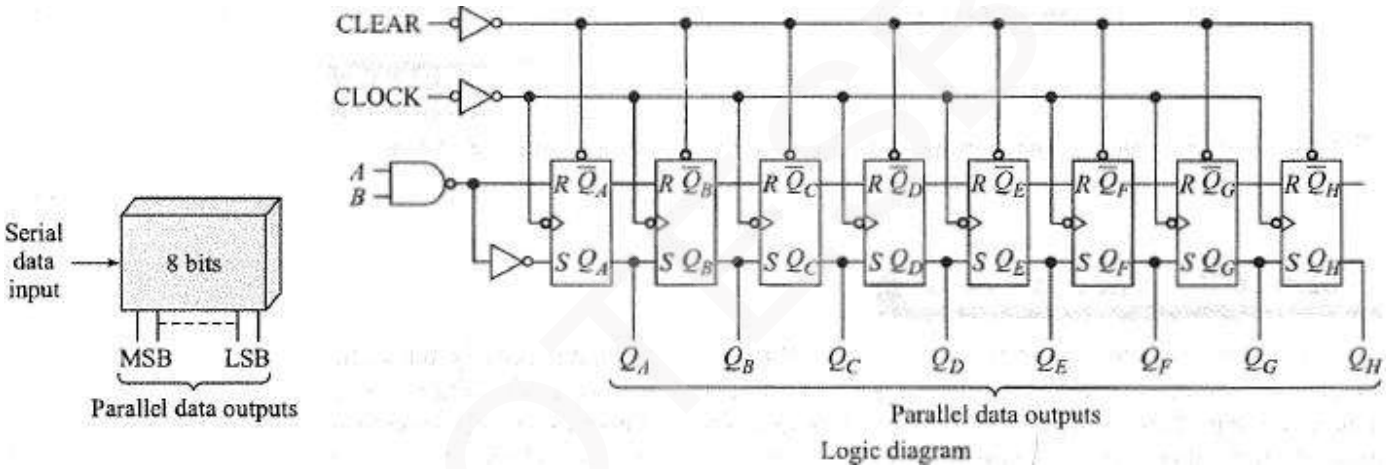


Figure 9.7:74164 8-bit shift-register

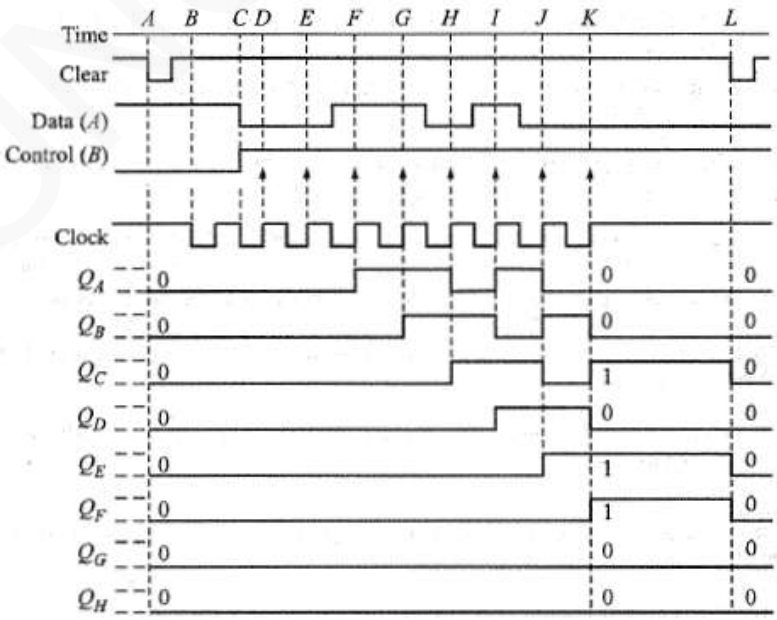


Figure 9.9:Waveform for entering data 00110100 into register

PARALLEL-IN SERIAL-OUT (PISO)

• Here, binary-data is shifted into register parallelly, but shifted out in serial fashion.

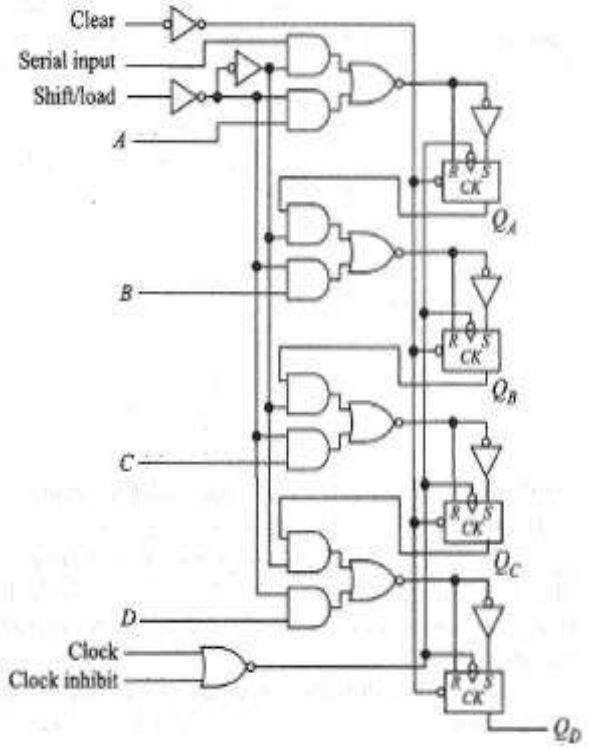


Figure 9.10: 74166

Inputs						Internal Levels		Outputs
Clear	Shift/ load	Clock inhibit	Clock	Serial	Parallel A ... D	Q_A and Q_B		Q_D
L	X	X	X	X	X	L	L	L
H	X	L	L	X	X	Q_{AO}	Q_{BO}	Q_{D0}
H	L	L	↑	X	a . . . d	a	b	d
H	H	L	↑	H	X	H	Q_{An}	Q_{Cn}
H	H	L	↑	L	X	L	Q_{An}	Q_{Cn}
H	X	H	↑	X	X	Q_{AO}	Q_{BO}	Q_{D0}

X = Irrelevant, H = High level, L = Low level
↑ = Positive transition
a . . . d = Steady state input level at A . . D respectively
Q_{AO}, Q_{BO} = Level at Q_A, Q_B . . . before steady state
Q_{An}, Q_{Cn} = Level of Q_A or Q_B before most recent transition ( ) ↑

Figure 9.12: 74166 truth table

• Four flip-flops are connected to allow 2 different operations:
  1) The parallel entry of data and
  2) The operation of shifting data serially through the register from the first flip-flop Q_A toward the last flip-flop Q_D.
• 74166 has
  → four inputs: A, B, C, D.
  → one output: Q_D(Figure 9.10).
• **Shift/Load** input operates as follows:
  1) When **Shift/Load = LOW**.
  ➢ A single clock-transition loads 4 bits of data (ABCD) into the register in parallel.
  2) When **Shift/Load = HIGH**.
  ➢ Clock-transitions will shift data through the register serially, with entering data applied at the SERIAL INPUT.
• **Clock** is applied through a two-input NOR gate.
  1) When **clock inhibit = LOW**.
  ➢ The clock signal passes through the NOR gate inverted.
  ➢ Since the register flip-flops respond to NTs, data will shift into register on the PTs of clock.
  2) When **clock inhibit = HIGH**.
  ➢ The NOR gate output is held low, and the clock is prevented from reaching the flip-flops. In ➢this mode, the register can be made to stop and hold its contents.
• **Clear** is asynchronous input i.e. it can be activated at any time.
  A low level at the clear-input will immediately reset all flip-flops low (0000)

PARALLEL IN-PARALLEL OUT (PIPO)

- Here, data can be shifted either into or out of the register in parallel (Figure 9.13).
- Each flip-flop is negative-edge-triggered, and thus a PT will shift data into the register.
- The 6 data bits (D_1 to D_6) are all loaded into the register in parallel.
- The loaded-data is immediately available in parallel, at the outputs (Q_1 to Q_6).
- Since this type of register is used to store data, it is also called a **data-latch**.
- Limitation: It is not possible to shift stored data either to the right or to the left.
- **Clear** is asynchronous input i.e. it can be activated at any time.

  A low level at the clear-input will immediately reset all flip-flops low (000000)

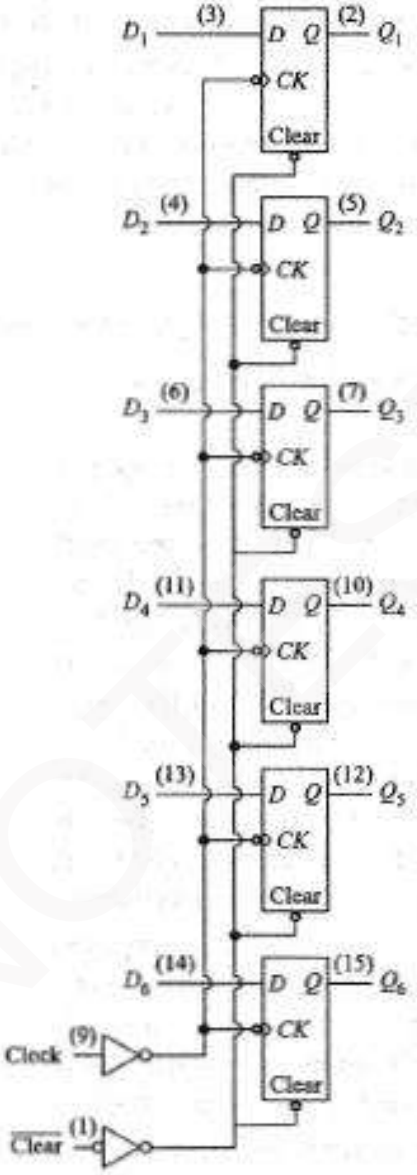
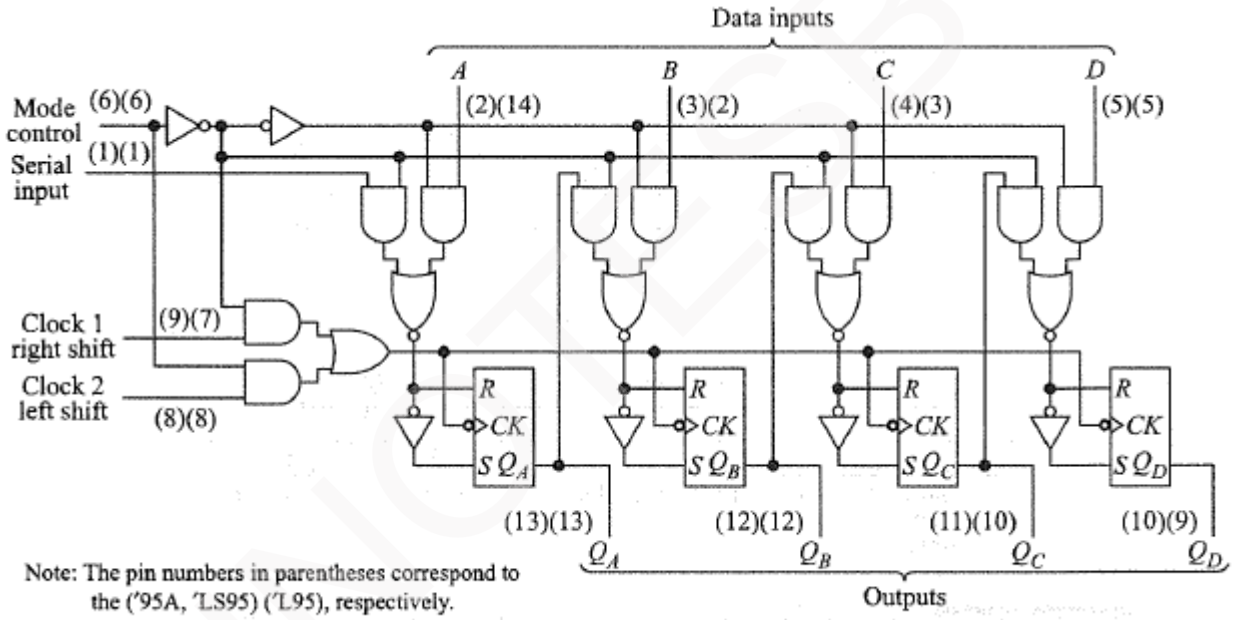


Figure 9.13: 74174

4-BIT PARALLEL ACCESS SHIFT-REGISTER (7495A)

- It can be used for entering/removing data in parallel (Figure: 9.15).
- It can also be used to shift data to the right (from Q_A->Q_B) and to the left (from Q_B->Q_A).
- The parallel data-outputs are simply the Q sides of each of the four flip-flops in the register.
- **Mode control line** operates as follows:
  1) When **mode control line = HIGH**.
  - The AND gates on the right input to each NOR gate is enabled while the left AND gates are disabled.
  - The data inputs, A, B, C and D will then be loaded into the register on a next clock-transition (this is parallel data input).

  2) When **mode control line = LOW**.
  - The AND gates on the right input to each NOR gate is disabled while the left AND gate is enabled.
  - On each clock NT, a data bit is entered serially into the register at the first flip-flop Q_A and each stored data bit is shifted one flip-flop to the right.
  - This is the serial input of data, and also the right-shift operation.
- The connections described here can either be hard wired or can be made by means of logic gates.
- There are two clock inputs:
  1) **Clock-1** is used for right-shift operation.
  2) **Clock-2** is used for left-shift operation.



(b) Logic diagram

Figure 9.15:7495A

UNIVERSAL SHIFT-REGISTER (74194)

- Four basic types of shift-register are:
  1) Serial in serial out 2) Serial in parallel out
  3) Parallel in serial out 4) Parallel in parallel out.
- Serial in or serial out can be made possible by shifting data in any of the two directions:
  1) Left shift ($Q_A \leftarrow Q_B \leftarrow Q_C \leftarrow Q_D \leftarrow$ Datain) 2) Right shift (Datain $\rightarrow Q_A \rightarrow Q_B \rightarrow Q_C \rightarrow Q_D$).
- A universal shift-register
  → can perform all the 4 operations &
  → is also bidirectional in nature (Figure 9.18).
- In 74194 register, there are 2 separate inputs for serial data for left and right shift.
- In addition, there are two mode control inputs which select the mode of operation for the universal shift-register according to below Table 9.1.

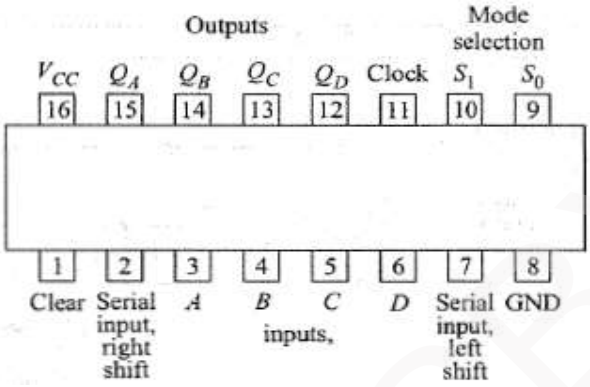


Figure 9.18: 74194 pinout

Table 9.1: Function table of 74194

Mode	*Control*	*Function*	*Next State (n+1-th state)*			
S_1	S_0		$Q_{A,n+1}$	$Q_{B,n+1}$	$Q_{C,n+1}$	$Q_{D,n+1}$
0	0	Hold	$Q_{A,n}$	$Q_{B,n}$	$Q_{C,n}$	$Q_{D,n}$
0	1	Shift right	Data in	$Q_{A,n}$	$Q_{B,n}$	$Q_{C,n}$
1	0	Shift left	$Q_{B,n}$	$Q_{C,n}$	$Q_{D,n}$	Data in
1	1	Load	A	B	C	D

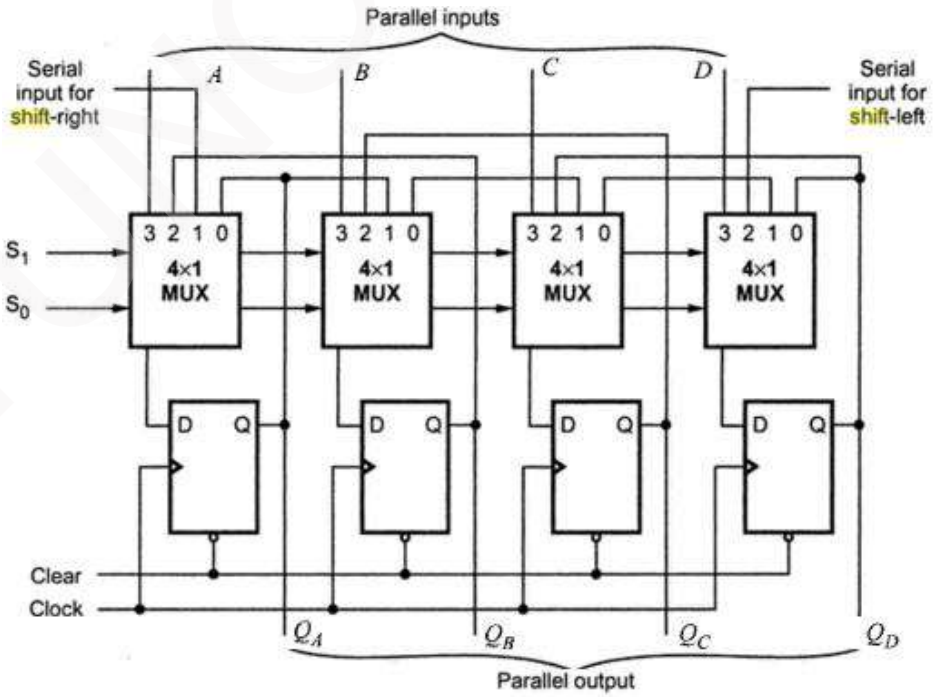
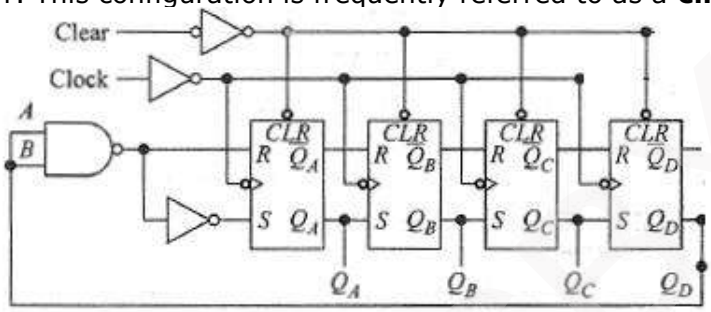


Figure 9.19: 74194, 4-bit universal shift-register

APPLICATIONS OF SHIFT-REGISTER

RING COUNTER

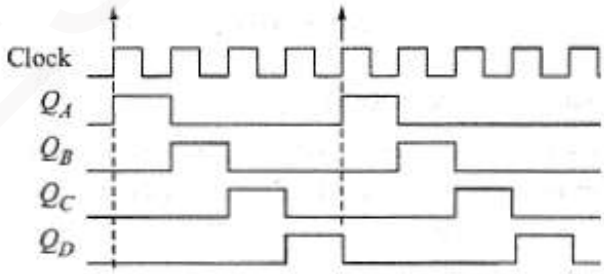
- It is a shift-register with output of last flip-flop fed to first flip-flop input
- The contents of the register simply circulate around the register when the clock is running.
- Here is how it works (Figure 9.20):
  - Suppose that FF-A is high(1) and all other flip-flops are low(0).
  - The first clock pulse will shift the 1 from FF-A to FF-B, while FF-A resets (since the 0 in D will shift into A).
  - The second clock pulse will shift the 1 from FF-B to FF-C, while FF-B resets.
  - The third clock pulse will shift the 1 from FF-C to FF-D, while FF-C resets.
  - The fourth clock pulse will shift the 1 from FF-D to FF-A, while FF-D resets. (By means of the feedback connection).
- The ping-pong balls simply circulate around the register in a clockwise direction, moving ahead one flip-flop with each clock PT. This configuration is frequently referred to as a **circulating register**.



(a)

Clock	Serial in=Q_D	Q_A	Q_B	Q_C	Q_D
0	0	1	0	0	0
1	0	0	1	0	0
2	0	0	0	1	0
3	1	0	0	0	1
4	0	1	0	0	0
5	0	0	1	0	0
6	0	0	0	1	0
7	1	0	0	0	1
8	0	1	0	0	0 repeats

(b)



(c)

Figure 9.20: Ring Counter (a) 4-bit shift-register with feedback line from Q_D to A-B
(b) truth table (c)Waveform when register has a single 1 and three 0s

- Ring counters are frequently used in the control section of a digital system.
- They are ideal for controlling events that must occur in a strict time sequence-that is, event A, then event B, then C, and so on.

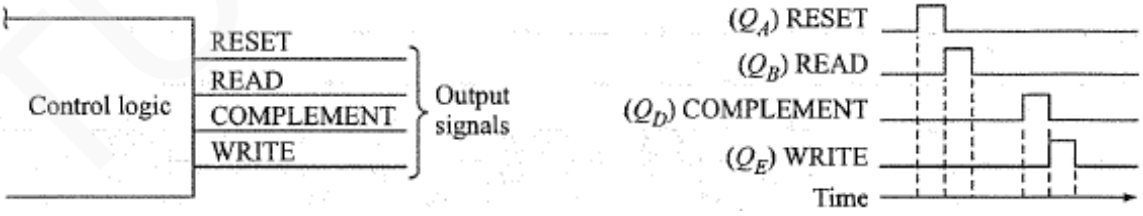


Figure 9.21: control section of a digital system

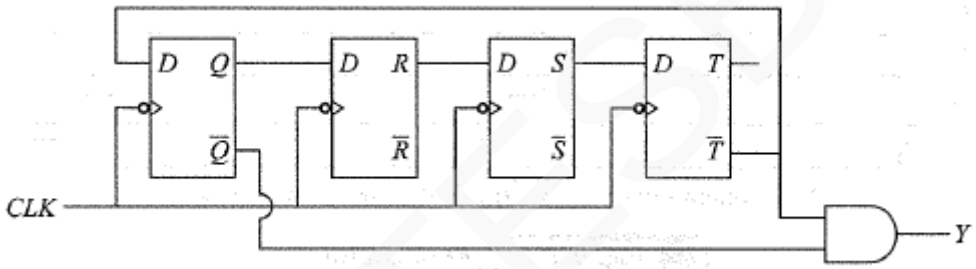
Application of Ring Counter

- As shown in Figure 9.21, RESET, READ, COMPLEMENT, and WRITE control signals can be generated as a set of control pulses that occur one after the other sequentially.
- **Drawback:**
  - In order to produce required waveforms, the counter should have one, and only one, 1 in it.
  - If the flip-flops are in the reset state when power is first applied, it will not work at all.

Solution: The counter has to be preset to the desired state before it can be used.

JOHNSON COUNTER (SWITCHED TAIL COUNTER)

- It is a shift-register with inverting output of last flip-flop fed to first flip-flop input (Figure 9.22).
- Here is how it works:
  - Consider 4-bit counter. Initially, all the flip-flops are cleared. i.e. QRST=0000.
  - When first clock trigger occurs, flip-flop stores QRST=1000.
    The output of last flip-flop is 0.
    Therefore, complement output of last flip-flop is 1.
    This is connected back to the D input of first flip-flop. So, D=1.
  - When second clock trigger occurs, flip-flop stores QRST=1100.
  - When third clock trigger occurs, flip-flop stores QRST=1110.
  - When fourth clock trigger occurs, flip-flop stores QRST=1111.
- Output of circuit is Y = Q'T'.
- The state of the circuit repeats every eighth clock cycle. Thus, this 4-bit shift-register circuit can count 8 clock pulses (or called modulo-8 counter).
- In general, for any N-bit shift-register.
  - Johnson counter
    → can count up to 2N number of clock pulse and
    → gives modulo-2N counter.
  - The output Y gives a logic high at every 2N-th clock cycle.
- The two-input AND gate which decodes states repeating in the counter is called **decoding gate**.
  Decoding gate generates the output that signals counting of a given number of clock pulses.



(a)

Clock	Serial in = T	Q	R	S	T	$Y = Q'T'$
0	1	0	0	0	0	1
1	1	1	0	0	0	0
2	1	1	1	0	0	0
3	1	1	1	1	0	0
4	0	1	1	1	1	0
5	0	0	1	1	1	0
6	0	0	0	1	1	0
7	0	0	0	0	1	0
8	1	0	0	0	0	1
9	1	1	0	0	0	0 repeats

(b)

Figure 9.22: (a) 4 bit Johnson Counter b)it's state table

- **Drawback:**
  - Decoding complexity increases with increasing number of flip-flops.
  - The counter is to be initialized with one of the valid state of the counting sequence on which the design is based. Otherwise, the counter will follow a completely different state sequence (mutually exclusive) and decoding will not be proper.

SEQUENCE GENERATOR AND SEQUENCE DETECTOR

Sequence Generator

- This is used to generate a prescribed-sequence repetitively (Figure 9.23a).
- The sequence may be
  - → synchronizing bit pattern sent by a digital data transmitter or
  - → control word directing repetitive control task.
- This uses shift-register which contains 4 flip-flops.
- The leftmost flip-flop is connected to serial data-in.
  Rightmost flip-flop is connected to serial data-out.
- When clock trigger occurs, data-transfer takes place.
- The shift-register is connected like a ring-counter.
  - ➢ When clock trigger occurs, the binary-word stored in the register comes out sequentially from serial out but does not get lost.
  - ➢ The binary-word is fed back as serial-in to fill the register.

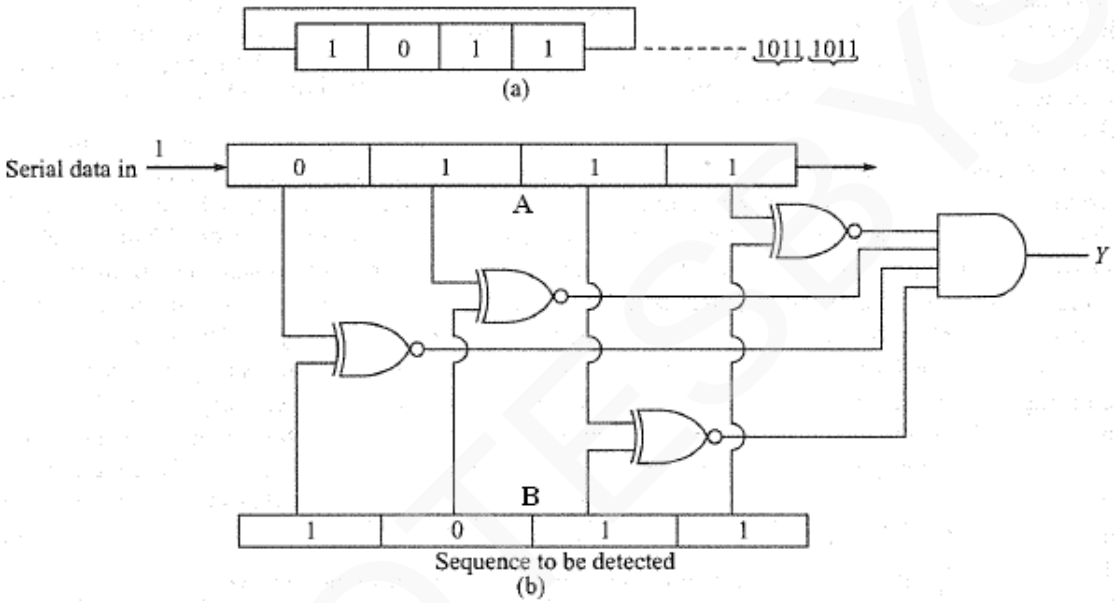


Figure 9.23: (a)4-bit sequence generator (b)4 bit sequence detector

Sequence Detector

- Sequence detector
  - → checks binary data-stream and
  - → generates a signal when a particular sequence is detected (Figure 9.23b).
- Basically, this has 2 registers:
  1) First register A is used to store binary-word of the incoming data-stream.
  2) Second register B is used to store binary-word which has to be detected from data-stream.
- Here is how it works:
  - ➢ Input-data-stream
    - → enters a second register A as serial data-in and
    - → leaves a second register A as serial-out.
  - ➢ At every clocking instant, bit-wise comparisons of the 2 registers (A & B) are done through Ex-NOR gate.
  - ➢ Two-input Ex-NOR gives logic HIGH when
    - → both inputs are low or
    - → both inputs are high.
  - ➢ Final output is taken from 4-input AND gate, which becomes 1 only when all its inputs are 1.
  - ➢ If all the bits of 2 registers are matched,
    then output of AND gate will be 1 (i.e. sequence detection successful).
    If any bit of 2 registers is mismatched,
    then output of AND gate will be 0 (i.e. sequence detection unsuccessful).

SERIAL ADDER

- Serial Adder is used to add 2 binary numbers.
- Let two numbers to be added are
  A_7 ... A_0 and
  B_7 ... B_0,
  Let Sum is S_7 ... S_0.
- Here is how an addition is done:

$$\begin{array}{ll} A_7A_6A_5A_4 & A_3A_2A_1A_0 \\ +\ B_7B_6B_5B_4 & B_3B_2B_1B_0 \\ \hline S_7S_6S_5S_4 & S_3S_2S_1S_0 \end{array} \qquad \begin{array}{r} 0000\ 0111\ (+7) \\ +\ 0000\ 1000\ (+8) \\ \hline 0000\ 1111\ (+15) \end{array}$$

- This
  → converts parallel data to serial and
  → uses full-adder(FA) block sequentially (Figure 9.24).
- Here is how it works:
  - Two 8-bit numbers to be added are loaded in two 8-bit shift-registers A & B.
  - Full-adder accepts 3 data-inputs:
    1) Serial-data out of shift-register A.
    2) Serial-data out of shift-register B.
    3) Carry-in(C_{i-1})
  - The carry-in(C_{i-1}) is fed from FA's own carry output(C_i) delayed by 1 clock period by D flip-flop
  - The sum(S) output of FA(full adder) is fed to serial data-in of shift-register A.
  - Both registers and D flip-flop are driven by same clock.
- The addition takes place as follows:
  - During 1st clock cycle, full-adder (FA)
    → adds the LSBs of two numbers(A_0 and B_0) appearing at serial out of registers(A & B) &
    → generates sum(S_1) and carry(C_1).
  - S_1 is available at serial data input of register A.
    C_1 is available at input of D flip-flop.
  - When clock trigger occurs, registers shift its content to right by one bit.
  - S_1 becomes MSB of register A.
    C_1 appears at D flip-flop output.
    X (don't care condition) is entered as MSB of register B.
  - This process goes on and is stopped after 8 clock cycles.

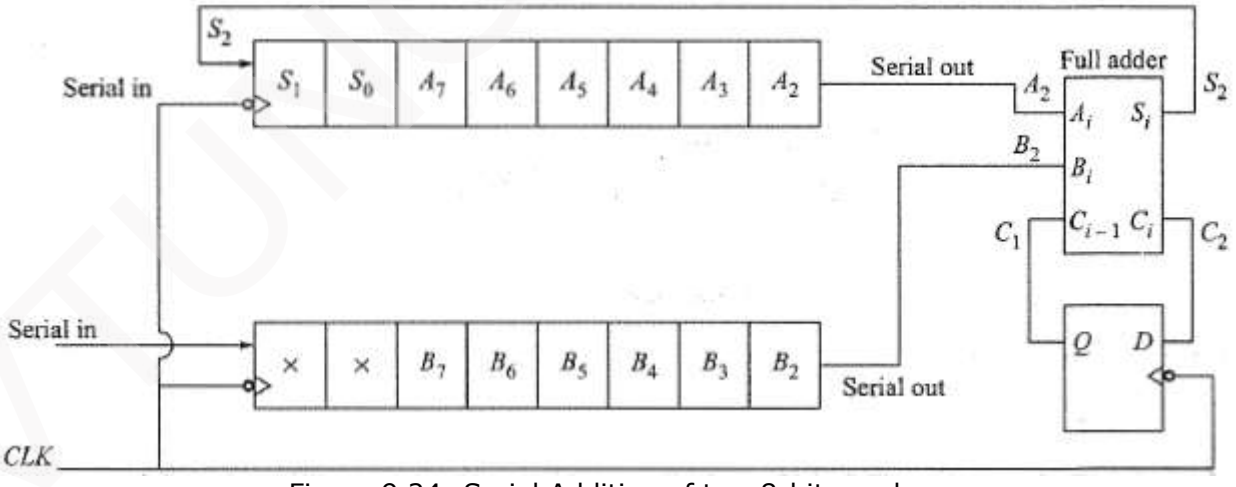


Figure 9.24: Serial Addition of two 8-bit numbers

- **Drawback:** The final addition result is delayed by 8 clock cycles.
  Solutions:
  1) Use parallel adder.
  2) Using a high frequency clock, the delay factor can be reduced.

Write a verilog code for a shift-register of 5 bits constructed using D flip-flops

```
module Reg74174(D, Clock, Clear, Q);
    input Clock, Clear;
    input [5:0] D;
    output [5:0] Q;
    reg [5:0] Q;
    always @ (negedge Clock or negedge Clear)
    if(~Clear)
       Q=6'b0;            //Q stores 6 binary 0
    else
       Q=D;
endmodule
```

Write a verilog code for 4-bit serial input shift-register

```
module ShiftReg(D, Clock, T);
    input Clock, D;
    output T;
    reg T;
    reg Q,R,S;
    always @ (negedge Clock)
    begin
      Q<=D;
      R<=Q;
      S<=R;
     T<=S;
    end
endmodule
```

Write a verilog code for 4-bit parallel out right shift-register

```
module ShiftRegister(D, Clock D);
    input Clock, D;
    output [3:0] Q;
    always @ (negedge Clock)
    begin
      Q[0]<=D;
      Q[1]<=Q[0];
      Q[2]<=Q[1];
      Q[3]<=Q[2];
    end
endmodule
```

Write a verilog code for Johnson counter

```
module JohnsonC(Clock,Clear,Y);
    input Clock,Clear;
    output Y;
    reg Q,R,S,T;
    assign Y=(~Q) & (~T);
    begin
    if(~Clear)
      Q=6'b0;
    else
      begin
        Q<=~T;          //tail is switched and connected to input
        R<=Q;
        S<=R;
        T<=S;
      end
endmodule
```

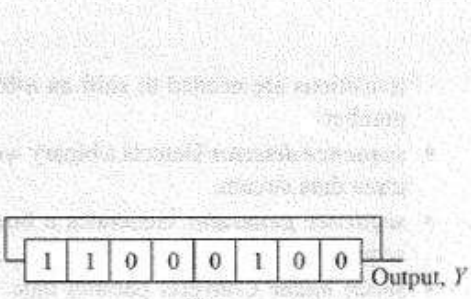
Example 9.5

Design an 8-bit sequence generator that generates the sequence 11000100 repetitively using shift-register.

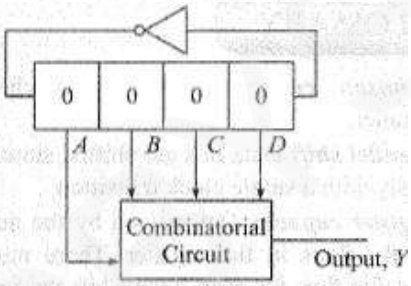
Solution We use ring counter and switched-tail counter derived from shift registers for this purpose.

In Method-1, we load an 8-bit ring counter as shown in Fig. 9.27a with the given sequence and at the output, the sequence will be repetitively generated.

In Method-2, we consider a modulo-8 switched-tail counter developed from 4-bit shift register. Let it be initially loaded with 0000. Then the 8 repetitive states of the counter will be as shown in Fig. 9.27b and is reproduced in Fig. 9.27c. We then design a combinatorial circuit which for each of the state generates one bit of the sequence. The Karnaugh Map for this is shown in Fig. 9.27d. Note that the unused states can be considered as 'don't care'. The logic equation of the combinatorial circuit realized as Fig. 9.27b can be written as $Y = A'D' + A'B + AB'$



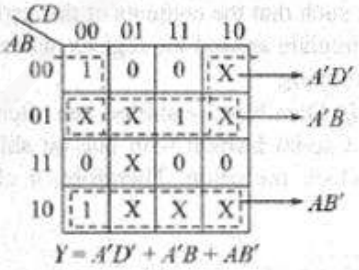
(a)



(b)

Counter A B C D	Sequence Generator Output, Y
0 0 0 0	1
1 0 0 0	1
1 1 0 0	0
1 1 1 0	0
1 1 1 1	0
0 1 1 1	1
0 0 1 1	0
0 0 0 1	0
0 0 0 0	...repeats...

(c)



(d)

Figure 9.27: (a) Solution with Method-1, (b)-(d) Solution with Method-2, (b) Targeted realization, (c) Counter sequence vs. sequence generator output, (d) Karnaugh Map to generate logic equation

Example 9.6

Show how modulo-8 switched tail counter works if is initialized with '1001'. How to decode this counter?

Solution:

Decoder output $Y = R'S'$

Clock	*Serial in = T'*	*Q*	*R*	*S*	*T*	*Y = R'S'*
0	0	1	0	0	1	1
1	1	0	1	0	0	0
2	1	1	0	1	0	0
3	0	1	1	0	1	0
4	1	0	1	1	0	0
5	0	1	0	1	1	0
6	0	0	1	0	1	0
7	1	0	0	1	0	0
8	0	1	0	0	1	1
9	1	0	1	0	0	0
						repeats

MODULE 4 (CONT.): COUNTERS

COUNTER

• This is a digital-circuit designed
→ to keep track of a number of events or
→ to count number of clock cycles (Figure 1.21).
• This can be constructed using
→ number of flip-flops (e.g. JK, SR, D and T) &
→ additional electronic circuits.
• This is similar to a register, since it is also capable of storing a binary number (e.g. 1011).
• The input to the counter is rectangular waveform labeled CLOCK.
• Each time the clock signal changes state from low to high (0->1), the counter will add one (1) to the number stored in its flip-flops. In other words, the counter will count the number of clock-transitions from low to high.
• Since the clock-pulses occur at known intervals, the counter can also be used as an instrument for measuring time and therefore period(T) or frequency(f).

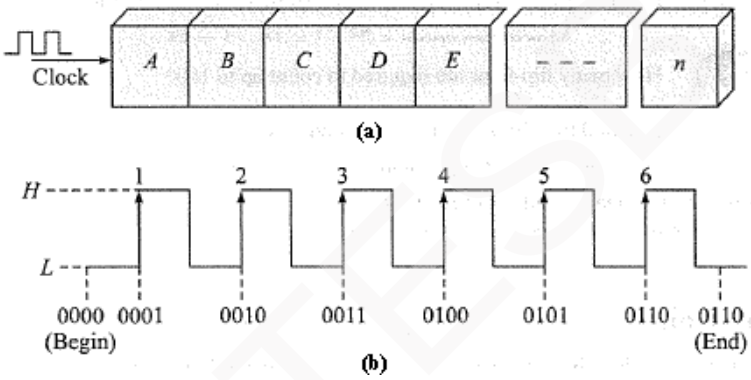


Figure 1.21: a)A counter constructed with 'n' flip flops for a count of 6

• There are 2 different types of counters:

1) Asynchronous Counter (Serial Counter)

➢ Each flip-flop is driven by the output of the previous flip-flop i.e. the output of a flip-flop is used as the clock-input for the next flip-flop and
➢ Advantage: It requires less hardware.
➢ Disadvantage: It operates at low speed.

2) Synchronous Counter (Parallel Counter)

➢ All flip-flops change states simultaneously since all clock inputs are driven by the same clock. i.e. all the flip-flops change states in synchronism.
➢ Advantage: It operates at high speed.
➢ Disadvantage: It requires more hardware.

• Serial and parallel counters are used in combination to compromise between speed of operation and hardware count.
• Counters can also be classified as:

1) Up Counter

➢ It can be used to count upward from a 0 to maximum-count
e.g. 0→1→2→3. . . . →n-2→n-1→n.
➢With each clock-transition, the content of the counter is increased by 1.

2) Down Counter:

➢ It can be used to count downward from a maximum-count to 0.
e.g. n→n-1→n-2. . . .→3→2→1.
➢With each clock-transition, the content of the counter is decreased by 1.

3) Up-down Counter:

➢ It is capable of counting in either an upward or a downward direction.

Properties	Three flip-flop counter	Four flip-flop counter
A counter having 'n' flip-flops will have 2^n output conditions (or states)	It has 2^3 = 8 output conditions (000 through 111).	It has 2^4 = 16 output conditions (0000 through 1111).
The largest binary number that can be represented by n flip-flops has a decimal equivalent of 2^n-1.	It has a maximum decimal number of 2^3-1=7.	It has a maximum decimal number of 2^4-1=15.
The **modulus** of a counter is the total number of states through which the counter can progress.	A three-flip-flop counter is referred to as a mod-8 counter. A mod-8 counter has 8 states.	A four-flip-flop counter is referred to as a mod-16 counter. A mod-16 counter has 16 states.

Example 10.1

How many flip-flops are required to construct a mod-128 counter? A mod-32? What is the largest decimal number that can be stored in a mod-64 counter?

Solution A mod-128 counter must have seven flip-flops, since $2^7 = 128$. Five flip-flops are needed to construct a mod-32 counter. The largest decimal number that can be stored in a six-flip-flop counter (mod-64) is 111111 = 63. Note carefully the difference between the modulus (total number of states) and the maximum decimal number.

ASYNCHRONOUS 3-BIT UP-COUNTER (RIPPLE-COUNTER)

• Each flip-flop is driven by the output of the previous flip-flop i.e. the output of a flip-flop is used as the clock-input for the next flip-flop (Figure 10.1).

• **Up-Counter:** It can be used to count upward from a 0 to maximum-count.
i.e. it counts in an upward direction (e.g. 0→1→2→3. . . . →n-2→n-1→n).
With each clock-transition, the content of the counter is increased by 1.

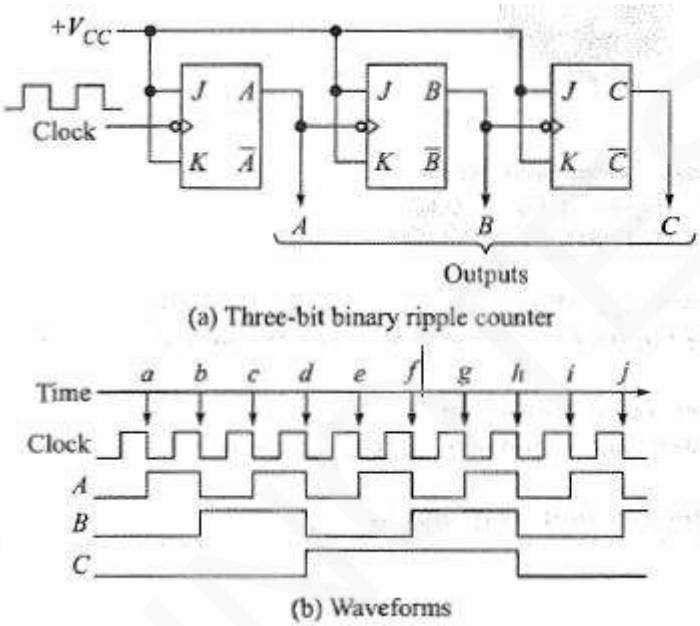
• A 3-bit counter can be used to count a maximum of 7 clock-transitions (Figure 10.1c).

• This can be constructed using clocked JK flip-flops.

• All J and K inputs of flip-flops are tied to $+V_{cc}$. This means, with each clock-transition, the flip-flop will change its state (toggle). (Toggle means 0→1 or 1→0)

• Here is how it works (Figure 10.1a):
- The counter outputs are A, B, and C.
- The system-clock is used to drive(or trigger) flip-flop A.
The output of flip-flop A is used to drive flip-flop B.
Likewise, the output of flip-flop B is used to drive flip-flop C.
- The triggers move through flip-flops like a ripple in water. Hence, it is called **ripple counter**.
- Flip-flop A toggles with each clock-transition.
Flip-flop B will toggle each time A goes LOW (1→0).
Likewise, flip-flop C will toggle each time B goes LOW (1→0).

• The overall propagation delay time is the sum of the individual delays.



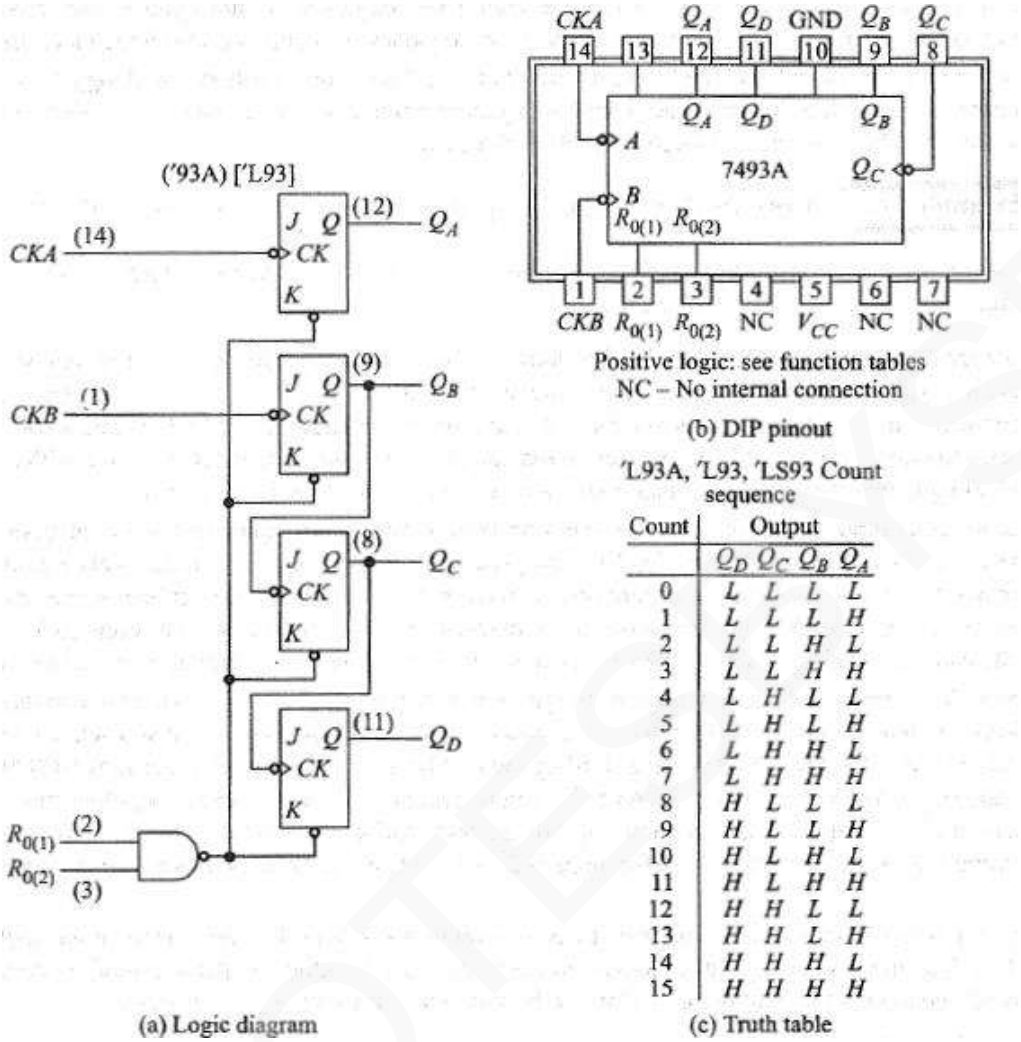
Negative clock transitions	C	B	A	State or count
---	0	0	0	0
a	0	0	1	1
b	0	1	0	2
c	0	1	1	3
d	1	0	0	4
e	1	0	1	5
f	1	1	0	6
g	1	1	1	7
h	0	0	0	0

(c) Truth table

Figure 10.1: a) 3-bit ripple counter b) waveforms c)truth table

7493A

• This TTL MSI circuit is a 4-bit binary counter that can be used in either a mod-8 or a mod-16 configuration.



Count	Output			
	Q_D	Q_C	Q_B	Q_A
0	L	L	L	L
1	L	L	L	H
2	L	L	H	L
3	L	L	H	H
4	L	H	L	L
5	L	H	L	H
6	L	H	H	L
7	L	H	H	H
8	H	L	L	L
9	H	L	L	H
10	H	L	H	L
11	H	L	H	H
12	H	H	L	L
13	H	H	L	H
14	H	H	H	L
15	H	H	H	H

Figure 10.2:7493A

• Here is how it works (Figure 10.2):

1) If the clock is applied at input CKB, the outputs will appear at Q_B, Q_C, and Q_D, and this is a mod-8 binary ripple counter. In this case, flip-flop Q_A is simply unused.

2) On the other hand, if the clock is applied at input CKA and flip-flop Q_A is connected to input CKB, we have a mod-16, 4-bit binary ripple counter. The outputs are Q_A, Q_B, Q_C, and Q_D.

• All the flip-flops in the 7493A have direct reset inputs that are active low.

• Thus, a high level at both reset inputs of the NAND gate, $R_{0(1)}$ and $R_{0(2)}$ is needed to reset all flip-flops simultaneously.

ASYNCHRONOUS 3-BIT DOWN-COUNTER

• Each flip-flop is driven by the output of the previous flip-flop i.e. the output of a flip-flop is used as the clock-input for the next flip-flop (Figure 10.4).

• **Down Counter:** It can be used to count downward from a maximum-count to 0
i.e. it counts in an downward direction (e.g. n→n-1→n-2. . . .→3→2→1).
With each clock-transition, the content of the counter is decreased by 1.

• A 3-bit counter can be used to count a maximum of 7 clock-transitions (Figure 10.4c).

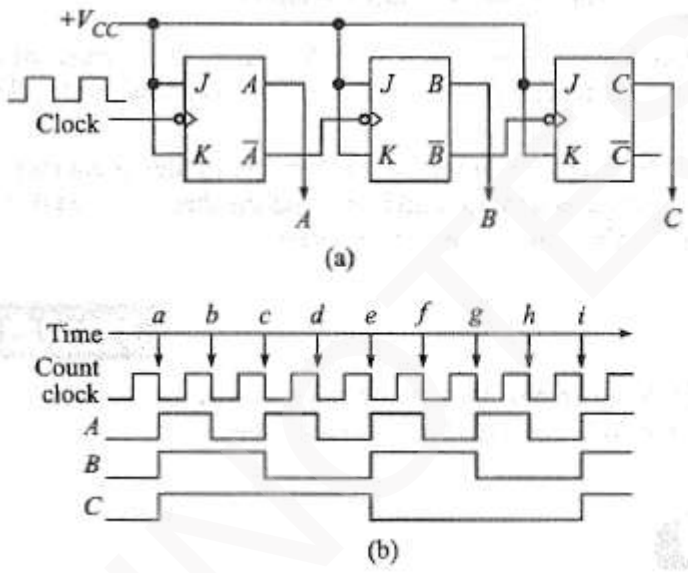
• This can be constructed using clocked JK flip-flops.

• All J and K inputs of flip-flops are tied to $+V_{cc}$. This means, with each clock-transition, the flip-flop will change its state (toggle). (Toggle means 0→1 or 1→0)

• Here is how it works (Figure 10.4a):

- The counter outputs are A, B, and C.
- The system-clock is used to drive(or trigger) flip-flop A.,
  But, the complement of A (i.e. A') is used to drive flip-flop B,
  Likewise B' is used to drive flip-flop C.
- The triggers move through flip-flops like a ripple in water. Hence, it is called **ripple counter**.
- Flip-flop A toggles with each clock-transition.
  But flip-flop B will toggle each time A goes HIGH (0→1).
  Likewise, flip-flop C will toggle each time B goes HIGH (0→1).

• The overall propagation delay time is the sum of the individual delays.



Count	C	B	A
7	1	1	1
6	1	1	0
5	1	0	1
4	1	0	0
3	0	1	1
2	0	1	0
1	0	0	1
0	0	0	0
7	1	1	1

(c)

Figure 10.4:a) 3 bit down counter b) waveforms c)truth table

ASYNCHRONOUS 3-BIT UP-DOWN-COUNTER

- **Up-down Counter:** It is capable of counting in either an upward or a downward direction.
- It is simply a combination of the up counter & down counter.
- A 3-bit counter can be used to count a maximum of 7 clock-transitions.
- This can be constructed using clocked JK flip-flops.
- All J and K inputs of flip-flops are tied to $+V_{cc}$. This means, with each clock-transition, the flip-flop will change its state (toggle). (Toggle means 0→1 or 1→0)
- Here is how it works (Figure 10.5a):

1) To operate in the Count-up Mode

- If Count-up = high and the Count-down = low, the lower AND gates Y_1 and Y_2 are disabled.
- The clock applied to flip-flop A will be steered into the other flip-flops by AND gates X_1 and X_2.

2) To operate in the Count-down Mode

- If Count-up = low and the Count-down =high, the upper AND gates X_1 and X_2 are disabled.
- The clock applied to flip-flop A will be steered into the other flip-flops by AND gates Y_1 and Y_2.

- Drawbacks: The additional gates (AND & OR) introduce additional delays.

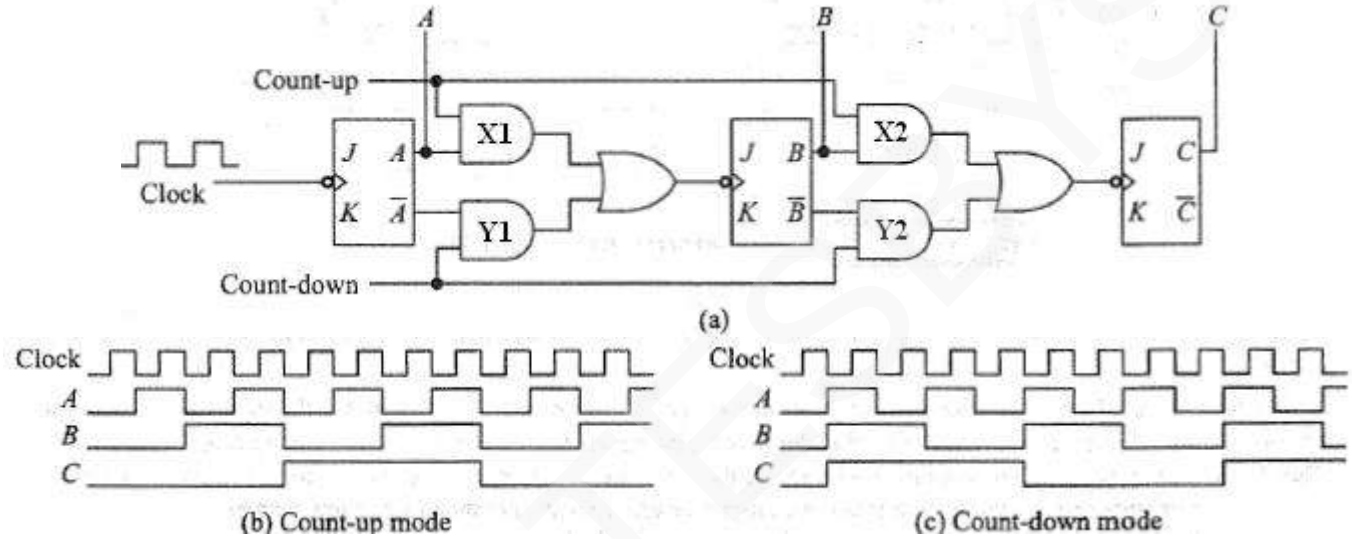


Figure 10.5: 3-bit binary up-down counter a) logic diagram b) Count up mode waveform c) Count-down mode waveform

Example 10.2

Draw the correct output waveforms for a 7493A connected as a mod-16 counter.

Solution The correct waveforms are shown in Fig. 10.3. The contents of the counter is 0000 at point *a* on the time line. With each negative clock transition, the counter is advanced by one until the counter contents are 1111 at point *b* on the time line. At point *c*, the counter resets to 0000, and the counting sequence repeats. Clearly, this is a mod-16 counter, since there are 16 states (0000 through 1111), and the maximum decimal number that can be stored in the flip-flops is decimal 15 (1111).

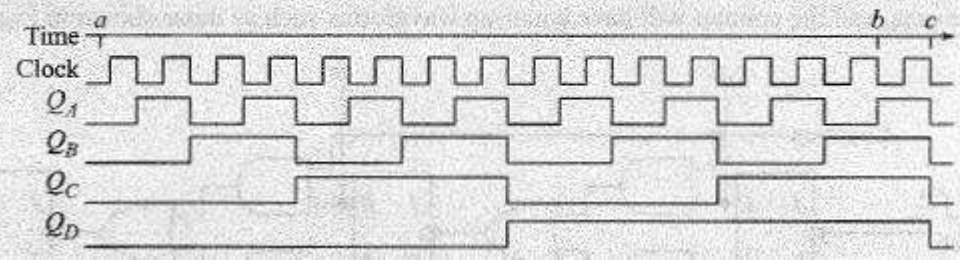


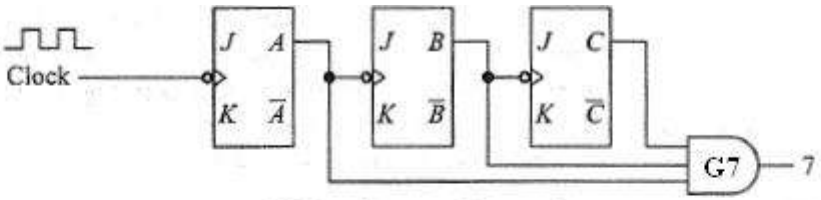
Figure 10.3: Waveforms for a mod 16 ,7493A

DECODING GATE

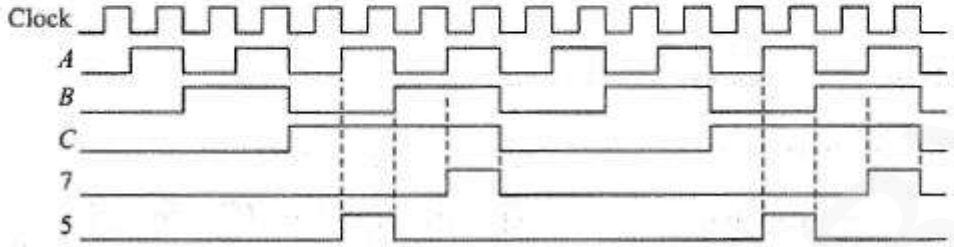
- It is a logic gate whose output is high(or low) only during one of the unique states of a counter.
- For example:

As shown in Figure 10.6a, the decoding-gate (G7) will decode state-7 (CBA=111).

Thus, the gate output will be HIGH only when A=1, B=1 and C=1.



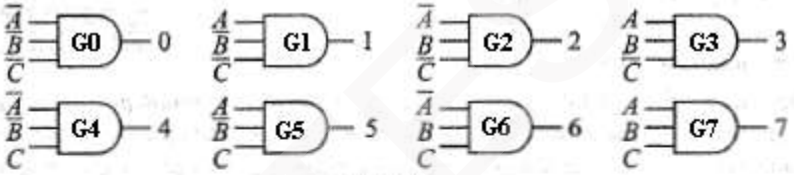
(a) Decoding gate for state 7



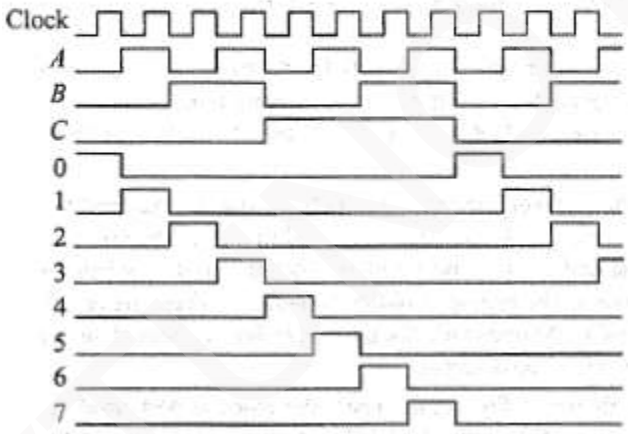
(b) Waveforms

(c) Gate to decode state 5

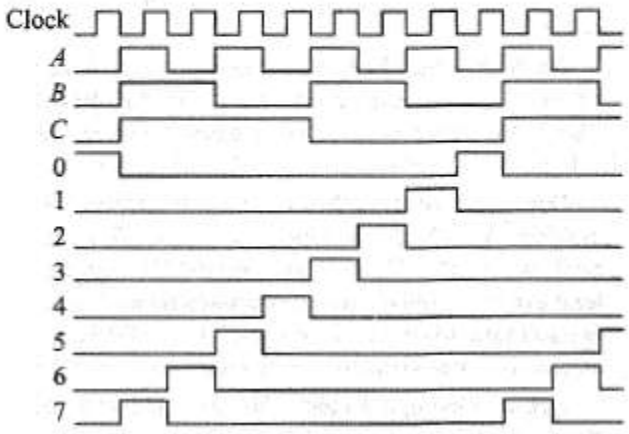
Figure 10.6: a)decoding gate for state-7 b)waveforms c)gate to decode state-5



(a) Gates



(b) Count-up mode



(c) Count-down mode

Figure 10.7: Decoding gates for a 3-bit binary ripple counter

DRAWBACKS IN ASYNCHRONOUS COUNTER

1) Low Speed: There is a limit to its highest operating frequency.

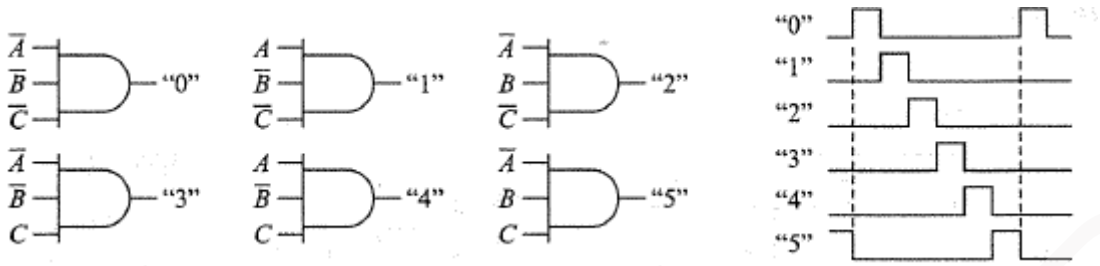
2) Long Delay: Each flip-flop has a delay time. These delay times are additive, and the total "settling" time for the counter is approximately the delay time times the total number of flip-flops.

3) Glitches: There is the possibility of glitches occurring at the output of decoding gates used with a counter.

Example 10.3

Draw decoding gates and all waveforms for the mod-6 counter

Solution:



SYNCHRONOUS 3 BIT UP-COUNTER (PARALLEL BINARY COUNTER)

• All flip-flops change states simultaneously since all clock inputs are driven by the same clock i.e. all the flip-flops change states in synchronism (Figure 10.10).

• **Up Counter:** It can be used to count upward from a 0 to maximum-count.
i.e. it counts in an upward direction (e.g. 0→1→2→3. . . . →n-2→n-1→n).
With each clock-transition, the content of the counter is increased by 1.

• A 3-bit counter can be used to count a maximum of 7 clock-transitions (Figure 10.10b).

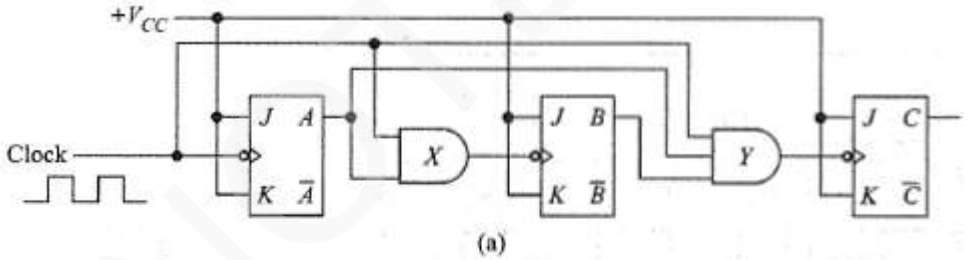
• This can be constructed using clocked JK flip-flops.

• All J and K inputs of flip-flops are tied to $+V_{cc}$. This means, with each clock-transition, the flip-flop will change its state (toggle). (Toggle means 0→1 or 1→0)

• This logic configuration is referred to as **steering logic**, since the clock-pulses are steered to each individual flip-flop.

• Here is how it works (Figure 10.10a):

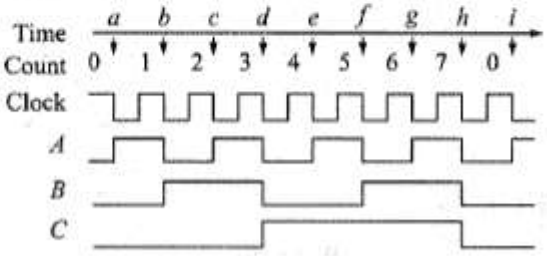
- The clock is applied directly to the flip-flop A. As result, with each clock-transition, flip-flop A will change its state (toggle).
- When A is high, AND gate-X is enabled and hence it transmits clock-pulse to flip-flop B.
- When A & B are high, AND gate-Y is enabled & hence it transmits clock-pulse to flip-flop C.



(a)

C	B	A	Count
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7
0	0	0	0

(b)



(c)

Figure 10.10: a) logic diagram b) truth table c) waveforms

SYNCHRONOUS 4-BIT UP-DOWN COUNTER

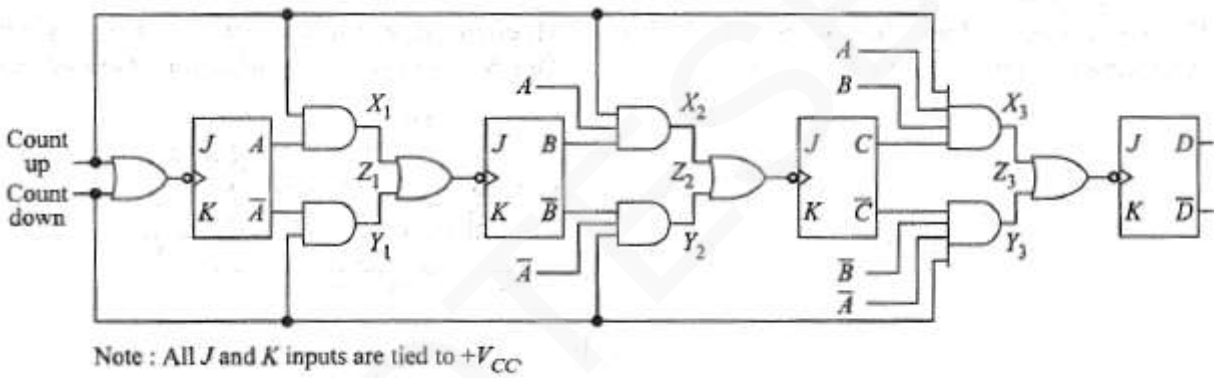
- **Up-down Counter**
  - ➢ It is capable of counting in either an upward or a downward direction (Figure 10.12c).
  - ➢ It is a combination of
    - → up counter &
    - → down counter.
- A 4-bit counter can be used to count a maximum of 15 clock-transitions.
- This can be constructed using clocked JK flip-flops.
- All J and K inputs of flip-flops are tied to $+V_{cc}$. This means, with each clock-transition, the flip-flop will change its state (toggle). (Toggle means 0→1 or 1→0)
- Here is how it works (Figure: 10.12a):

1) To operate in the Count-up Mode

- ➢ The system-clock is applied at the count-up input, while the count-down input is held low
- ➢ If the count-down = LOW, the lower AND gates Y_1, Y_2 and Y_3 are disabled.
- ➢ The clock applied at count-up will then go directly into flip-flop A.
- ➢ Then, the clock will be steered into the other flip-flops by AND gates X_1, X_2 and X_3.

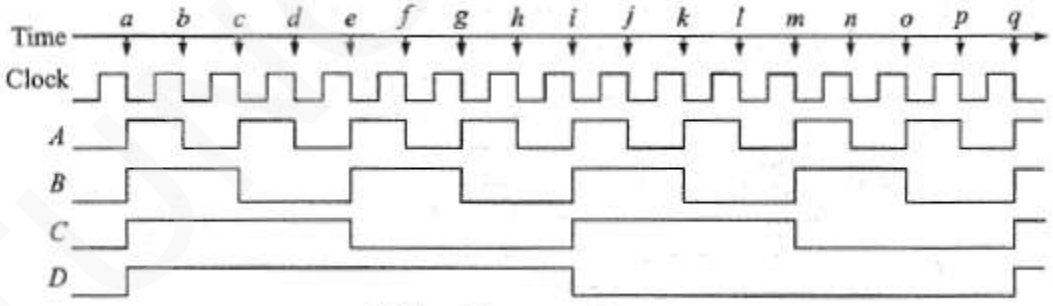
2) To operate in the Count-down Mode

- ➢ The system-clock is applied at the count-down input while holding the count-up input low.
- ➢ If the count-up = LOW, the upper AND gates X_1, X_2 and X_3 are disabled.
- ➢ The clock applied at count-down will then go directly into flip-flop A.
- ➢ Then, the clock will be steered into the other flip-flops by AND gates Y_1, Y_2 and Y_3.



Note : All J and K inputs are tied to $+V_{CC}$

(a) Logic diagram



(c) Count down waveforms

Figure 10.12:a)logic diagram b)count up waveforms c)count down waveforms

Asynchronous Counters	Synchronous Counters
Flip flops are connected in such a way that the output of the first flip-flop drives the second flip-flop, the output of the second flip-flop drives the third flip-flop etc.	There is no connection between output of first flip-flop and clock input of the next flip-flop.
All flip-flops are not clocked simultaneously, since all the clock inputs are not connected to the system-clock. Only the first flip-flop is driven by the clock.	All flip-flops are clocked simultaneously by connecting the clocks inputs of all flip-flops in parallel to the system-clock.
Logic circuit is quite simple, whatever the number of states.	As the number of states increases, the design of the logic circuit tends to become complex.
Main drawback is their low speed as the clock is propagated through number of flip-flops before it reaches last flip-flop.	Since all the flip-flops have the system-clock applied simultaneously, propagation delay in minimum, it being equal to the propagation delay of one flip-flop plus propagation delay of one AND gate.

Example 10.4
What is the clock frequency in Fig. 10.1 if the period of the waveform at C is 24 μs?

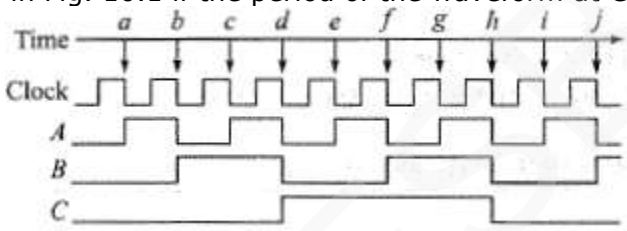


Figure 10.1: 3-bit counter

Solution Since there are eight clock cycles in one cycle of C, the period of the clock must be $24/8 = 3$ μs. The clock frequency must then be $1/(3 \times 10^{-6}) = 333$ kHz.

Example 10.5
A 4-bit binary asynchronous counter is connected with a clock of 500 KHz frequency. Find the time period of the waveform at the output of the first & the last JK flip flop.
Solution:
Time period of the waveform at the output of the first JK Flip Flop=4μs
Time period of the waveform at the output of the last JK Flip Flop=32μs

CHANGING THE COUNTER MODULUS

COUNTER MODULUS

- **Modulus** is defined as the number of states through which a counter can progress.
- **Natural counter:** The counters which
  - → progress 1 count at a time in a strict binary fashion, and
  - → has a modulus given by 2^n, where n=number of flip-flops.
- For example,
  - → mod-2 counter consists of 1 flip-flop, and it counts two discrete states (0->1).
  - → mod-4 counter consists of 2 FFs, and it counts through 4 discrete states (00->01->10->11).
  - → mod8 counter consists of 3 FFs, and it counts through 8 discrete states.
- It is often desirable to construct counters having a modulus other than 2, 4, 8 and so on. For example, a counter having a modulus of 3 or 7 would be useful.
- A small modulus counter can always be constructed from a larger modulus counter by skipping states. Such counters are called a **modified-counter**.
- Firstly, it is necessary to determine the number of flip-flops required for modified counter.
  - ➢ The correct number of flip-flops is determined choosing the lowest natural count that is greater than the desired modified count.
  - ➢ For example, a mod-7 counter requires 3 flip-flops, since 8 is the lowest natural count greater than the desired modified count of 7.

Example 10.6

Indicate how many flip-flops are required to construct each of the following counters: (a) mod-3, (b) mod-6, and (c) mod-9.

Solution

a. The lowest natural count greater than 3 is 4. Two flip-flops provide a natural count of 4. Therefore, it requires at least two flip-flops to construct a mod-3 counter.

b. Construction of a mod-6 counter requires at least three flip-flops, since 8 is the lowest natural count greater than 6.

c. A mod-9 counter requires at least four flip-flops, since 16 is the lowest natural count greater than 9.

Example 10.7

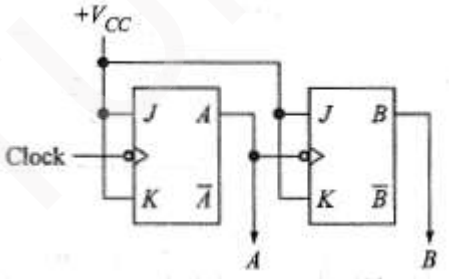
What modulus counters can be constructed with the use of four flip-flops?

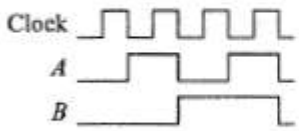
Solution A four-flip-flop counter has a natural count of 16. We can thus construct any counter that has a modulus between 16 and 2, inclusive. We might choose to use four flip-flops only for counters having a modulus between 16 and 9, since only three flip-flops are required for a modulus of less than 8, and only two are required for a modulus of less than 4.

Example 10.8

Draw the logic diagram, truth table, and waveforms for a two-flip-flop ripple counter

Solution:





Clock	B	A
0	0	0
1	0	1
2	1	0
3	1	1
0	0	0

MOD-3 COUNTER

• A mod-3 counter can count in sequence from 00 to 11(00->01->10)
• Mod-3 counter
→ requires 2 flip-flops which has a natural count of 4
→ skips one state (i.e. 11) {Figure: 10.16c}.
• Here's how it works (Figure: 10.16b):
1) Prior to point 'a' on the time-line, A=0 and B=0. A clock-transition at 'a' will cause:
i. 'A' to toggle to a 1, since it's inputs J=HIGH & K=HIGH.
ii. 'B' to reset to 0, since it's inputs J=LOW & K=HIGH.
2) Prior to point 'b' on the time-line, A=1 and B=0. A clock-transition at 'b' will cause:
i. 'A' to toggle to a 0, since it's inputs J=HIGH & K=HIGH.
ii. 'B' to toggle to a 1, since it's inputs J=HIGH & K=HIGH.
3) Prior to point 'c' on the time-line, A=0 and B=1. A clock-transition at 'c' will cause:
i. 'A' to reset to 0, since it's inputs J=LOW & K=HIGH.
ii. 'B' to reset to 0 since it's inputs J=LOW and K=HIGH.
4) The counter has now progressed through all three of its states, advancing one count with each clock-transition.
• It can be considered as a divide-by-3 block, since the output waveform at B has a period equal to three times that of the clock. In other words, this counter divides the clock frequency by 3.
• This is a synchronous counter since both flip-flops change state in synchronism with the clock.

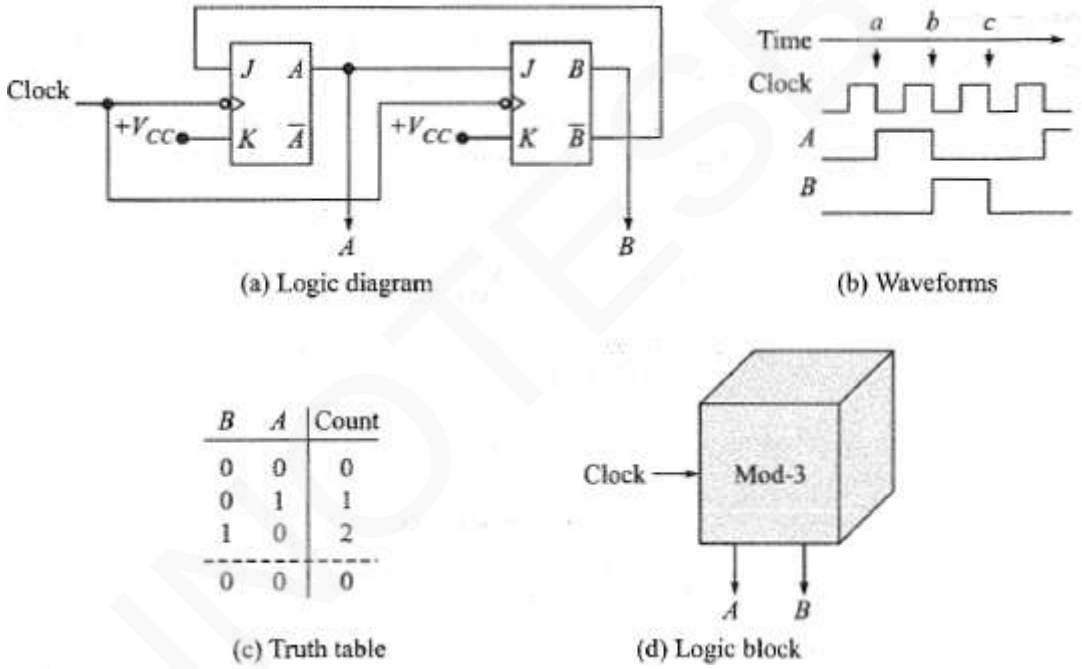


Figure 10.16: a)logic diagram b)waveforms c)truth table d)logic block

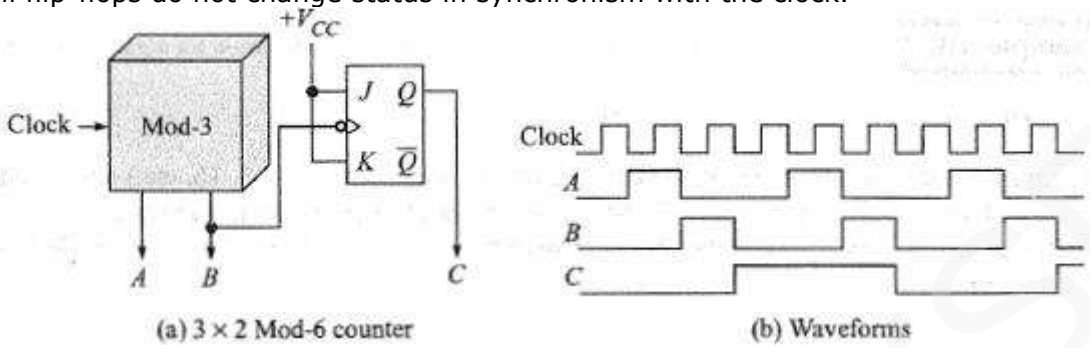
NOTE:

• A higher-modulus counters can be formed by using the product of any number of lower-modulus counters.
• For example,
1) mod-6 counter can be formed by using the mod-3 counter in conjunction with a flip-flop
2) mod-10 counter can be formed by using the mod-5 counter in conjunction with a flip-flop

MOD-6 COUNTER

• This counter can be constructed by using
→ mod-3 counter &
→ one flip-flop (Figure:10.7).

• This logic circuit can no longer be considered a synchronous counter since flip-flop C is driven by flip-flop B i.e. all flip-flops do not change status in synchronism with the clock.



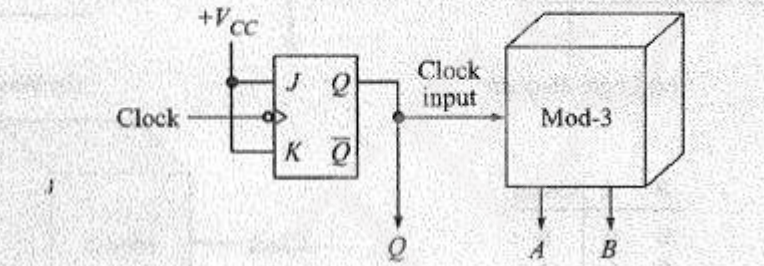
(a) 3 × 2 Mod-6 counter (b) Waveforms

Figure 10.17: a)3*2 mod-6 counter b)waveforms

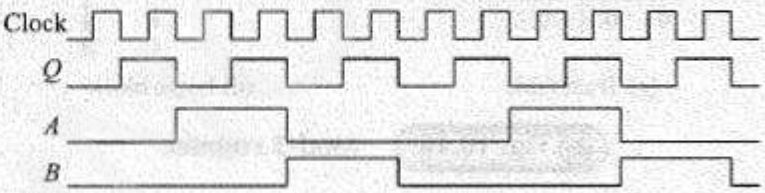
Example 10.9

Draw the waveforms you would expect from the mod-6 counter by connecting a single flip-flop in front of the mod-3 counter in Fig. 10.16.

Solution The resulting counter is a 2 × 3 = mod-6 counter that has the waveforms shown in Fig. 10.18. Notice that *B* now has a period equal to six clock periods, but it is not symmetrical.



(a) 2 × 3 Mod-6 counter



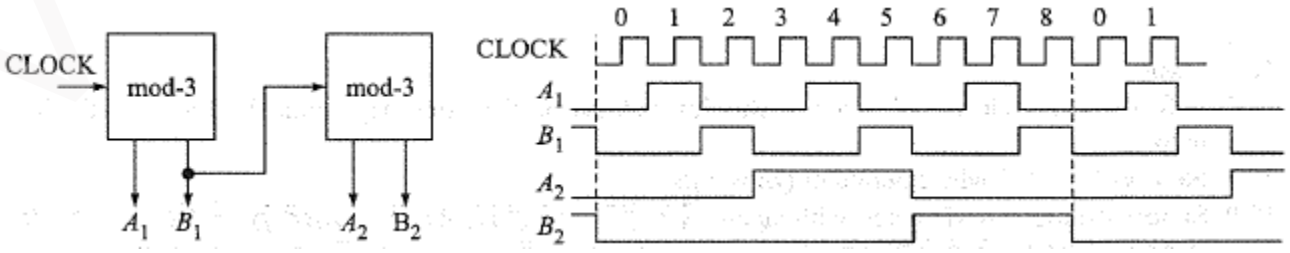
(b) Waveforms

Figure 10.18

Example 10.10

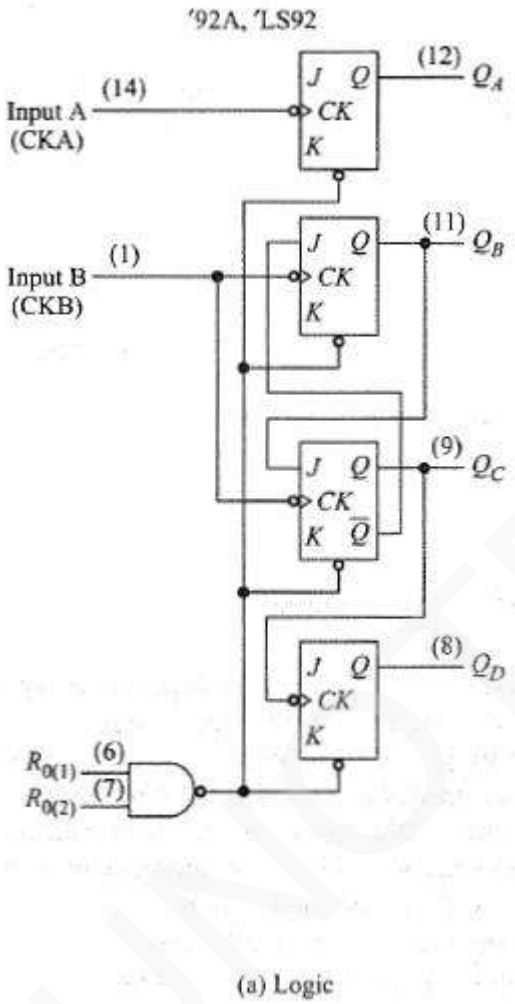
Draw the logic diagram, truth table, and waveforms for a mod-9 counter using two mod-3 counters connected in series.

Solution:



7492A

- The 7492A is a TTL divide-by-12, MSI counter.
- Here's how it works (Figure 10.19):
  - If the clock is applied to input B of the 7492A and the outputs are taken at Q_B, Q_C, and Q_D, this is a mod-6 counter.
  - If the clock is applied at input A and Q_A is connected to input B, we have a 2x3x2 mod-12 counter.
- This logic circuit can be considered a asynchronous counter since all flip-flops do not change states at the same time.



Count	Output Q_D	Q_C	Q_B	Q_A
0	L	L	L	L
1	L	L	L	H
2	L	L	H	L
3	L	L	H	H
4	L	H	L	L
5	L	H	L	H
6	H	L	L	L
7	H	L	L	H
8	H	L	H	L
9	H	L	H	H
10	H	H	L	L
11	H	H	L	H

(b) Truth table

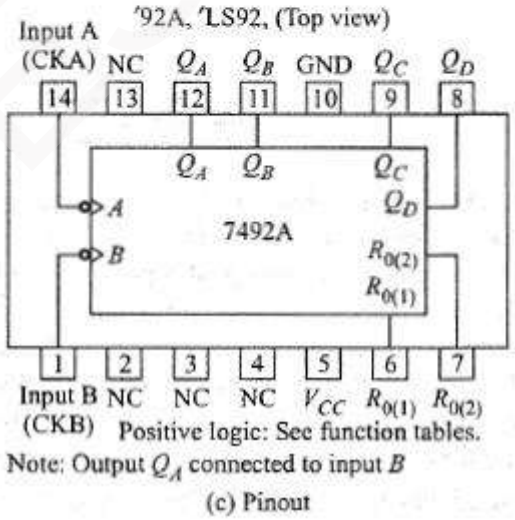


Figure 10.19: 7492A

MOD-5 COUNTER

- This counter has a natural count of 8, but it will skip over 3 states i.e. 101,110 & 111.
- While constructing a mod-5 counter, the omitted states need to be examined to ensure that the counter get into invalid states 101, 110, or 111 (Figure 10.20a).
- Here's how it works (Figure 10.20c):
  - The counter can be in 3 invalid states: 1) CBA=101 2) CBA=110 3) CBA=111
  - We consider first 2 invalid states (Figure 10.20e).

Case 1: Invalid State CBA=101.

- Assuming that the counter is in valid state 5 (CBA=101).
- On the next clock-transition, the following events occur:
  1) Since C' is low, flip-flop A is reset. Thus A changes from a 1 to a 0.
  2) When A goes from a 1 to a 0, flip-flop B triggers and B changes from a 0 to a 1
  3) Since J input to flip-flop C is low, flip-flop C is reset and C changes from a 1 to a 0.
  4) Thus, the counter progresses from the illegal state 5 to the legal state 2(CBA=010).

Case 2: Invalid State CBA=110.

- Assume that the counter is in state 6 (CBA=110).
- On the next clock-transition, the following events occur:
  1) Since C' is low, flip-flop A is reset. Since A is already a 0, it just remains a 0.
  2) Since A does not change, flip-flop B does not change and B remains a 1.
  3) Since J input to flip-flop C is low, flip-flop C is reset and C changes from a 1 to a 0.
  4) Thus, the counter progresses from the illegal state 6 to the legal state 2 (CBA=010).

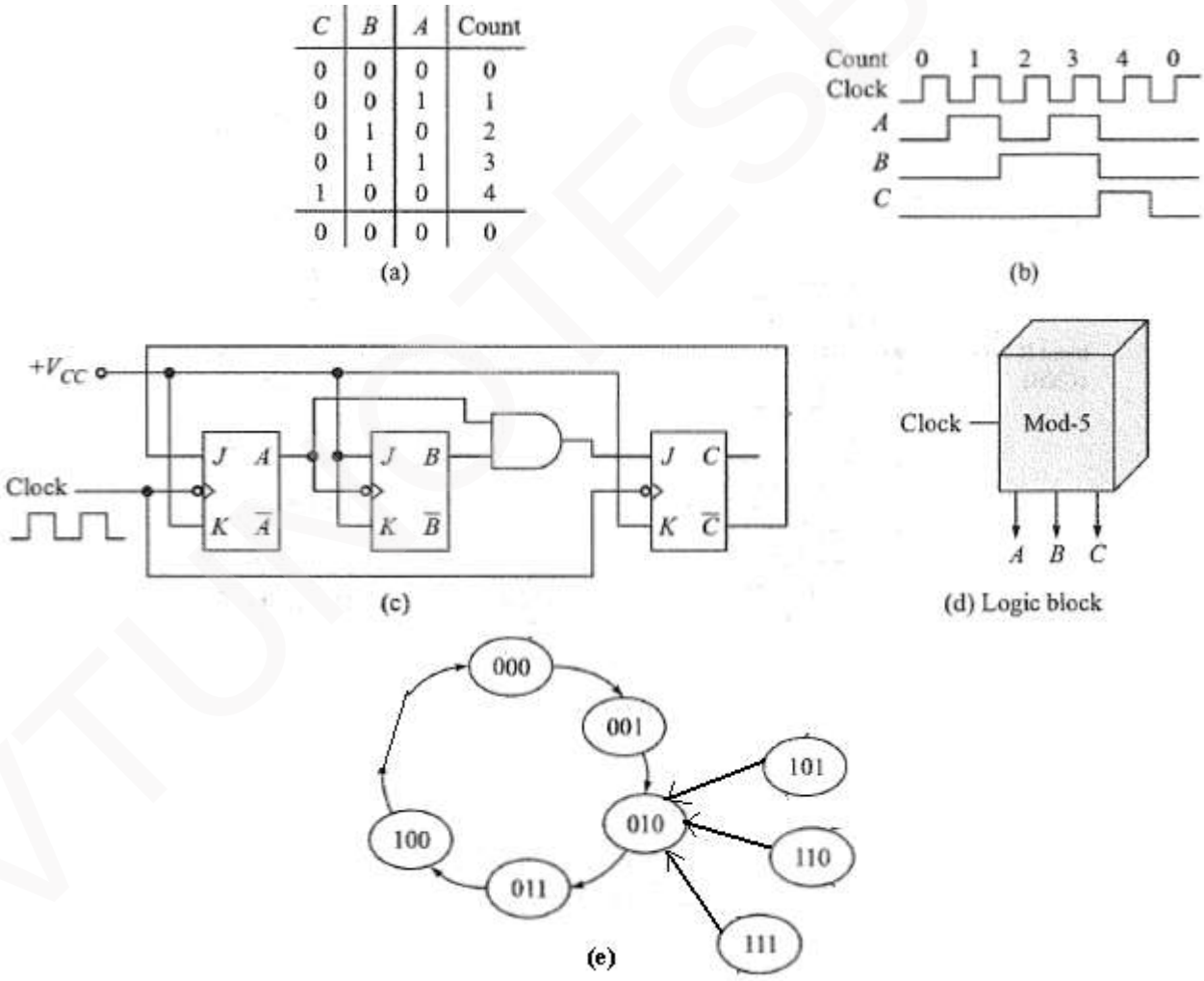


Figure 10.20:a)truth table b)waveforms c)logic diagram d)logic block e)invalid states 101,110 & 111 to valid state 010

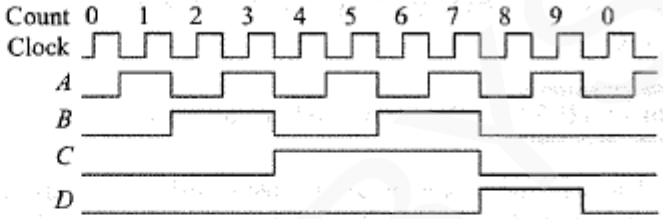
MODULE 5: COUNTERS (CONT.)

DECADE COUNTER (MOD-10 COUNTER)

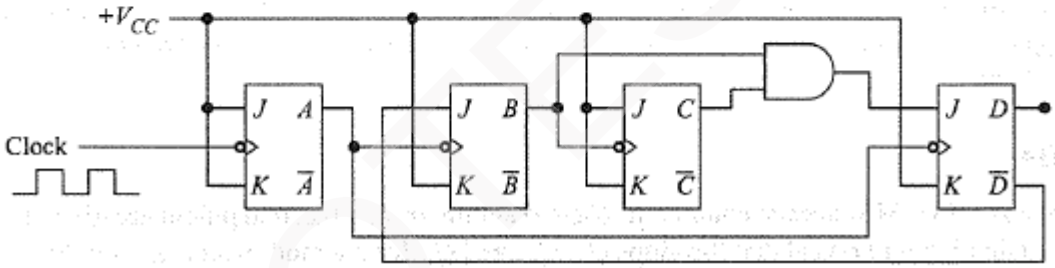
- This counter can be constructed by using
  - → mod-5 counter &
  - → one flip-flop (Figure:10.21).
- For example, a 2x5 or a 5x2 will form a mod-10 counter (or decade counter).

D	C	B	A	Count
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
0	0	0	0	0

(a)



(b)



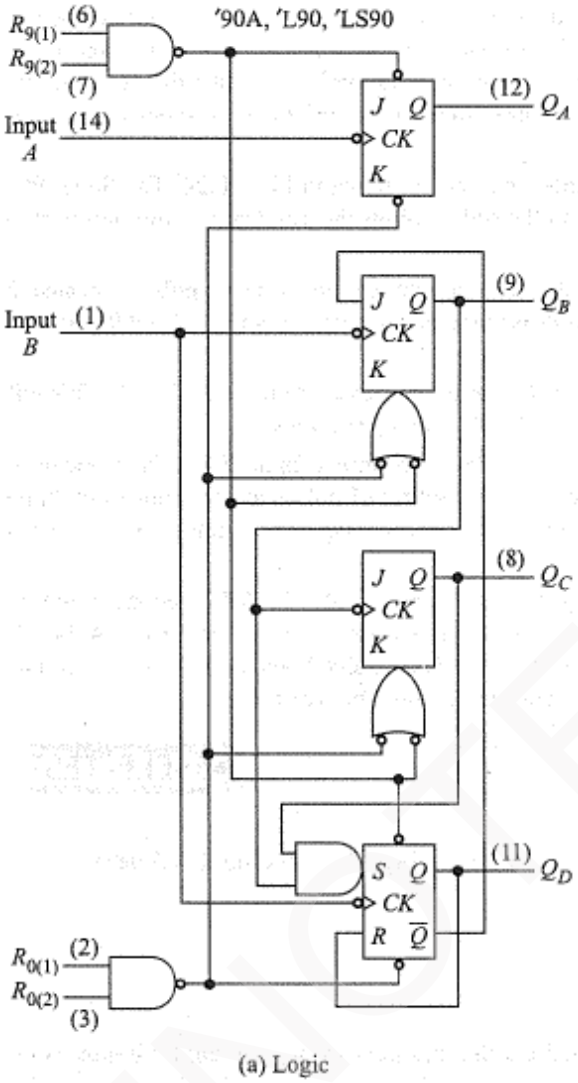
(c)

Figure 10.21:a)logic diagram b)truth table c)waveforms

7490A

- 7490A is a TTL decade counter.
- Flip-flops Q_B, Q_C, and Q_D form a mod-5 counter (Figure 10.22).
- Here's how it works:

1) If system-clock is applied at input A & Q_A is connected to input-B, we have **binary decade counter**
2) If system-clock is applied at input B & Q_D is connected to input-A, we have **biquinary counter**.



(a) Logic

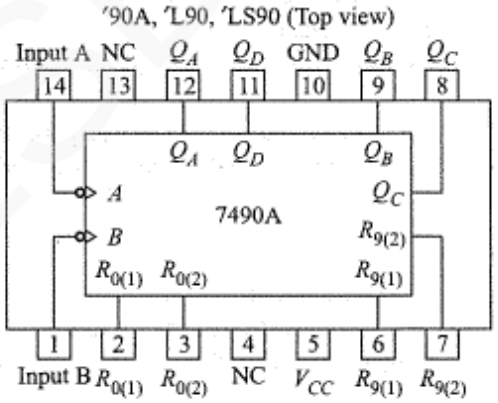
'90A, 'L90, 'LS90
BCD count sequence
(See note A)

Count	Output Q_D	Q_C	Q_B	Q_A
0	L	L	L	L
1	L	L	L	H
2	L	L	H	L
3	L	L	H	H
4	L	H	L	L
5	L	H	L	H
6	L	H	H	L
7	L	H	H	H
8	H	L	L	L
9	H	L	L	H

'90A, 'L90, 'LS90
Bi-quinary (5-2)
(See note B)

Count	Output Q_A	Q_D	Q_C	Q_B
0	L	L	L	L
1	L	L	L	H
2	L	L	H	L
3	L	L	H	H
4	L	H	L	L
5	H	L	L	L
6	H	L	L	H
7	H	L	H	L
8	H	L	H	H
9	H	H	L	L

(b) Truth table



Positive logic: See function tables

Note: A output Q_A connected to input B
B output Q_D connected to input A

(c) Pinout

Figure 10.22: 7490A

- Application of 7490A: Cascaded 7490's can count from 0 to 999.
  - As shown in Figure 10.23, the three '90A counters are connected in series.
  - Rightmost '90A counts no. of input pulses at its input. This block is called a **units-counter**.
  - The middle '90A will advance one count each time the units-counter progresses from count 9 to 0. This will occur once for every 10 input pulses. This block is called the **tens-counter**.
  - Leftmost '90A will advance one count each time the tens-counter progresses from count 9 to 0. This will occur once for every 100 input pulses. This block is called the **hundreds-counter**.
  - This is widely used in digital voltmeters, frequency counters.

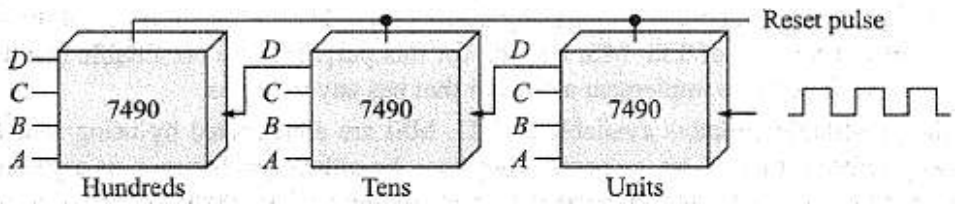


Figure 10.23:Cascaded 7490's can count to 999

PRESETTABLE COUNTER

- The presettable counter can be used to implement a counter that has any modulus (Figure: 10.24).
- Nearly, all presettable-counters are built by using 4 flip-flops, .'. They are called as **4-bit counters**.
- For example,
  - Both 74161 and 74163 are synchronous binary counters that operate in a count-up mode.
  - Both 74191 and 74193 are synchronous binary counters, but they can operate in either a count-down or count-up mode.
- Since the decade counter is a very useful configuration, many 4-bit counters are internally connected to provide a modified count of 10.

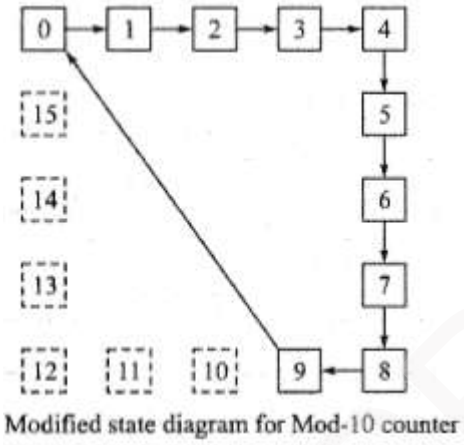


Figure 10.24: Presettable counter

PRESETTABLE SYNCHRONOUS 4-BIT COUNTER (74163)

- **Q_A, Q_B, Q_C,** and **Q_D** are outputs of four flip-flops (Figure 10.25).
- **CARRY** output can be used to enable successive counter stages (e.g. in a units, tens application).
- The two **ENABLE** inputs are used to control the counter.
    1) If either ENABLE input is low, the counter stops to count;
    2) If both ENABLE input are high, the counter starts to the count.
- **CLEAR'** input is used to reset all flip-flop.

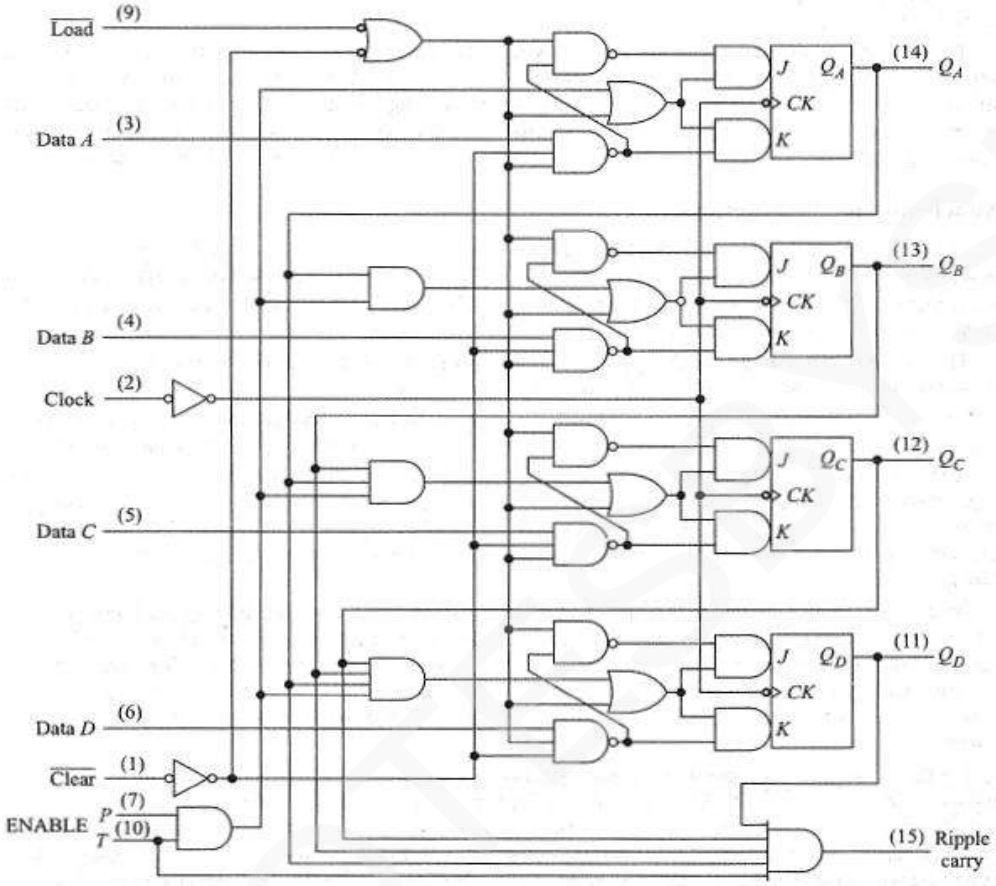
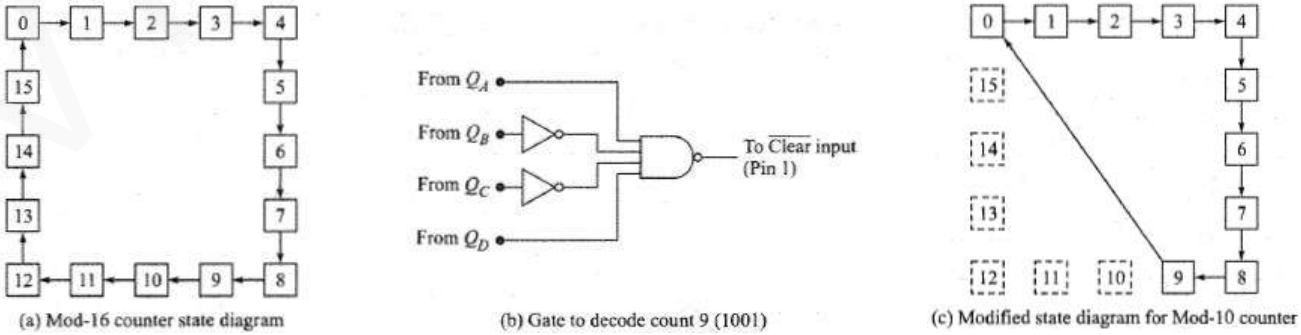


Figure 10.25: 74163

- **LOAD'** input is used to load the values present on the four data inputs (D, C, B, and A) into four flip-flops. This is a very useful function when it is desired to have the counter begin counting from a predetermined count.
- The count-length can be modified by making use of the CLEAR' input.
    - Use a NAND gate to decode the maximum count desired (Figure 10.26).
    - Use the output of this NAND gate to clear the counter to count 0000.
    - Then, counter will count from 0000 to maximum desired count & then clear back to 0000.
- For ex, if max count = 9, we connect inputs of NAND gate to decode count 9=DCBA =1001. We then have a mod-10 counter, since the count sequence is from 0000 to 1001. (Fig 10.26b & 10.26c)



(a) Mod-16 counter state diagram (b) Gate to decode count 9 (1001) (c) Modified state diagram for Mod-10 counter

Figure 10.26: a)mod-16 counter state diagram b)modified state diagram for mod-10 counter

PRESETTABLE SYNCHRONOUS UP-DOWN COUNTERS

- 74193 is a 4-bit synchronous up-down binary counter (Figure 10.29).
- Pin **PL'** is a control input for loading data into pins **P_A, P_B, P_C,** and **P_D**.
  When the device is used as a counter, these four pins are left open and PL' must be held high.
- Pin **MR** is the master reset, and it is normally held low. (A high level on MR will reset all flip-flops.)
- Outputs **TC_U** and **TC_D** are to be used to drive the following units, such as in a cascade arrangement.
- The clock inputs are **CP_U** and **CP_D**.
  - Placing the clock on CP_U will cause the counter to count up.
  - Placing the clock on CP_D will cause the counter to count down.
- **Q_A, Q_B, Q_C** and **Q_D** are the outputs of the counter.

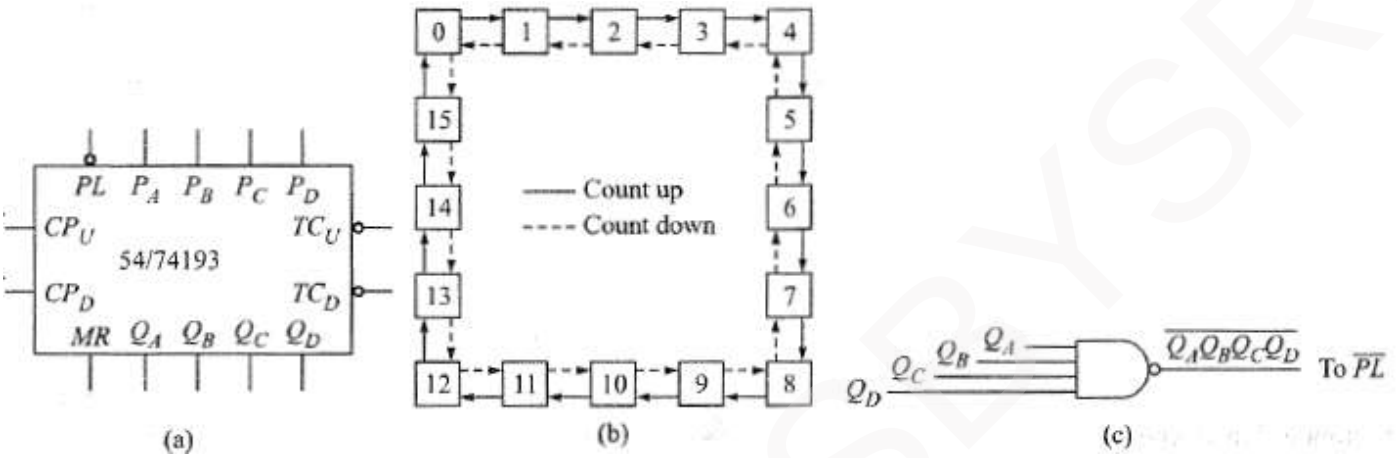


Figure 10.29: 4-bit binary counter (presettable)

- 74193 can be reset to any desired count with the parallel load inputs.
  - The counter can be preset to the number present on parallel-data-entry inputs P_B, P_C, & P_D.
  - When PL'=low, the data present at the four inputs is shifted into the counter; that is, the counter is preset to the number held by $P_DP_CP_BP_A$.
- How to modify the count?
  - Use a NAND gate to detect any of the stable states, say, state 15(1111), and use this gate output to make PL' low. (Figure 10.29c)
  - The only time PL' will be low is when Q_D, Q_C Q_B and Q_A are all high, or state 15(1111).
  - At this time, the counter will be preset to the data $P_DP_CP_BP_A$.
  - For example (Figure 10.30),
    - Suppose that $P_DP_CP_BP_A$=1001 (the number 9).
    - When the clock is applied, the counter will progress naturally to count 15(1111).
    - At this time, PL' will go low and the number 9 (1001) will be shifted into the counter.
    - The counter will then progress through states 9, 10, 11, 12, 13, and 14, and at count 15 it will again be preset to 9.

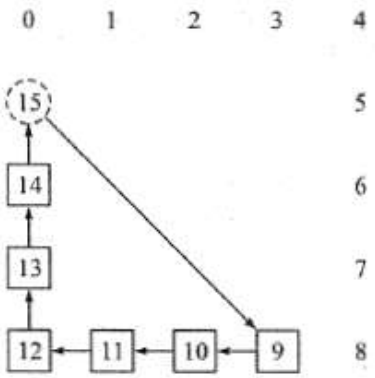


Figure 10.30

COUNTER DESIGN AS A SYNTHESIS PROBLEM

Example 10.11

Design a rnodulo-6 counter as described in Figure:10.32.

Solution:

- In mod-6 counter, there are 6 states. So, we need 3 flip-flops to design mod-6 counter.
- Let the three JK flip-flops be FF-A, FF-B and FF-C
- With 3 flip-flops, 8 different states are possible but states 110 & 111 are not used.

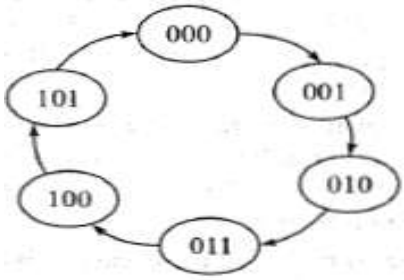


Figure 10.32: State sequence of mod 6 counter

$Q_n \rightarrow Q_{n+1}$		S	R	J	K	D	T
0	0	0	×	0	×	0	0
0	1	1	0	1	×	1	1
1	0	0	1	×	1	0	1
1	1	×	0	×	0	1	0

Table 8.35: Excitation Table of flip-flops

Table 10.1: State Table for Design of mod-6 counter

C_n	B_n	A_n	C_{n+1}	B_{n+1}	A_{n+1}	J_C	K_C	J_B	K_B	J_A	K_A
0	0	0	0	0	1	0	×	0	×	1	×
0	0	1	0	1	0	0	×	1	×	×	1
0	1	0	0	1	1	0	×	×	0	1	×
0	1	1	1	0	0	1	×	×	1	×	1
1	0	0	1	0	1	×	0	0	×	1	×
1	0	1	0	0	1	×	1	0	×	×	1

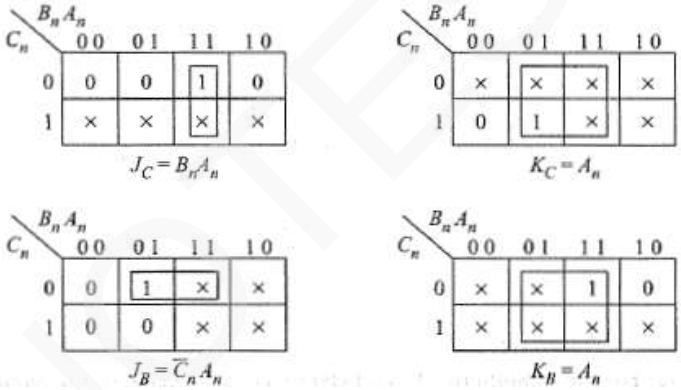


Figure 10.33: Derivation of design equations from kmap.

- From the kmaps, the J, K inputs to the flip-flops A, B and C are obtained as:

$J_A=1$	$K_A= 1$	for flip-flops A
$J_B=C'_n A_n$	$K_B=A_n$	for flip-flops B
$J_C=B_nA_n$	$K_C= A_n$	for flip-flops C

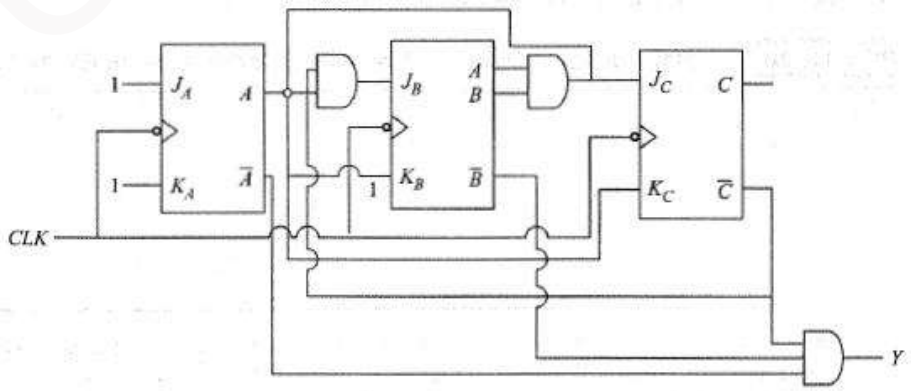


Figure 10.34: Circuit diagram of mod6 synchronous counter

SELF-CORRECTING COUNTER

• **Lock out of a counter** means counter getting locked into unused states (Figure: 10.35a).
Solution: Use self-correcting counter (Figure: 10.35b).

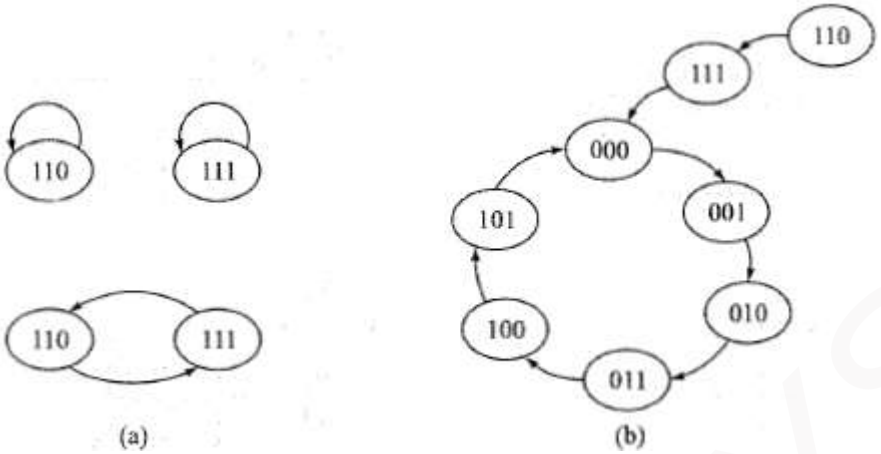


Figure 10.35: (a) Lock-in conditions, (b) Full state transition diagram for circuit in Fig. 10.34

• As shown in Fig.10.35b, unused states 110 and 111 can no longer be considered as don't care.
• This type of design is called **self-correcting** as the circuit comes out on its own from an invalid state to a valid state.

Note:
• **Present state** ($C_n\ B_n\ A_n$) means state of the flip-flops before the occurrence of a clock-pulse.
Next state ($C_{n+1}\ B_{n+1}\ A_{n+1}$) means state of the flip-flops after the occurrence of the clock-pulse.

Example 10.12

Design a self-correcting rnodulo-6 counter in which all the unused state leads to state CBA = 000.

Solution:

Table 10.2: State Table for design of self-correcting mod-6 counter

C_n	B_n	A_n	C_{n+1}	B_{n+1}	A_{n+1}	J_C	K_C	J_B	K_B	J_A	K_A
0	0	0	0	0	1	0	×	0	×	1	×
0	0	1	0	1	0	0	×	1	×	×	1
0	1	0	0	1	1	0	×	×	0	1	×
0	1	1	1	0	0	1	×	×	1	×	1
1	0	0	1	0	1	×	0	0	×	1	×
1	0	1	0	0	1	×	1	0	×	×	1
1	1	0	0	0	0	×	1	×	1	0	×
1	1	1	0	0	0	×	1	×	1	×	1

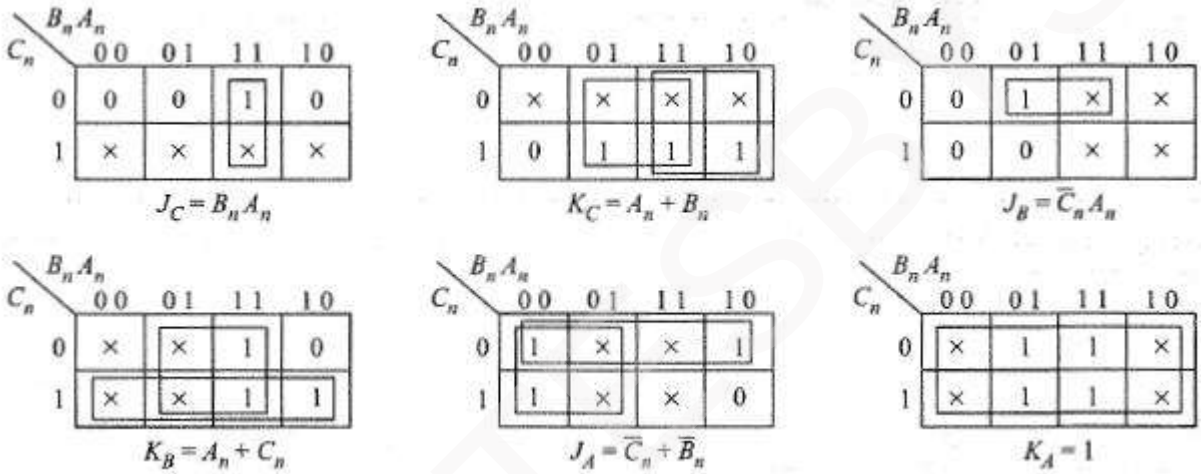


Figure 10.36: Design equations

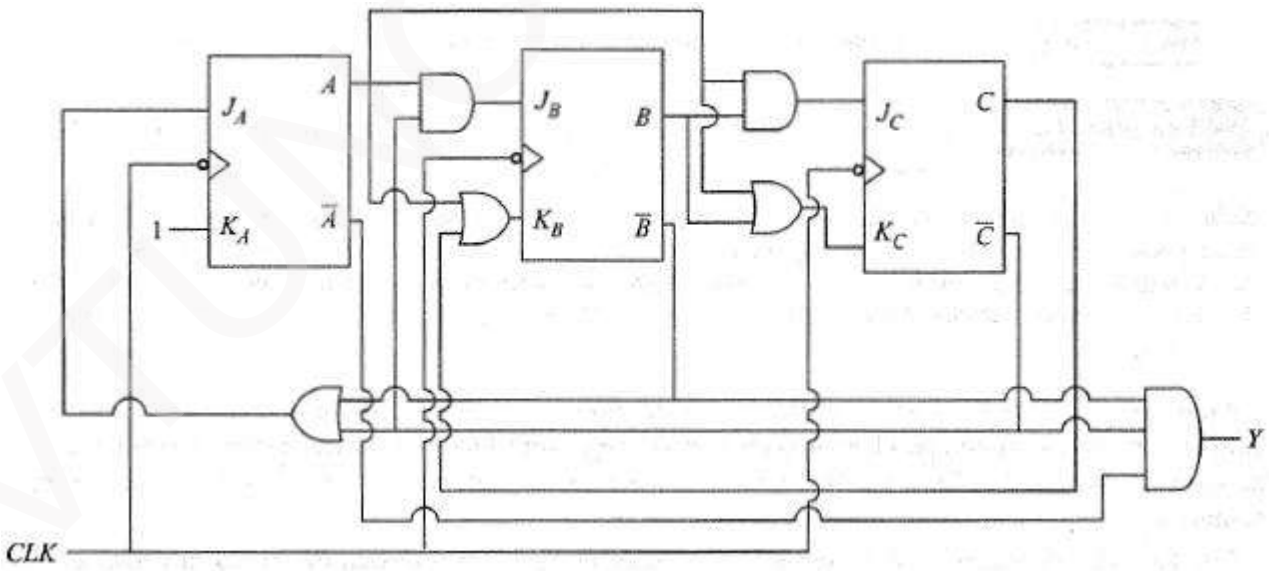
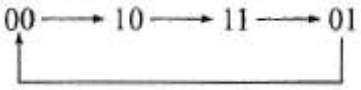


Figure 10.37: Circuit diagram

Example 10.13

Design a modulo-4 irregular counter with following counting sequence using D flip-flop.



Solution Using state excitation table of *D* flip-flop (Fig. 8.34), the state table can be formed as shown in Table 10.2.

Table 10.2: State Table for Design of Irregular Counter

B_n	A_n	B_{n+1}	A_{n+1}	D_B	D_A
0	0	1	0	1	0
0	1	0	0	0	0
1	0	1	1	1	1
1	1	0	1	0	1

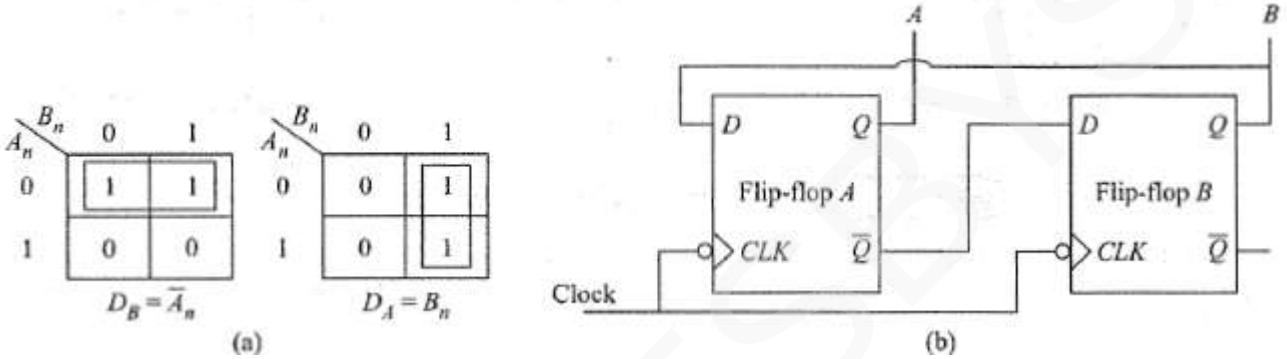


Figure 10.38: (a) Deriving design equations, (b) Circuit diagram

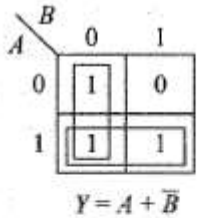
Example 10.14

Show how a modulo-4 counter designed with two flip-flops can generate a repetitive sequence of binary word '1101' with minimum number of memory elements?

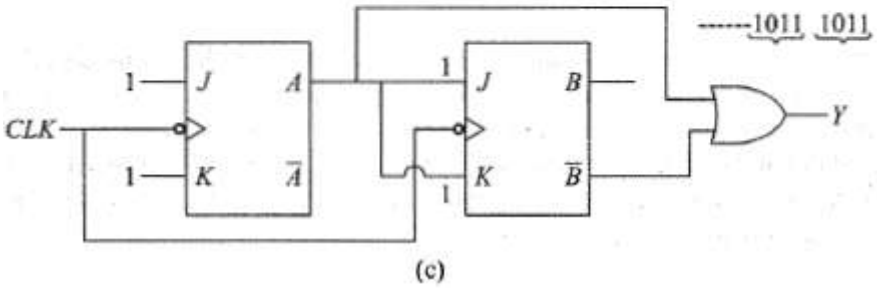
Solution Let the counting sequence of two flip-flops *B* and *A* be 00 → 01 → 10 → 11 → 00 ..., i.e. a modulo-4 synchronous up counter. The corresponding output is 1 → 1 → 0 → 1 → 1.... As shown in Fig. 10.39(a) the sequence '1101' will be generated repetitively by *Y*. Figure 10.39(b) gives Karnaugh Map representation of *Y* and we get $Y = A + B'$. A standard modulo-4 up counter and an 2-input OR gate connected as shown in Fig. 10.39(c) generates the given sequence.

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

(a)



(b)



(c)

Figure 10.39: Sequence generator circuit using synchronous counter, (a) State Table, (b) Output equation, (c) Circuit diagram

Example 10.15

Design a self correcting modulo-3 down counter.

Solution We need 2 flip-flops, say B and A for this purpose which has 4 states. Let the down counter count like $10 \rightarrow 01 \rightarrow 00 \rightarrow 10\ldots$ and undesired state 11 corrects itself to 10. The excitation table of Fig. 8.35 is used for the design purpose.

In Method-1, we use *SR* flip-flop for design purpose. Figure 10.45a shows the state table and in the second column, necessary inputs for the two *SR* flip-flops are given. Figure 10.45b shows the use of Karnaugh Map to get the design equations.

In Method-2, we use *JK* flip-flop for design purpose. The fourth column shows necessary inputs for the two *JK* flip-flops. Figure 10.45c shows the use of Karnaugh Map to get the design equations.

In Method-3, we use *D* flip-flop for design purpose. The third column shows necessary inputs for the two *D* flip-flops. Fig. 10.45d shows the use of Karnaugh Map to get the design equations.

In Method-4, we use *T* flip-flop for design purpose. The last column shows necessary inputs for the two *D* flip-flops. Figure 10.45e shows the use of Karnaugh Map to get the design equations.

Present State B_nA_n	Next State $B_{n+1}A_{n+1}$	S_BR_B	S_AR_A	D_B	D_A	J_BK_B	J_AK_A	T_B	T_A
0 0	1 0	1 0	0 X	1	0	1 X	0 X	1	0
0 1	0 0	0 X	0 1	0	0	0 X	X 1	0	1
1 0	0 1	0 1	1 0	0	1	X 1	1 X	1	1
1 1	1 0	X 0	0 1	1	0	X 0	X 1	0	1

(a)

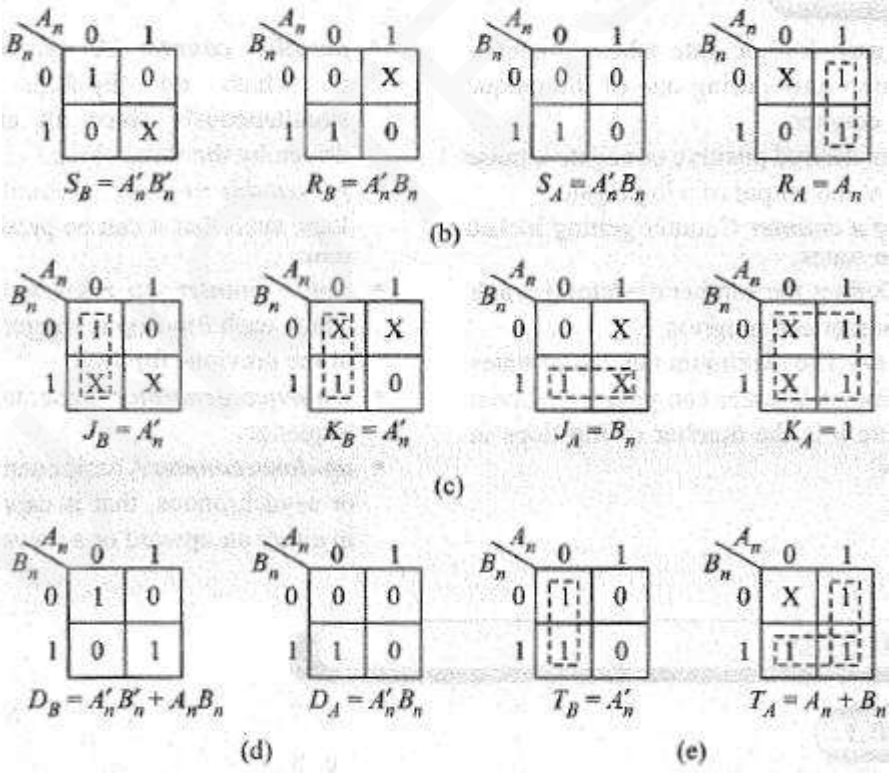


Figure 10.45: (a) State table for the self correcting modulo~3 counter and required inputs, (b) Design with SR flip-flops, (c) Design with JK flip-flops, (d)Design with D flip-flops, (e) Design with T flip-flops

DIGITAL CLOCK

• As shown in figure 10.40, the first divide-by-60 counter divides the 60Hz power signal down to a 1Hz square wave. The second divide-by-60 counter changes state once each second and has 60 discrete states. It can be decoded to provide signals to display seconds. This counter is referred to as the seconds counter.

• The third divide-by-60 counter changes state once each minute and has 60 discrete states. This counter is referred to as the minutes counter.

• The last counter changes state once each 60 minutes. Thus, if it is a divide-by-12 counter, it will have 12 states that can be decoded to provide signals to display the correct hour. This is referred to as the hours counter.

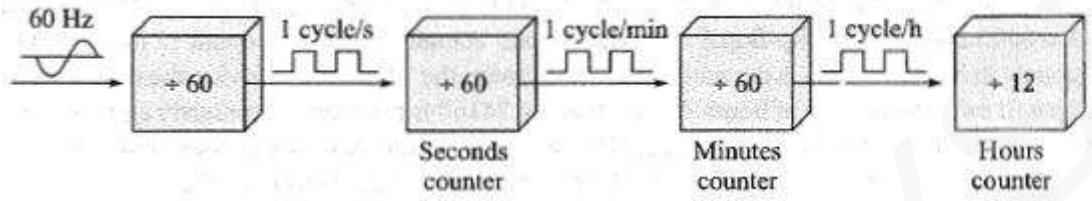


Figure 10.40: Block diagram of digital clock

DIVIDE-BY-60 COUNTER

• The divide-by-60 counter can be implemented by cascading counters(10*6=60)

• 7490 decade counter can be used as a divide-by-10 counter and the 7492 can be used as a divide-by-6 counter. Cascading these two counters will provide a divide-by-60 counter (Figure: 10.41).

• The amplifier at the input provides a 60Hz square wave of the proper amplitude to drive the 7490.

• In figure:10.42, the 7492 is connected as a divide-by-12 counter, but only outputs Q_A,Q_B & Q_C are used. In this fashion, the 7492 operates essentially as a divide-by-6 counter.

• For decoding 'seconds counter' to represent each of the 60s in 1min, we construct a mod-10 counter in series with a mod-6 counter.

• The mod-10 counter can then be decoded to represent the units digit of seconds, and the mod-6 counter can be decoded to represent the tens digits of seconds.

• Since both the 7490 & 7492 count in the straight 8421, a 7447 decoder-driver can be used with each to drive two 7-segment indicators.

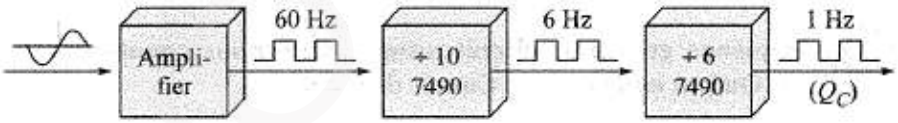


Figure 10.41:Divide-by-60 counter

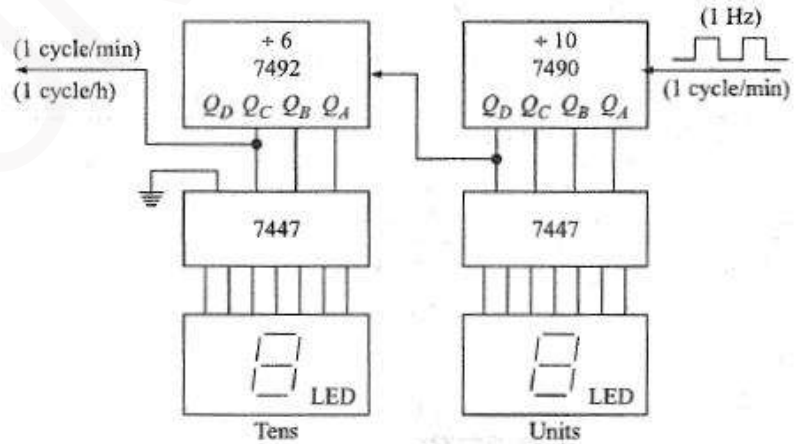


Figure 10.42: A 10*6 mod-60 counter with units and tens decoding

Write a verilog code for a modulo-8 up counter

```
module UC(Clock, Reset,Q);
input Clock, Reset;
output  [2:0] Q;
  //modulo 8 requires 3 flip-flop
reg [2:0] Q;
always @ (negedge Clock or negedge Reset)
  if (~Reset) Q=3'b0;
  else Q = Q+1;
endmodule
```

Write a verilog code for a modulo-8 up counter using JK flip-flop

```
module UCJK(A,B,Clock,Reset);
input Clock,Reset;
output  A,B; //modulo-3 requires 2 flip-flop
wire JA,JB,KA,KB;
assign JA=~B;

assign KA=1'b1;
assign JB=A;
assign KB=1'b1;
JKFF JK1(A,JA,KA,Clock,Reset); //instantiates JKFF
JKFF JK2(B,JB,KB,Clock,Reset); //instantiates JKFF
endmodule

module JKFF(Q,J,K,Clock,Reset);
input J,K,Clock,Reset;
 output Q;
 reg Q;
always @ (negedge Clock or negedge Reset)
   if(~Reset) Q=1'b0;
   else Q <=  (J&~Q) | (~K&Q);
endmodule
```

Write a verilog code for a modulo-8 up down counter which counts in upward direction if input MODE=0, else counts in downward direction. It has a parallel load facility. When PL=1,a 3-bit number D is asynchronously loaded to the counter. The counter counts at the negative edge of CLOCK and its output is represented by Q.

```
module UDCPL(CLOCK,PL,MODE,D,Q);    //Up Down Counter
input CLOCK,PL,MODE;                //with parallel load
input [2:0] D;
output  [2:0] Q; //modulo 8 requires 3 flip-flop
reg [2:0] Q;
integer updown; //updown will be +1 or -1 depending on MODE
always @ (negedge CLOCK)
    begin
      if (MODE) updown=-1; //If MODE=1, counts downward
      else updown=1;
      if (PL) Q=D;  //If PL=1, parallel loading takes place
      else Q=Q+updown;  //Last else statement responds to clock
    end
Endmodule
```

MODULE 5 (CONT.): D/A CONVERSION AND A/D CONVERSION

INTRODUCTION

- In general, there are 2 types of electrical signals: 1) Analog signal & 2) Digital signal.
- Analog-signal is a signal whose amplitude can take any value between given limits.
  Digital-signal is a signal whose amplitude can have only given discrete values b/w defined limits.
  i.e. Analog → continuous
  Digital → discrete (step by step).
- The linear electronics circuit (like linear amplifier, transducers) generates analog signals.
  Whereas, the computers generate digital signals.

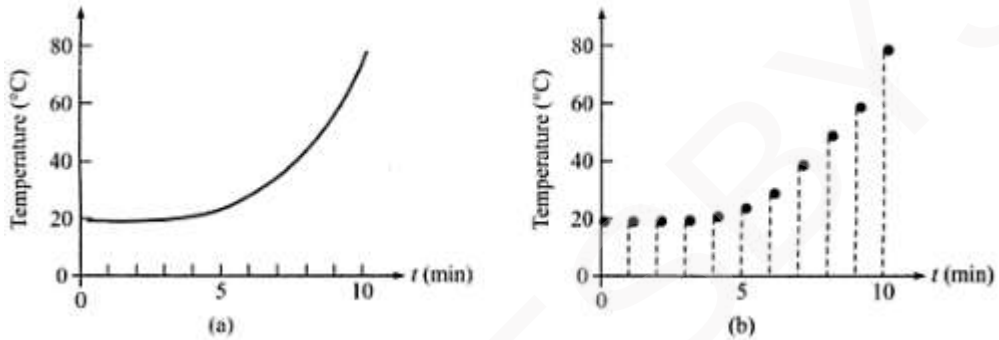


Figure 1.1: (a) analog signal (b) digital signal

NEED FOR A/D & D/A CONVERSION

- Whenever, we need to interface (connect) analog and digital circuits, we need to convert either analog signal into proportional digital signal or vice versa. Therefore, we need the A/D & D/A converters. (A/D → Analog-to-Digital D/A → Digital-to-Analog)

A/D Conversion	D/A Conversion
Process of converting analog signal into equivalent digital signal is called **A/D conversion**.	Process of converting digital signal into equivalent analog signal is called **D/A conversion**.
The electronics circuit, which does this process, is called **A/D converter (ADC)**.	The electronics circuit, which does this process, is called **D/C converter (DAC)**.
The circuit has only one analog input & 'n' number of digital data outputs.	The circuit has 'n' number of digital data inputs & only one analog output.
ADC is often referred to as an encoding device.	DAC is often referred to as a decoding device.
For ex: ADC can be used to change the analog output signals from transducers (measuring temperature/pressure) into equivalent digital signals.	
Analog-to-digital conversion is a complex process.	Digital-to-analog conversion is a straightforward process and is considerably easier than A/D conversion
Popular ADCs: 1) Simultaneous ADC 2) Counter type ADC 3) Continuous ADC 4) Successive Approximation ADC 5) Section counter ADC 6) Single RAMP ADC 7) Dual-Slope ADC	Popular DACs: 1) Resistive Divider DAC 2) Binary ladder DAC (or R–2R ladder)

DIGITAL TO ANALOG CONVERTER (DAC)

- The process of converting digital signal into equivalent analog signal is called **D/A conversion**.
- The electronics circuit, which does this process, is called **D/A converter (DAC)**.
- The circuit has 'n' number of digital data inputs with only one output.
- Popular DACs are: 1) Weighted resistors DAC and 2) Binary ladder(or R–2R ladder DAC).

BASICS OF DAC

- A digital signal has to be converted into an equivalent analog signal.
- Here, the basic problem is to change the 'n' digital voltage levels into one equivalent analog voltage.
- This can be most accomplished by designing a resistive-network.
- The resistive-network will change each digital level into an equivalent binary weighted voltage (or current).

Binary Equivalent Weight

- The value assigned to each bit in a digital number, expressed as a fraction of the total.
- The values are assigned in binary fashion according to sequence 1,2,4,8, ... ,2^n (n = total no. of bits).
- Consider the truth table for the 3-bit binary signal shown in Table 12.1.

2^2	2^1	2^0
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Table 12.1

- Assume we want to convert a 3-bit digital input into an analog output.
- The smallest number represented is 000; let us make this equal to 0V.
  The largest number is 111; let us make this equal to +7V.
- Between 000 and 111, there are 7 discrete levels to be defined.
  Therefore, the analog signal can be divided into 7 levels.
- The smallest incremental change in digital signal is represented by LSB=$2^{0.}$
- In general, the binary equivalent weight assigned to the LSB is given by

$$\text{LSB weight} = \frac{1}{(2^n - 1)}$$

where n = number of bits.

- The remaining weights are found by multiplying by 2, 4, 8, and so on.

RESISTIVE DIVIDER DAC

- Resistive divider is a circuit of 2 or more resistors with a voltage source.
  The algebraic sum of voltage drops across each resistor is equal to voltage source.
- This can be used to convert a digital input into an analog output.
- Assume we want to convert a three-bit digital input into an analog output (Fig 12.3).
  Also, assume the digital input levels are 0 = 0V and 1 = +7V (Table 12.1).
- The resistive divider must do 2 things:
  1) The 2^0 bit must be changed to +1V.
  2^1 bit must be changed to +2V.
  2^2 bit must be changed to +4V.
  2) These 3 voltages must be summed together to form the analog output voltage.

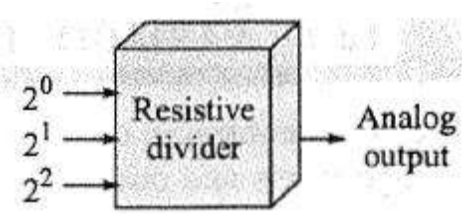


Figure 12.3: Resistive Divider DAC

Digital input	Analog output
0 0 0	+0 V
0 0 1	+1 V
0 1 0	+2 V
0 1 1	+3 V
1 0 0	+4 V
1 0 1	+5 V
1 1 0	+6 V
1 1 1	+7 V

Table 12.2

- Resistors R_0, R_1 & R_2 forms the divider network. Resistor R_L represents the load to which divider is connected.

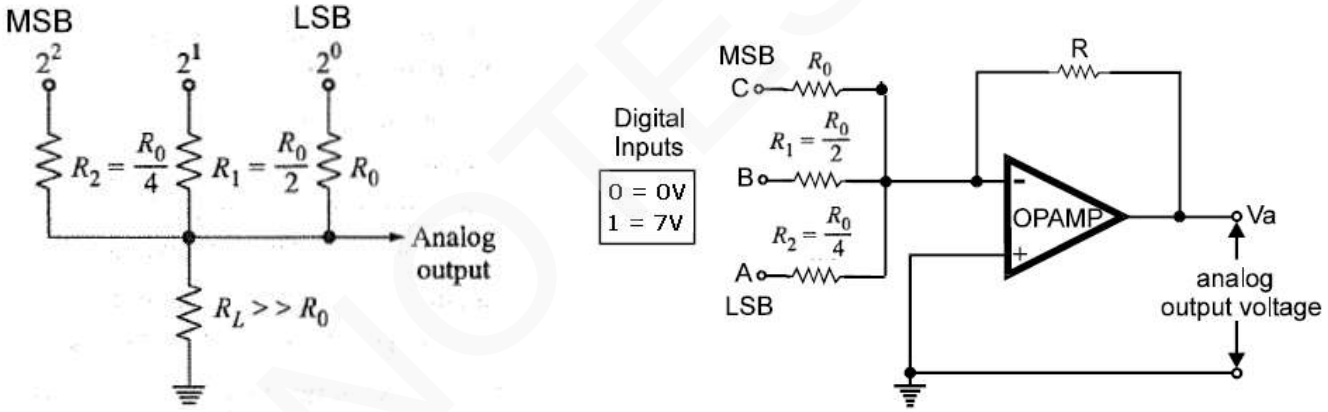


Figure 12.4: Resistive Divider DAC

- The following criteria can be applied to the divider:
  1) There must be one input resistor for each digital bit.
  2) Beginning with LSB, each following resistor value is one-half the size of the previous resistor.
  3) The full-scale output voltage is equal to the positive voltage of the digital input signal.
  4) The LSB has a weight of $1/(2^n-1)$, where n = number of input bits.
  5) The change in output voltage due to a change in the LSB is equal to $V/(2^n-1)$,
  where V = digital input voltage level.
  6) The output voltage V_A can be found by using Millman's theorem:

$$V_A = \frac{V_0 2^0 + V_1 2^1 + V_2 2^2 + V_3 2^3 + \cdots + V_{n-1} 2^{n-1}}{2^n - 1}$$

- **Drawbacks:**
  1) Each resistor in the network has different value. So, error in value of each resistor adds up.
  2) The value of resistor at MSB is the lowest. Hence, it draws more current. Therefore, the resistor used for the MSB is required to handle a much larger current than the LSB resistor.
- **Solution:** Binary ladder DAC overcomes this problem.

Example 12.1

Find the binary equivalent weight of each bit in a 4-bit system.

Solution The LSB has a weight of $1/(2^4 - 1) = 1/(16 - 1) = \frac{1}{15}$, or 1 part in 15. The second LSB has a weight of $2 \times \frac{1}{15} = \frac{2}{15}$. The third LSB has a weight of $4 \times \frac{1}{15} = \frac{4}{15}$, and the MSB has a weight of $8 \times \frac{1}{15} = \frac{8}{15}$. As a check, the sum of the weights must equal 1. Thus $\frac{1}{15} + \frac{2}{15} + \frac{4}{15} + \frac{8}{15} = \frac{15}{15} = 1$. The binary equivalent weights for 3-bit and 4-bit systems are summarized in Fig. 12.2.

Bit	Weight
2^0	1/7
2^1	2/7
2^2	4/7
Sum	7/7

(a)

Bit	Weight
2^0	1/15
2^1	2/15
2^2	4/15
2^3	8/15
Sum	15/15

(b)

Figure 12.2 Binary equivalent weights

Example 12.2

For a 5-bit resistive divider, determine the following:

(a) The weight assigned to the LSB.
(b) The weight assigned to the second and third LSB.
(c) The change in output voltage due to a change in the LSB, the second LSB, & the third LSB.
(d) The output voltage for a digital input of 10101.

Assume O =0V and 1 = +10V.

Solution

(a) The LSB weight is $1/(2^5 - 1) = 1/31$.
(b) The second LSB weight is 2/31, and the third LSB weight is 4/31.
(c) The LSB causes a change in the output voltage of 10/31 V. The second LSB causes an output voltage change of 20/31 V, and the third LSB causes an output voltage change of 40/31 V.
(d) The output voltage for a digital input of 10101 is

$$V_A = \frac{10 \times 2^0 + 0 \times 2^1 + 10 \times 2^2 + 0 \times 2^3 + 10 \times 2^4}{2^5 - 1}$$

$$= \frac{10(1 + 4 + 16)}{32 - 1} = \frac{210}{31} = +6.77\ \text{V}$$

Example 12.3

What is the binary equivalent weight of each bit in a 6-bit resistive divider?

Solution:

$$\frac{1}{63}, \frac{2}{63}, \frac{4}{63}, \frac{8}{63}, \frac{16}{63}, \frac{32}{63}$$

BINARY LADDER DAC (R–2R LADDER)

• This can be used to convert a digital input into an analog output.

• **Binary Ladder** is a resistive network that produces an analog output equal to the weighted sum of digital inputs.

• It is made up of only 2 different values of resistor i.e. R & 2R. This overcomes first drawback of the resistive divider DAC.

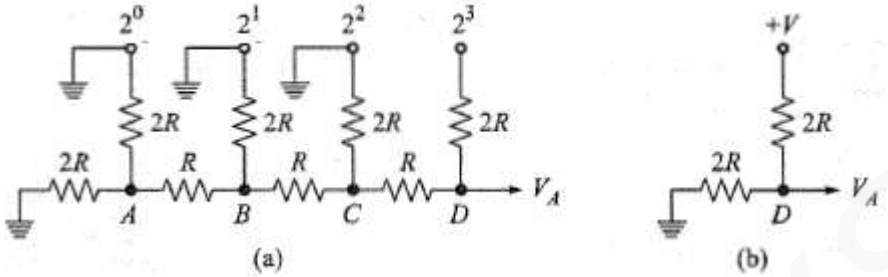


Figure 12.8: (a) Binary ladder with a digital i/p of 1000
(b) Equivalent circuit for a digital i/p of 1000

• Consider, binary ladder network for a four-bit DAC (Figure 12.8).

• Suppose the digital input is DCBA = 1000. Then the circuit is reduced to a small circuit. Its output is given by

$$V_A = V \times \frac{2R}{2R+2R} = \frac{+V}{2}$$

• The analog output voltage V_A is given by

$$V_A = \frac{V}{2} + \frac{V}{4} + \frac{V}{8} + \frac{V}{16} + \cdots + \frac{V}{2^n}$$

• In general, for an n-bit DAC, V_A is given by

$$V_A = \frac{V_0 2^0 + V_1 2^1 + V_2 2^2 + V_3 2^3 + \cdots + V_{n-1} 2^{n-1}}{2^n}$$

where V_0 ,V_1 ,V_2 ,V_3 , , , V_{n-1} are digital input voltage levels.

• The output of the ladder should be terminated with a load of 2R. This is because

1) To keep the ladder in perfect balance and to maintain symmetry. This will result in a lowering of the output voltage.

2) To ensure that the input resistance to the ladder seen by each of the digital voltage sources is constant.

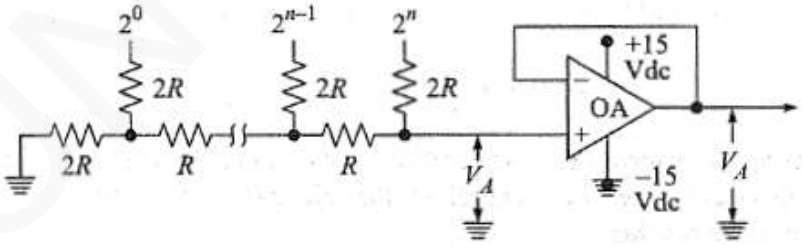


Figure 12.12: R–2R ladder DAC

• **Operational Amplifier** (OA) is connected as a unity-gain non-inverting amplifier (Figure. 12.12).

• It has very high input impedance and the output voltage is equal to the input voltage.

• It is thus a good buffer amplifier for connection to the output of a resistive ladder.

• It will not load down the ladder and thus will not disturb the ladder output voltage V_A; V_A will then appear at the output of the OA.

Example 12.4

What are the output voltages caused by each bit in a 5-bit ladder if the input levels are 0 = 0V and 1 = +10V?

Solution The output voltages can be easily calculated by using Fig. 12.10. They are

$$\text{First MSB } V_A = \frac{V}{2} = \frac{+10}{2} = +5\text{ V}$$

$$\text{Second MSB } V_A = \frac{V}{4} = \frac{+10}{4} = +2.5\text{ V}$$

$$\text{Third MSB } V_A = \frac{V}{8} = \frac{+10}{8} = +1.25\text{ V}$$

$$\text{Fourth MSB } V_A = \frac{V}{16} = \frac{+10}{16} = +0.625\text{ V}$$

$$\text{LSB = fifth MSB } V_A = \frac{V}{32} = \frac{+10}{32} = +0.3125\text{ V}$$

Bit position	Binary weight	Output voltage
MSB	1/2	$V/2$
2d MSB	1/4	$V/4$
3d MSB	1/8	$V/8$
4th MSB	1/16	$V/16$
5th MSB	1/32	$V/32$
6th MSB	1/64	$V/64$
7th MSB	1/128	$V/128$
.	.	.
.	.	.
.	.	.
Nth MSB	$1/2^N$	$V/2^N$

Figure 12.11: Binary ladder output voltages

Example 12.5

Find the output voltage from a 5-bit ladder that has a digital input of 11010.
Assume that 0=0V and 1 =+10V.

Solution

$$V_A = \frac{V_0 2^0 + V_1 2^1 + V_2 2^2 + V_3 2^3 + \cdots + V_{n-1} 2^{n-1}}{2^n}$$

$$V_A = \frac{0\times 2^0 + 10\times 2^1 + 0\times 2^2 + 10\times 2^3 + 10\times 2^4}{2^5}$$

$$= \frac{10(2+8+16)}{32} = \frac{10\times 26}{32} = +8.125\text{ V}$$

Example 12.6

What is the full-scale output voltage of the 5-bit ladder?

Solution The full-scale voltage is simply the sum of the individual bit voltages. Thus

$$V = 5 + 2.5 + 1.25 + 0.625 + 0.3125 = +9.6875\text{ V}$$

D/A CONVERTERS

• **Level Amplifier** is used to ensure that the digital signals presented to the network
  → are all of the same level and
  → are constant (Figure. 12. 13a).
• Level amplifier has 2 inputs:
  1) One input is the +10V from the precision voltage source &
  2) Another input is from a flip-flop.
• The amplifiers work in such a way that:
  ➢ When the input from a flip-flop is HIGH, the output of the amplifier is at +10V.
  ➢ When the input from a flip-flop is LOW, the output of the amplifier is at 0V.
• Finally, there must be some form of **gating on the input** of the register such that the flip-flops can be set with the proper information from the digital system.
• **Register** can be used to store the digital information (Figure. 12. 13b).
• Each flip-flop is a simple RS latch and requires a positive level at the R or S input to reset or set it.
• The flip-flops need not be reset (or set), each time new information is entered.
• When the READ IN line goes HIGH, only one of the two gate outputs connected to each flip flop is HIGH is set or reset accordingly.

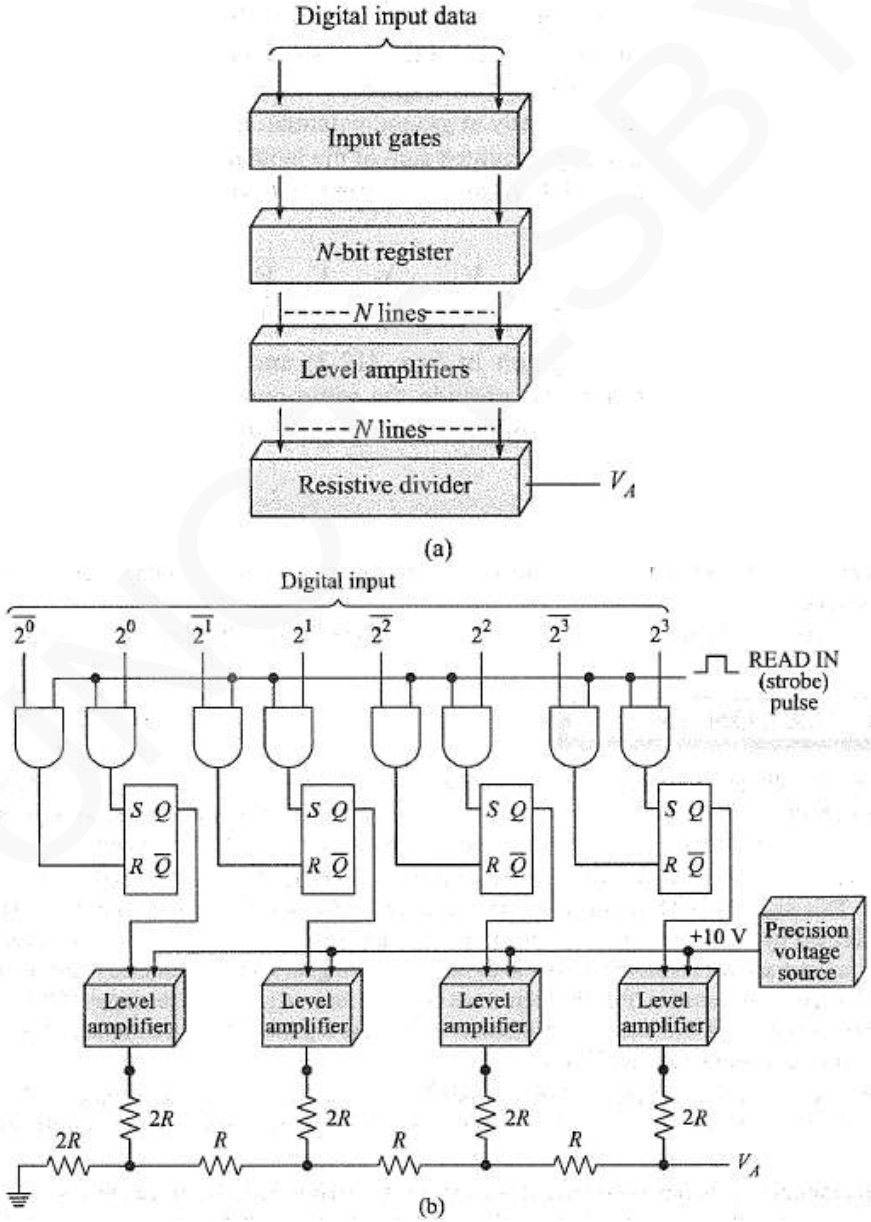


Figure 12.13: 4-bit DAC

MULTIPLE SIGNALS

• Often, it is necessary to decode more than one signal for example, the X and Y coordinates for a board.

• In this event, there are 2 ways to decode the signals:

1) Use one DAC for each signal.

Advantage

➢ Each signal to be decoded is held for in its each register & analog voltage is then held fixed. The digital input lines are connected in parallel to each converter. The proper converter is then selected for decoding by the select lines

2) Use of only one DAC and switching its output. This is called **multiplexing**.

Disadvantage

➢ The analog output signal must be held between sampling periods. The outputs must therefore be equipped with sample and hold amplifier.

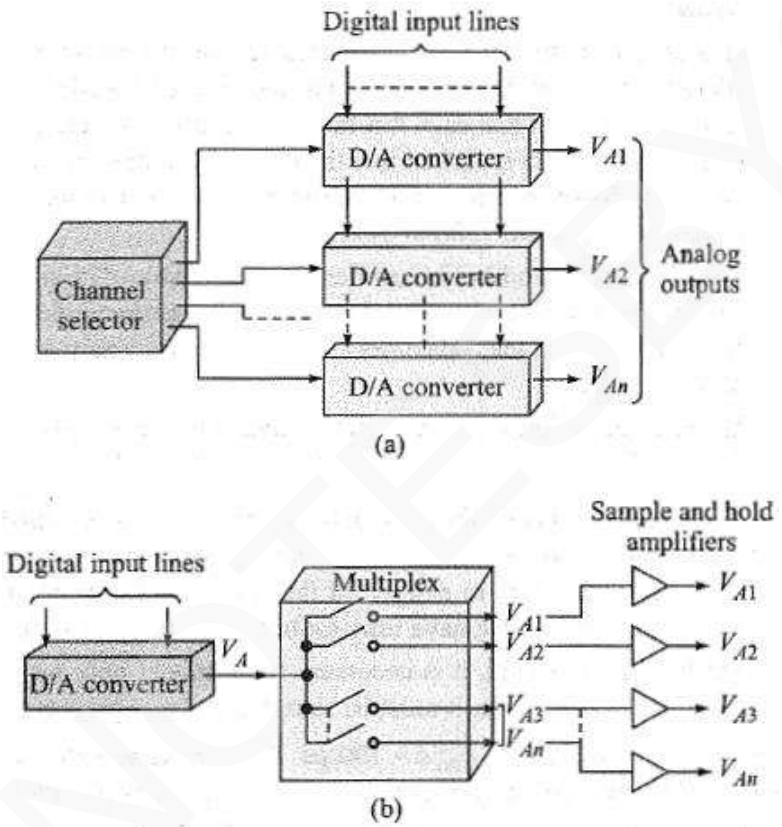


Figure 12.14 Decoding a number of signals: (a) Channel selection method, (b) Multiplex method

SAMPLE AND HOLD CIRCUIT

- This
  - → samples analog voltage signal &
  - → holds briefly to facilitate analog to digital conversion.
- This consists of
  - → One capacitor &
  - → Two operational amplifiers (OA).
- An OA is connected as a unity-gain non-inverting voltage amplifier i.e. $V_0 = V_i$ (Figure 12.14a).
- When the switch is closed, the capacitor charges to the DAC output voltage (Figure 12.14b).
  When the switch is opened, the capacitor holds the voltage level until the next sampling time.
- The operational amplifier
  - → provides a large input impedance so as not to discharge the capacitor appreciably &
  - → offers gain to drive external circuits.

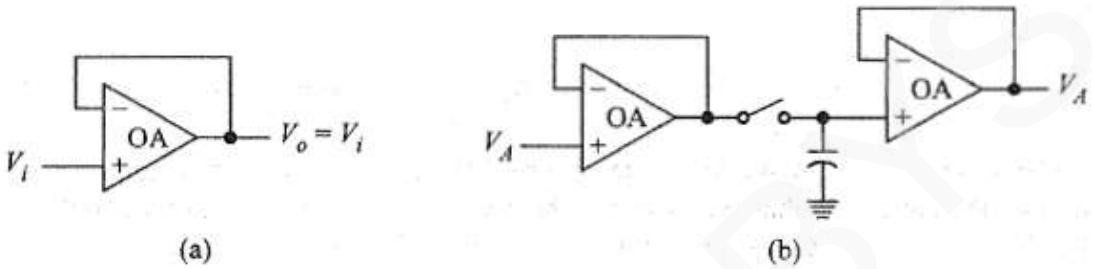


Figure 12.14 (a) Unity gain amplifier, (b) Sample-and-hold circuit

DAC TESTING

• Two simple tests can be performed to check the proper operation of the DAC: 1) Steady state accuracy test and 2) Monotonicity test.

1) Steady state accuracy test involves
- → setting a known digital number in the input register.
- → measuring the analog output with an accurate meter &
- → comparing with the theoretical value.

2) Checking for Monotonicity means checking that the output voltage increases regularly as the input digital signal increases.

- This can be accomplished by
  - → using a counter as the digital input signal &
  - → observing the analog output on an oscilloscope (Figure. 12.16b).
- For proper monotonicity, output waveform should be perfect staircase waveform (Figure.12.16a).
- The monotonicity test does not check the system for accuracy, but if the system passes the test, it is relatively certain that the converter error is less than 1 LSB.

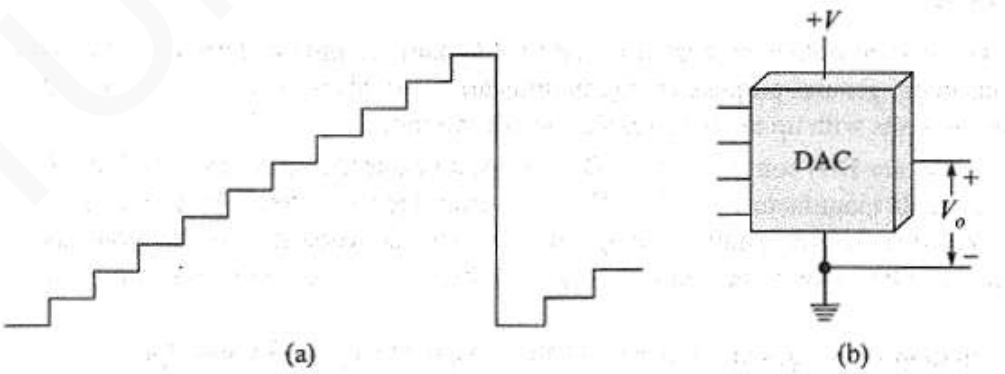


Figure 12.16 Correct output voltage waveform for monotonicity test

DAC ACCURACY & RESOLUTION

• Two important aspects of the DAC are 1) Resolution and 2) Accuracy.

1) Accuracy

• Accuracy is a measure of how close the actual output voltage is to the theoretical output value.

• The accuracy of DAC is primarily a function of

→ accuracy of the precision resistors used in the ladder &

→ precision of the reference voltage supply used.

• For example,

- Suppose that the theoretical output voltage for a particular input should be +10V.
- For example:

i) An accuracy of 10% means that the actual output voltage must be somewhere between +9 and +11 V.

ii) An accuracy of 1% means that the actual output voltage must be somewhere between +9.9 and +10.1 V

2) Resolution

• Resolution means the smallest change in digital signal that will result in a change in the analog output.

• Resolution (R) is primarily a function of the number of bits in the digital input signal.

$$\mathbf{R} = \frac{\mathbf{1}}{2^N}$$

• For example:

In a 4-bit converter, there are 2^4 =16 possible values.

Hence, resolution is 1/16. (n=4).

• The smallest increment in output voltage is determined by the LSB.

• The voltage resolution is obtained by multiplying the weight of the LSB by the full scale output voltage V_r.

$$\Delta V = \frac{V_r}{2^N}$$

V = Reference voltage range
N = Number of bits in digital output.
2^N = Number of states.
ΔV = Voltage Resolution.

Example 12.7

What is the resolution of a 9-bit DAC which uses a ladder network? What is this resolution expressed as a percent? If the full-scale output voltage of this converter is +5V, what is the resolution in volts?

Solution The LSB in a 9-bit system has a weight of $\frac{1}{512}$. Thus this converter has a resolution of 1 part in 512. The resolution expressed as a percentage is $\frac{1}{512} \times 100$ percent $\cong$ 0.2 percent. The voltage resolution is obtained by multiplying the weight of the LSB by the full-scale output voltage. Thus the resolution in volts is $\frac{1}{512} \times 5 \cong 10$ mV.

Example 12.8

How many bits are required at the input of a converter if it is necessary to resolve voltages to 5 mV and the ladder has + 10V full scale?

Solution The LSB of an 11-bit system has a resolution of $\frac{1}{2048}$. This would provide a resolution at the output of $\frac{1}{2048} \times +10 \cong +5$ mV.

Example 12.9

What is the resolution of a 12-bit DAC which uses a binary ladder? If the full-scale output is +10v, what is the resolutions in volts?

Solution:

LSB in a 12 bit system has a weight of 1/4096. Thus, this converter has a resolution of 1 part in 4096. The voltage resolution is obtained by multiplying the weight of the LSB by the full scale output voltage. Thus, the resolution in volts is (1/4096)*10 = 0.00244V = 2.44 mV

A/D CONVERTER

- The process of converting analog signal into equivalent digital signal is called **A/D conversion**.
- The electronics circuit, which does this process, is called **A/D converter (ADC)**.
- The circuit has only one input with 'n' number of digital outputs.
- Popular ADCs are:
  1) Simultaneous ADC
  2) Counter type ADC
  3) Continuous ADC
  4) Successive Approximation ADC
  5) Section counter ADC
  6) Single/ Dual RAMP ADC

SIMULTANEOUS CONVERSION ADC (PARALLEL OR FLASH TYPE)

- This is based on using a number of comparators.
- Here, two ADCs are discussed: i) 2-bit Simultaneous ADC & ii) 3-bit Simultaneous ADC.

2-bit Simultaneous ADC

- Here, three comparators are used for defining 4 voltage ranges.
- Each comparator has 2 inputs:
  1) First input is connected to analog signal to be digitized.
  2) Second input is connected to standard reference voltage like are +V/4, +V/2, and +3V/4.
- The system can accept an analog input voltage between 0 and + V.
- Basic principle:
  The **Comparator**
  → compares the analog input voltage with a reference voltage &
  → then produces proportional output i.e. it will produce either a HIGH or a LOW.
- For example,
  - If C1 is HIGH and C2 & C3 are LOW, the input must be b/w +V/4 and +V/2 (Figure. 12.19).
  - The results are summarized in Table 12.1.

Input voltage	Comparator output		
	C_1	C_2	C_3
0 to + $V/4$	Low	Low	Low
+$V/4$ to + $V/2$	High	Low	Low
+$V/2$ to + $3V/4$	High	High	Low
+$3V/4$ to + V	High	High	High

Table 12.1: Comparator outputs for input voltage ranges

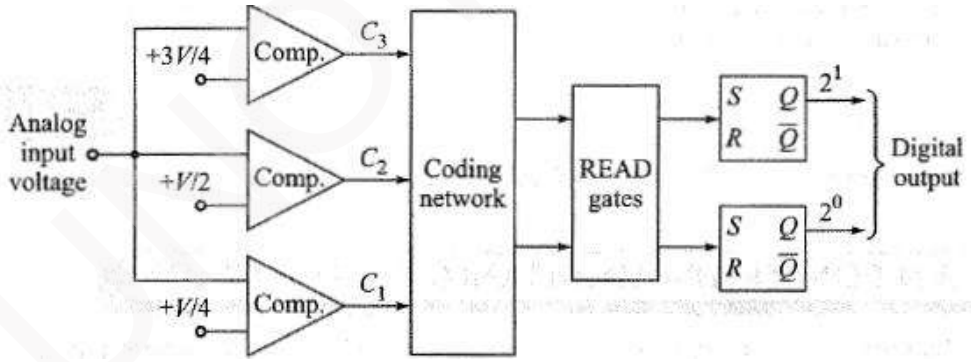


Figure 12.20: 2-bit simultaneous ADC

- This converter can detect 4 voltage ranges.
- 4 ranges can be represented by 2 bits.
- The 3 comparator outputs is then fed into a **coding network** to provide 2 bits which are equivalent to the input analog voltage.
- The bits at the output of the coding network are then entered into a flip-flop register for storage.
- Advantages:
  1) Construction of ADC is straightforward and easy to understand.
  2) It can perform extremely fast analog-to-digital conversion i.e. it takes less conversion time.
- Disadvantage:
  1) As the number of bits in the desired digital signal increases, the number of comparators increases very rapidly.

3-bit Simultaneous ADC

• Here, 7 comparators are used for defining 8 ranges.

• Some of the comparators have inverters at their outputs, since both C and C' are needed for the encoding matrix.

• **Encoding Matrix**

→ accepts 7 input levels and

→ encodes them into a 3-bit binary number (having eight possible states).

• The 2^2 bit is easiest to determine since it must be HIGH (the 2^2 flip-flop must be set) whenever C_4 is HIGH.

• The 2^1 line must be HIGH whenever C_2 is HIGH and C_4' is HIGH, or whenever C_6 is HIGH.

In equation form, we can write

$$2^1 = C_2\overline{C}_4 + C_6.$$

• The logic equation for the 2° bit can be found in a similar manner, it is

$$2^0 = C_1\overline{C}_2 + C_3\overline{C}_4 + C_5\overline{C}_6 + C_7$$

• The transfer of data from the encoding matrix into the register is carried out in 2 steps:

1) A positive reset pulse appears on the RESET line to reset all the flip-flops low.

2) Then, a positive READ pulse

→ allows the proper READ gates to go high &

→ thus transfers the digital information into the flip-flops.

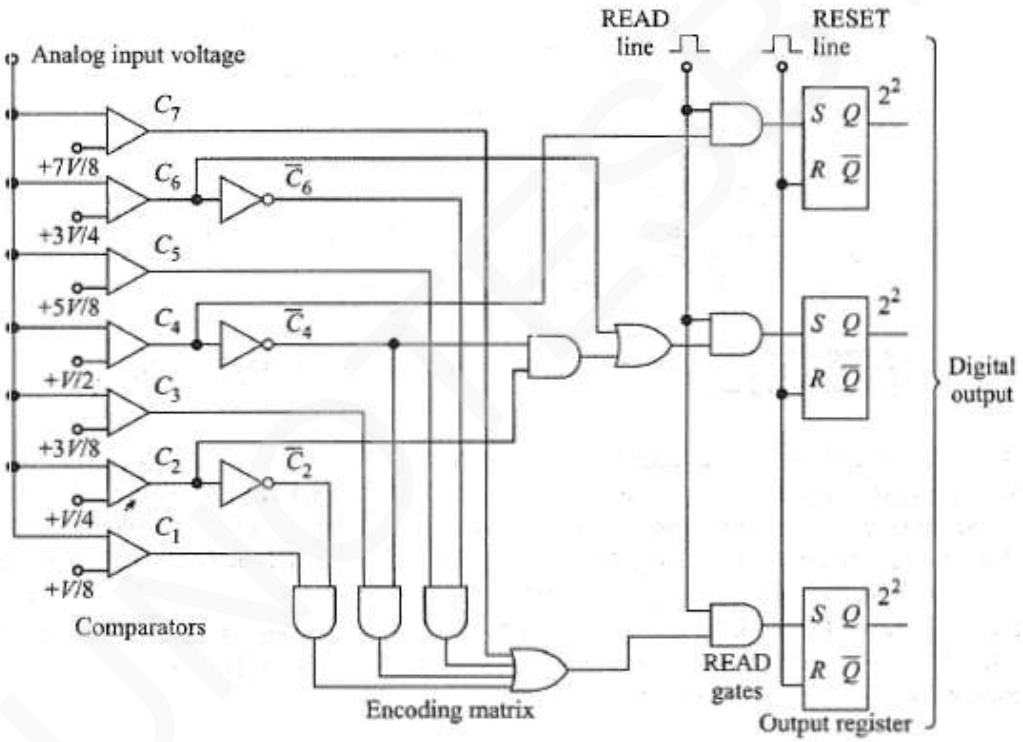


Figure 12.21:3-bit simultaneous ADC: Logic diagram

Input voltage	Comparator for level							Binary output		
	C_1	C_2	C_3	C_4	C_5	C_6	C_7	2^2	2^1	2^0
0 to V/8	Low	Low	Low	Low	Low	Low	Low	0	0	0
V/8 to V/4	High	Low	Low	Low	Low	Low	Low	0	0	1
V/4 to 3V/8	High	High	Low	Low	Low	Low	Low	0	1	0
3V/8 to V/2	High	High	High	Low	Low	Low	Low	0	1	1
V/2 to 5V/8	High	High	High	High	Low	Low	Low	1	0	0
5V/8 to 3V/4	High	High	High	High	High	Low	Low	1	0	1
3V/4 to 7V/8	High	High	High	High	High	High	Low	1	1	0
7V/8 to V	High	High	High	High	High	High	High	1	1	1

Figure 12.22 Logic table tor the converter in Figure. 12.21

COUNTER TYPE ADC

- This consists of
  - → ADC which in turn consists of the counter, level amplifiers & the binary ladder
  - → One comparator
  - → Clock block and
  - → Gate & control circuit (Figure. 12.32).
- **Comparator** has 2 inputs:
  - 1) First input is connected to analog signal to be digitized.
  - 2) Second input is connected to the analog output of binary ladder(R–2R ladder).
- **Clock–input block** produces square waves. They are connected to the gate–control block.
- If output of comparator is HIGH (logic–1);
  - Then, the comparator connects the clock pulses to the input of counter.

  If output of comparator is LOW (logic–0);
  - Then, the comparator cuts off the clock pulses, so that counting stops.
- **Level Amplifier** is used to amplify the output voltage of counter proportionally.
- Here's how it works.
  - 1) First, the counter is reset to all 0s (Figure 12.23).
  - 2) When a convert signal appears on the START line
    - → input gate is enabled &
    - → clock pulses are applied to the clock-input of the counter.
  - 3) The counter advances through its normal binary count sequence, and the staircase waveform is generated at the output of the ladder.
  - 4) This staircase waveform is applied to one side of the comparator.
    - And, the analog input voltage is applied to the other side of the comparator.
  - 5) When the reference voltage equals (or exceeds) the input analog voltage
    - → gate is disabled
    - → counter stops and
    - → conversion is complete.
  - 6) The number stored in the counter is now the digital equivalent of the analog input voltage.

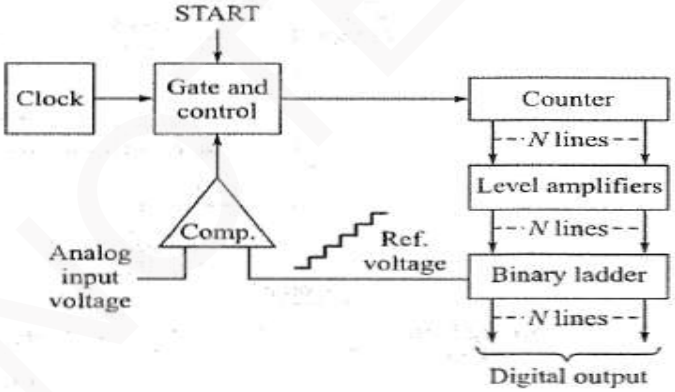


Figure 12.23 Counter type ADC

- Advantage:
  - 1) It provides a good method for digitizing to a high resolution.
- Disadvantages:
  - 1) It has longer conversion time.
  - 2) It is not suitable for digitizing rapidly changing analog signals.
- This can really be considered as a closed-loop control system.
- An error signal is generated at the output of the comparator by taking the difference between
  - → analog input signal and
  - → feedback signal (staircase reference voltage).
- When error is detected by the control circuit, the clock is allowed to advance the counter.
- The counter advances in such a way as to reduce the error by increasing the feedback voltage.
- When the error is reduced to zero
  - → feedback voltage is equal to the analog input signal.
  - → control circuitry stops the clock from advancing the counter and
  - → system comes to rest.

Example 12.10:

Find reference voltages & number of comparators required for

i) 2-bit simultaneous ADC

ii) 3-bit simultaneous ADC

Solution:

- Let V = maximum amplitude of the analog signal that the ADC can digitize.
  n = number of bits in the digitized output.
- In general, the reference voltages to be used for comparators are $V/2^n$, $2V/2^n$, $3V/2^n$, $4V/2^n$ and so on.
- The number of comparators needed for n-bit ADC is 2^n-1.

	Reference voltages	Number of Comparators needed
Two-bit ADC	V/4, V/2 and 3V/4	3
Three-bit ADC	V/8, V/4, 3V/8, V/2, 5V/8, 3V/4 and 7V/8	7

Example 12.11:

Suppose that the Counter type ADC is an 8-bit converter driven by a 500-kHz clock.

Find (a) The maximum conversion time.

(b) The average conversion time.

(c) The maximum conversion rate.

Solution

(a) An 8-bit converter has a maximum of $2^8 = 256$ counts. With a 500-kHz clock, the counter advances at the rate of 1 count each 2 μs. To advance 256 counts requires $256 \times 2 \times 10^{-6} = 512 \times 10^{-6} = 512$ μs.

(b) The average conversion time is one-half the maximum conversion time. Thus it is $1/2 \times 0.512 \times 10^{-3} = 0.256$ ms.

(c) The maximum conversion rate is determined by the longest conversion time. Since the converter has a maximum conversion time of 0.512 ms, it is capable of making at least $1/(0.512 \times 10^{-3}) \cong 1953$ conversions per second.

CONTINUOUS ADC (TRACKING TYPE)

- This consists of
  - → up down counter
  - → control lines such as count up, count down & advance line
  - → level amplifiers &
  - → binary ladder (Figure. 12.27).
- **Comparator** has 2 inputs:
  1) First input of comparator is connected to analog signal to be digitized.
  2) Second input of comparator is connected to the analog output of binary ladder(R–2R ladder).
     i) When the analog voltage is more positive than the ladder output, the up output of the comparator is HIGH.
     ii) When the analog voltage is more negative than the ladder output, the down output of the comparator is HIGH.
- Here's how it works.
  1) If the up output of the comparator is high, then
     - → AND gate at the input of the up flip-flop is enabled
     - → first time the clock goes positive & up flip-flop is set and
     - → counter will advance by one count.
  2) The converter continues to operate one conversion cycle at a time.
  3) When the analog voltage becomes more negative than the ladder output, the up line of the comparator goes low and the down line goes high.
  4) The converter then goes through a count-down conversion cycle.
- The count-down and count-up lines cannot both be high at the same time because of exclusive-OR arrangement.

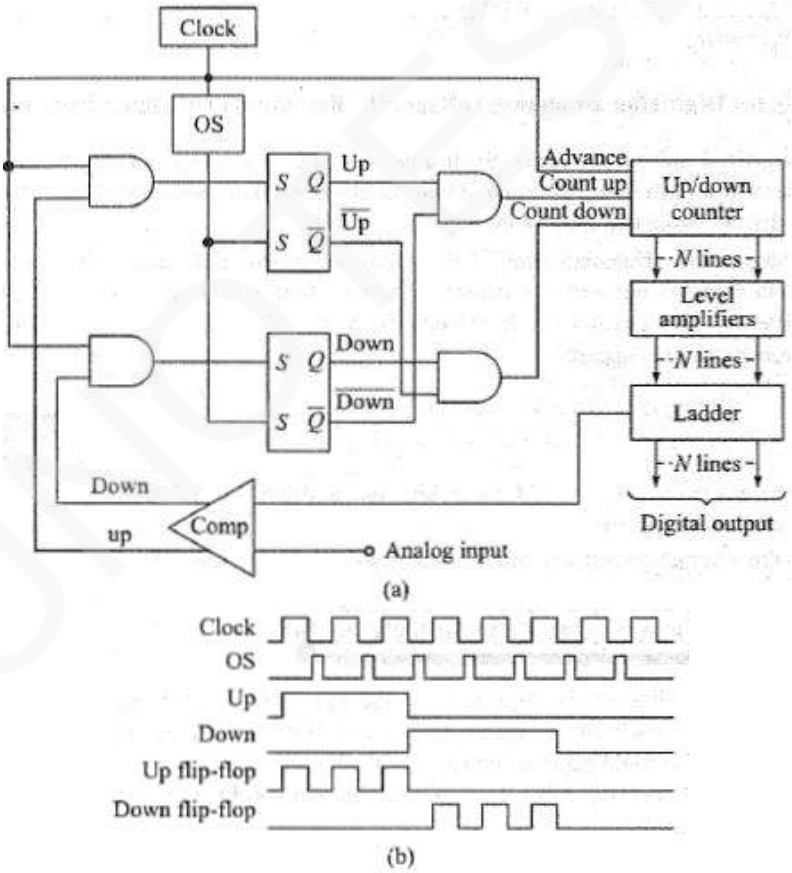


Figure 12.27 Continuous ADC

- The comparator such that the up output will not go high unless the ladder voltage is more than 1/2 LSB below the analog voltage. Similarly, down output will not go high unless the ladder voltage is more than 1/2 LSB above the analog voltage. This called **centering on the LSB** and provides a digital output which is within 1/2 LSB.

SUCCESSIVE APPROXIMATION ADC

- This is the process of approximating the analog voltage by trying 1 bit at a time.
- ADC consists of
  - → ring counter & up counter
  - → level amplifiers
  - → binary ladder and
  - → control logic & clock block (Figure. 12.27a) (SAR → Successive Approximation Register).
- The converter operates by successively dividing the voltage ranges in half (just like binary search). (The output of SAR advances with each MSB. The output of SAR does not increase step–by–step in BCD bus pattern, but individual bit becomes HIGH – starting from MSB).
- **Level Amplifier** is used to amplify the output voltage of counter proportionally.
- Here's how it works.
  1) The unknown analog voltage (V_X) is applied to the comparator.
  2) The counter starts up from 0000 and sets up first MSB i.e. 1000 (Figure. 12.27b).
  3) If V_X > 1000, the first MSB is fixed and second MSB is set.
     - i) If V_X > 1100, the second MSB is fixed and third MSB is set.
     - ii) If V_X < 1100, the second MSB is removed and third MSB is set.
  4) If V_X < 1000, the first MSB is removed and second MSB is set.
     - i) If V_X > 0100, the second MSB is fixed and third MSB is set.
     - ii) If V_X < 0100, the second MSB is removed and third MSB is set.
  5) The process is repeated down to the LSB.
  6) Finally, the desired number is in the counter.
- Since the conversion involves operating on one flip-flop at a time, a **ring counter** can be used.
- Example for SAR: Motorola MC6108 (Figure. 12.27c).

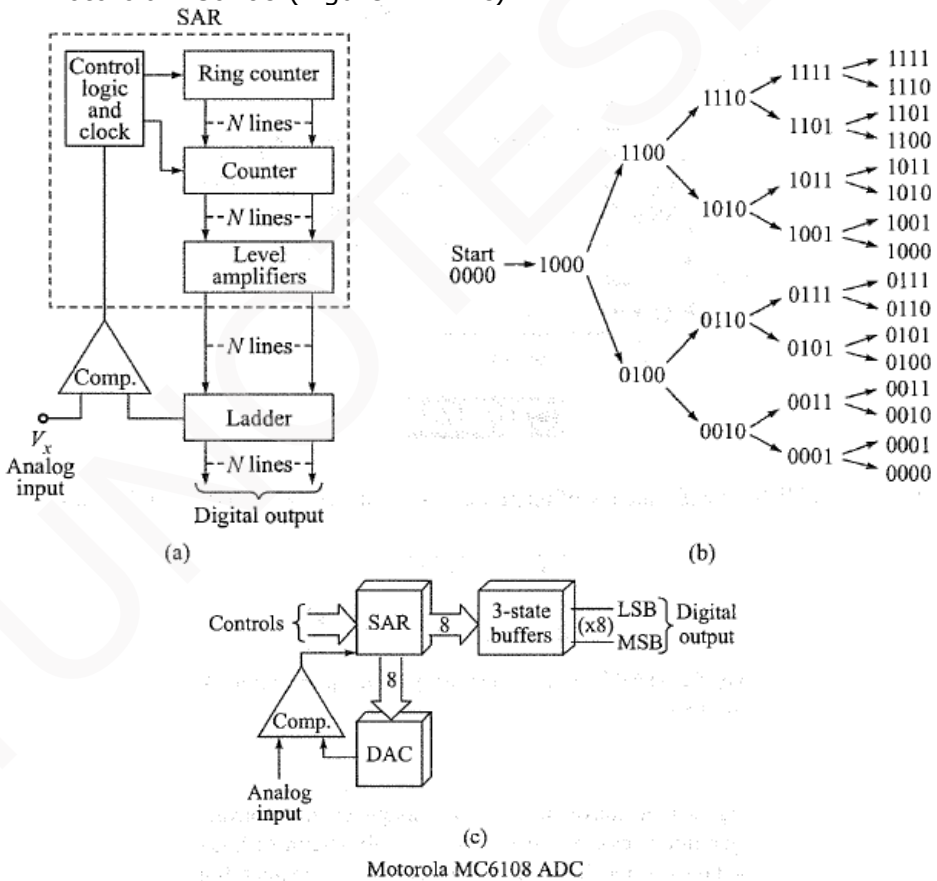


Figure 12.30 Successive approximation converter

- Advantages:
  1) This method is most useful when multiplexing is required.
  2) Each conversion
     - → takes the same time and
     - → requires one conversion cycle for each bit.

SECTION COUNTERS ADC

• A simple counter converter can be divided into sections, in order to reduce the total conversion time. Such a configuration is called **section counter**.
• Assume we have a standard 8-bit counter.
• 8-bit counter can be divided into 2 section counters of 4 bits each.
• Here's how it works.

1) The converter
   → sets the section containing the 4 LSBs to all 1s and
   → then advances the other sections until the ladder voltage exceeds the input voltage.
2) At this point the 4 LSBs are all reset. Then, advances this section until the ladder voltage exceeds the input voltage.

• A maximum of 2^4 = 16 counts is required for each section to count full scale. Thus, this method requires only $2x2^4=2^5=32$ counts to reach full scale. (This is a considerable reduction over the $2^8=256$ counts required for the straight 8-bit counter).
• There is, of course, some extra time required to set the counters initially and to switch from counter to counter during the conversion.
• This logical operation time is very small, however, compared with the total time saved by this method.

Applications:

- This type of converter is often used for digital voltmeters, since it is very convenient to divide the counters by counts of 10.
- Each counter is then used to represent one of the digits of the decimal number appearing at the output of the voltmeter.

SINGLE RAMP ADC

• This method involves comparison of the unknown input voltage with a reference voltage that begins at zero and increases linearly with time.
• The time required for the reference voltage to increase to the value of the unknown voltage is directly proportional to the magnitude of the unknown voltage.
• This time period is measured with a **digital counter**.
• This is referred to as a **single-ramp** method, since the reference voltage is sloped like a ramp.
• The heart of this converter is the ramp generator.
• **Ramp Generator** produces an output voltage ramp (Figure. 12.33a).
• The **three decade counters** are connected in cascade, and their outputs is strobed into three 4-flip-flop latch circuits.
• The **latches** are then decoded by **seven-segment decoders** to drive the LED displays as units, tens, and hundreds of counts.

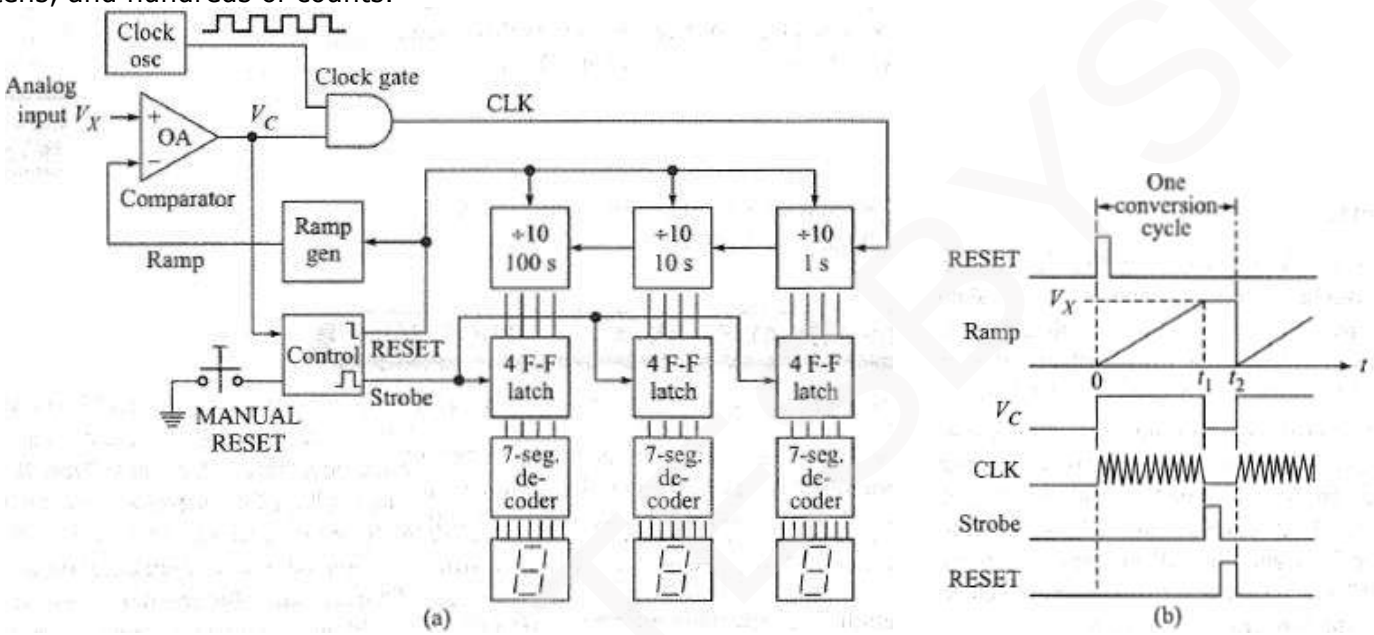


Figure 12.32 Single-slope ADC

• Here's how it works.

1) Initially,
→ decade counters are cleared to all 0s &
→ ramp is reset to 0.0 V.
2) Since V_X is positive and RAMP begins at zero, the output of the comparator V_A is HIGH.
3) This voltage V_A enables the CLOCK gate. Therefore, the clock CLK is applied to the decade counter.
4) The counter begins counting upward. And the RAMP continues counting upward until the ramp voltage is equal to the unknown input V_X.
5) At this point, time t_1, the output of the comparator V_C goes LOW, thus disabling the CLOCK gate and the counters stops counting.
6) Simultaneously, V_C=LOW generates a STROBE signal in the CONTROL box.
STROBE signal shifts contents of the 3 decade counters into three 4-flip-flop latch circuits.
7) Shortly thereafter, the contents of the previous conversion are contained in the latches and are displayed on the seven-segment LEDs.
8) In the meantime, a RESET pulse is generated by the CONTROL box.
The reset pulse
→ resets the RAMP
→ clears the decade counters to 0s and
→ begins another conversion cycle.

• Disadvantage:
➢ This depends on an extremely accurate ramp voltage.

Solution:
➢ Dual-slope ADC overcomes this problem.

DUAL SLOPE ADC

- The integrator is used as a ramp.
- There are 2 ramps:
  1) Using first ramp, the input is switched first to the unknown input voltage V_x
  2) Using second ramp, the input is switched then to a known reference voltage V_r.
- Here's how it works.
  1) Initially,
     → decade counters are cleared to all 0s
     → ramp is reset to 0.0 V and
     → input is switched to the unknown input voltage V_x.
  2) Since V_x is positive,
     → integrator outputs will be a negative ramp
     → comparator output V_g is thus positive
  3) This voltage V_g enables the CLOCK gate. Therefore, the clock CLK is applied to the decade counter.
  4) We allow the ramp to proceed for a fixed time period t_1, determined by the count-detector.
  5) When the counter reaches the fixed count at time t_1, the CONTROL unit generates a pulse to
     → clear the decade counters to all 0s &
     → switch the integrator input to the negative reference voltage V_r.
  6) The integrator will now begin to generate a ramp beginning at $-V_C$ and increasing steadily upward until it reaches 0.0V.
  7) All this time, the counter is counting, and the conversion cycle ends when V_C = 0.0V since the CLOCK GATE is now disabled.

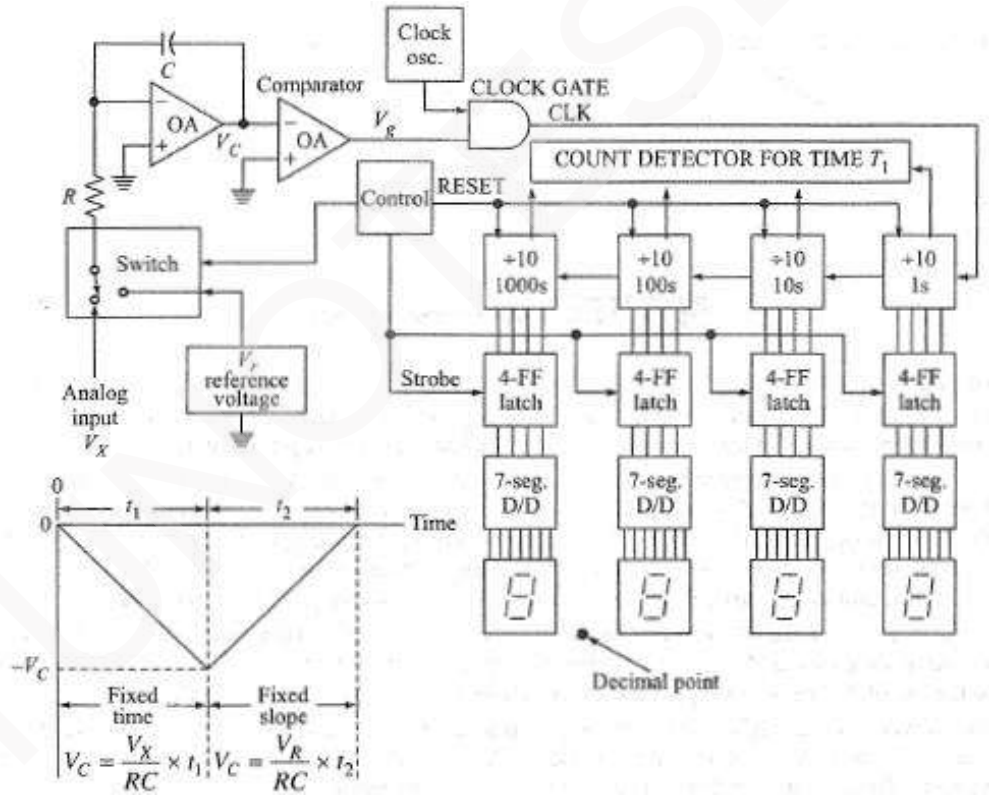


Figure 12.34 Dual-slope ADC

- Advantages:
  1) It is widely used in digital voltmeters and digital panel meters.
  2) It offers good accuracy, good linearity, and very good noise-rejection characteristics.

ADC ACCURACY AND RESOLUTION

• Since the ADC is a closed-loop system involving both analog and digital systems, the overall accuracy must include errors from both the analog and digital positions.

• **Quantization Error**
  - ➢ It is the error inherent in any digital system due to the size of the LSB.
  - ➢ It is commonly ± 1 bit.
  - ➢ If the comparator is **centered**, the quantization error can be made ±1/2 LSB.

• The main source of analog error in the ADC is comparator.
  - ➢ The sources of error in the comparator are centered around variations in the dc switching point.
  - ➢ The dc switching point is the difference between the input voltage levels that cause the output to change state.
  - ➢ Variations in dc switching are due primarily to offset, gain, and linearity of the amplifier used in the comparator.

• Differential linearity is an important measure of converter performance.

• **Differential Linearity**
  - ➢ It is a measure of the variation in size of the input voltage to an ADC which causes the converter to change from one state to the next.
  - ➢ It is usually expressed as a percent of the average step size.
  - ➢ It is best for the converters having counters that count continuously.

Example 12.12

What overall accuracy could one reasonably expect from the construction of a 10-bit ADC?

Solution A 10-bit converter has a quantization error of $\frac{1}{1024} \equiv 0.1$ percent. If the analog portion can be constructed to an accuracy of 0.1 percent, it would seem reasonable to strive for an overall accuracy of 0.2 percent.

Example 12.13

Determine the resolution of a 12-bit ADC having a full-scale analogue input voltage of 5 V.

Solution

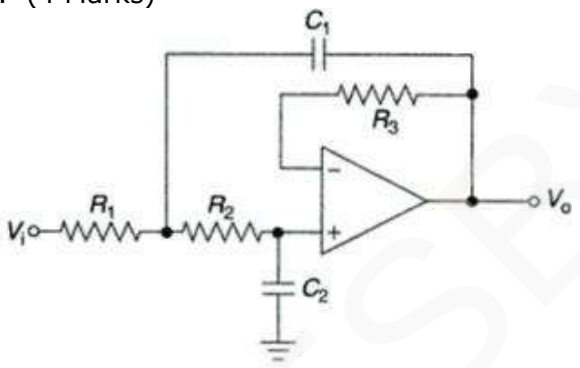
A 12-bit A/D converter resolves the analogue input voltage into $(2^{12} - 1)$ levels.

The resolution $= 5/(2^{12} - 1) = 5000/(4096 - 1) = (5000/4095) = 1.22mV$.

MODEL PAPER-1

1a. Explain with neat sketches, the operation, characteristics and parameters of N-channel depletion type MOSFET. (6 Marks)
1b. Explain with neat sketches, the operation of JFET along with its characteristics curve. (5 Marks)
1c. Explain the operation of monostable multivibrator with a neat diagram. (5 Marks)

2a. Compare ideal opamp vs. practical opamp. (6 Marks)
2b. With a circuit diagram, explain the working of window comparator. (6 Marks)
2c. Below figure shows a second-order low-pass filter built around a single opamp. Calculate the values of R_1, R_2, C_1, C_2 and R_3 if the filter had a cut-off frequency of 10 kHz, Q-factor of 0.707 and input impedance not less than 10 kΩ. (4 Marks)



3a. What are universal gates? Implement the basic gates using universal gates only. (4 Marks)
3b. Explain don't care condition with example. (4 Marks)
3c. What is a karnaugh map? Also, list out limitations of kmap. (4 Marks)
3d. Find minimal SOP of following boolean functions using kmap. Give implementation using NAND gates only: (4 marks)

Y=F(A,B,C,D)=∑m(7,9, 10, 11, 12, 13, 14, 15)

4a. Find minimal POS of following Boolean function using kmap. Give implementation using NOR gates only: (4 marks)

Y=F(A,B,C,D)= ∏M(4,6,12,14)

4b.Simplify the following Boolean function using Quine McClusky method: (8 marks)

Y=F(A,B,C,D)= ∑m(0,1,3,7,8,9,11,15)

4c. Write verilog HDL program for full adder. (4 Marks)

5a. Explain demultiplexer with block diagram. Show how two 1:16 demultiplexers can be connected to get a 1:32 demultiplexer. (5 Marks)
5b. Explain encoder with block diagram. Design a priority encoder the truth table of which is shown in below figure. The order of priority for three inputs is X1 > X2 > X3. However, if the encoder is not enabled by S or all the inputs are inactive the output AB= 00. (8 Marks)

	Intput			Output	
S	X_1	X_2	X_3	A	B
0	×	×	×	0	0
1	1	×	×	0	1
1	0	1	×	1	0
1	0	0	1	1	1
1	0	0	0	0	0

5c. Draw a 3-input & 3-output PAL circuit for following four Boolean functions: (3 Marks)

Y2=ABC+ABC'+B'C'; Y1=ABC'+AB'C+A'BC'; Y0=A'BC'+ABC'+A'BC

MODEL PAPER-1 (CONT.)

6a. Differentiate between combinational circuit & sequential circuits. (2 Marks)
6b. Explain edge-triggered JK flip-flop with its truth table & logic diagram. (6 Marks)
6c. Explain clocked RS flip-flop with its truth table & logic diagram. What is the drawback of clocked RS flip-flops. (8 Marks)

7a. Explain JK master slave flip-flop. (6 Marks)
7b. Show how a D Flip - Flop can be converted into JK flip-flop. (2 Marks)
7c. Explain the working of 7491 8-bit shift register with logic diagram. (6 Marks)
7d. How long will it take to shift an 8-bit binary number into the 8-bit shift register (54/74164) if the clock is 5 Mhz? (2 Marks)

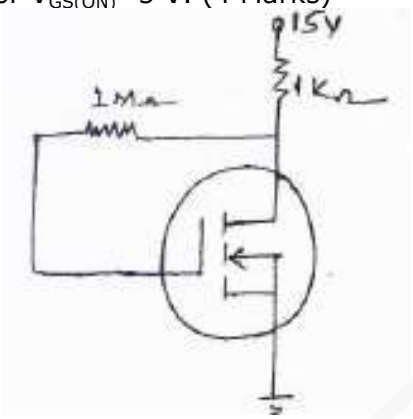
8a. Explain the working of 4-bit Johnson counter with logic diagram & truth table. (5 Marks)
8b. Write verilog code for 4-bit parallel out-right shift register. (3 Marks)
8c. Explain working of a 3-bit asynchronous up-counter with truth table and waveforms. (6 Marks)
8d. What is the clock frequency in mod-8 counter,
i) If the period of the waveform at MSB is 24 μs?
ii) If the period of the waveform at LSB is 18 μs? (2 Marks)

9a. Design a self correcting modulo-3 down counter using SR flip-flops. (4 Marks)
9b. Define the following: i) Decoding gate ii) Lock out of a counter. (2 Marks)
9c. Design a modulo-8 up down counter which counts in upward direction if input MODE = 0, else counts in downward direction. It should also have a parallel load facility. When PL = 1, a 3-bit number D is asynchronously loaded to the counter. The counter counts at the negative edge of CLOCK and its output is represented by Q. (4 Marks)
9d. With block diagram, explain operation of resistive divider DAC. (6 Marks)

10a. With block diagram, explain operation of 3-bit simultaneous ADC. (7 Marks)
10b. With block diagram, explain operation of successive approximation ADC. (7 Marks)
10c. What overall accuracy could one reasonably expect from construction of a 10-bit ADC? (2 Marks)

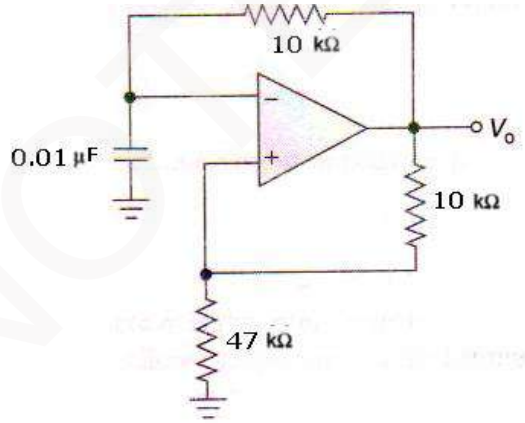
MODEL PAPER-2

1a. What is the differences between JFET & MOSFET's? (2 Marks)
1b. Calculate the value of operating point for the circuit shown below given that threshold voltage for the MOSFET is 2V and $I_{D(ON)}$=6 mA for $V_{GS(ON)}$=5 V. (4 Marks)



1c. Explain any 3 FET applications with circuit diagram. (4 Marks)
1d. Explain the operation of astable multivibrator using IC 555 with a neat diagram. (6 Marks)

2a. List out & explain performance parameters of opamp. (6 Marks)
2b. With a circuit diagram, explain the working of first-order low-pass active filter. (6 Marks)
2c. Refer to the below figure. Determine the peak-to-peak amplitude and frequency of the square wave output given that saturation output-voltage of the opamp is +12.5 V at power supply voltages of +15V. (4 Marks)



3a. Explain the following gates: i) AND, ii) OR & iii) NOT (4 marks)
3b. State 2 De Morgans theorem. (4 Marks)
3c. Find minimal SOP of following boolean functions using kmap. Give implementation using NAND gates only: (4 Marks)
Y=F(A, B, C)=A'B'C'+ A'B'C+ A'BC+ AB'C
3d. Simplify the following boolean function using entered variable map method: (4 Marks)
Y=f(A,B,C,D)= ∑m(0,1,5,6,7,8,9,13)

4a. Simplify the following Boolean function using Quine McClusky method: (8 marks)
Y=f(A,B,C,D)= ∑m(1,2,8,9,10,12,13,14)
4b. What are prime implicants and essential prime implicants. (2 marks)
4c. Explain static-1 hazard and its cover. (6 Marks)

MODEL PAPER-2 (CONT.)

5a. Write a verilog code for 4:1 multiplexer. Show how 8:1 multiplexer can be used to compare two 2-bit numbers, A_1A_0 and B_1B_0 to generate two outputs, A>B and A= B. (6 Marks)
5b. Explain BCD-to-Decimal decoder. (6 Marks)
5c. Design 7 segment decoder using PLA. (4 Marks)

6a. Explain functional representation of ALU IC 74181 with its truth table. (8 Marks)
6b. Draw the logic diagram of clocked D flip-flop. Write its truth table & characteristic equation, state diagram and excitation table. What is the drawback of JK flip flops? (8 Marks)

7a. What is switch contact bounce? Explain working of a simple RS latch debounce circuit. (4 Marks)
7b. Show how a SR flip-flop can be converted into JK flip–flop. Calculate the clock cycle for a system that uses a clock, that has a frequency of
i)10 MHz ii)50 MHz (4 Marks)
7c. Explain the working of PIPO shift register (74174) with logic diagram? (4 Marks)
7d. What are universal shift register (74194)? Explain along with function table? (4 Marks)

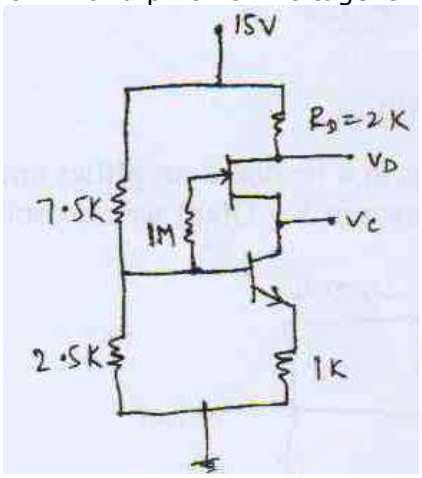
8a. What are sequence generator and sequence detector? Explain both with neat diagram. (5 Marks)
8b. Write verilog code for Johnson counter. (3 Marks)
8c. Explain working of a 3-bit asynchronous down-counter with truth table and waveforms. (6 Marks)
8d. What are decoding gates? Draw waveform for state 7 in a 3-bit synchronous up counter. (2 Marks)

9a. Write a short note on presettable counters. (4 Marks)
9b. Show how a modulo-4 counter designed with two flip-flops can generate a repetitive sequence of binary word '1101' with minimum number of memory elements. (4 Marks)
9c. With block diagram, explain operation of R–2R ladder DAC. (6 Marks)
9d. What is the resolution of a 12-bit DAC which uses a binary ladder? What is this resolution expressed as a percent? If the full-scale output is +10v, what is the resolutions in volts? (2 Marks)

10a. With block diagram, explain operation of continuous ADC. (8 Marks)
10b. Explain accuracy and resolution for ADC. (4 Marks)
10c. Suppose that the Counter type ADC is an 8-bit converter driven by a 500-kHz clock.
Find (i) The maximum conversion time.
(ii) The average conversion time.
(iii) The maximum conversion rate (4 Marks)

MODEL PAPER-3

1a. Explain the working of a N-channel E-MOSFET with neat diagram. Explain with a Diagram output characteristics of the same. (6 Marks)
1b. Find the values of voltages V_D and V_C for the circuit shown below. Assume ß=100, V_{BE}= 0.7 V, saturation drain current of JFET is -10 mA and pinch off voltage is -5V. (4 Marks)



1c. Explain the operation of bistable multivibrator with a neat diagram. (6 Marks)

2a. Opamp LM 741 is specified to have a slew rate of 0.5 V/μs. if the opamp were used as an amplifier and the expected peak output-voltage were 10V, determine the highest sinusoidal frequency that would get satisfactorily amplified. (4 Marks)
2b.With a circuit diagram, explain the working of zero-crossing detector. (5 Marks)
2c. With a circuit diagram, explain the working of relaxation oscillator. (5 Marks)
2d. Explain the various electrical characteristics of an ideal opamp. (2 Marks)

3a. Realize Y = AB + C' using only one type of gate. (4 Marks)
3b. What is HDL? Explain and list out its advantages. (4 Marks)
3c. Find minimal SOP of following boolean functions using kmap. Give implementation using NAND gates only: (4 marks)
Y=F(A,B,C,D)= ∑m(6,7,9,10,13)+d(1,4,5,11)
3d. List out the steps for simplifying Boolean equation using kmap method. Also, list out limitations of kmap. (4 marks)

4a. Find minimal POS of following boolean function using kmap. Give implementation using NOR gates only: (4 marks)
Y=F(A,B,C,D)=∏M(0,1,3,4,5,7)
4b. Simplify the following Boolean function using Quine McClusky method: (6 marks)
Y=F(A,B,C,D)=∑m(1,3,6,7,8,9,10,12,13,14)
4c. Explain dynamic hazard with example. (6 Marks)

5a. Explain multiplexer with diagram and truth table? Also, write verilog code for a 2:1 Mux (6 Marks)
5b. Show how 4:16 decoder can be used to compare two 2-bit numbers, A_1A_0 and B_1B_0 to generate two outputs, A>B and A= B. (4 Marks)
5c. What are parity generator & parity checker? Explain both of them along with neat diagrams? What are applications of parity generator & checker? (6 Marks)

6a. Differentiate transparent & gated flip flops. What are their applications? (4 Marks)
6b. Explain how RS flip-flop is constructed using NOR-gate latch. (6 Marks)
6c. Explain edge-triggered D flip-flop with its truth table & logic diagram. (6 Marks)

MODEL PAPER-3 (CONT.)

7a. Give state transition diagram of SR, D, JK and T Flip - Flop. (2 Marks)
7b. Using a behavioral model, write a verilog code for a D flip-flop. (2 Marks)
7c. Explain the working of 4-bit SISO shift register with logic diagram & truth table. (6 Marks)
7d. Explain the working of 4-bit parallel access shift register (7495A) with logic diagram. (6 Marks)

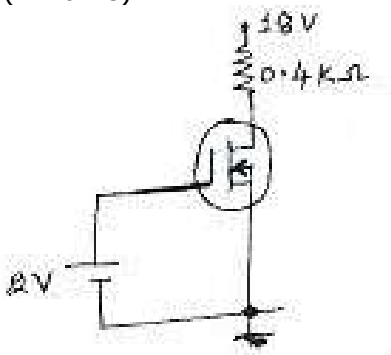
8a. What is a serial adder? Explain with neat diagram? (5 Marks)
8b. Write verilog code for 4-bit serial input shift register? (3 Marks)
8c. Design an 8-bit sequence generator that generates the sequence 11000100 repetitively using mod-8 Johnson counter. (3 Marks)
8d. Explain working of synchronous 3-bit up counter along with truth table and waveforms. (5 Marks)

9a. Design a self-correcting rnodulo-6 counter using JK flip-flops. (6 Marks)
9b. Write a verilog code for a modulo-8 up counter using JK flip-flop. (4 Marks)
9c. With block diagram, explain operation of 4-bit DAC. (6 Marks)

10a. With block diagram, explain operation of counter type ADC. (8 Marks)
10b. With block diagram, explain operation of dual-slope ADC. (8 Marks)

MODEL PAPER-4

1a. Explain the 4 characteristics of n-channel JFET and define various condition. (6 Marks)
1b. Below figure shows a biasing configuration using DE-MOSFET. Given that the saturation drain current is 8 mA and the pinch voltage is -2 V. Determine the value of gate source voltage, drain current and the drain source voltage. (4 Marks)

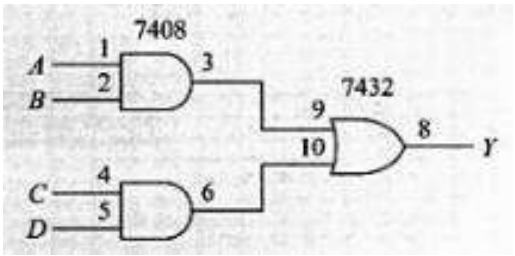


1c. Explain the working of CMOS inverter. Mention any 2 advantages. (6 Marks)

2a. Differential voltage gain and CMRR of an opamp when expressed in decibels are 110 dB and 100 dB respectively. Determine the common mode gain expressed as a ratio. (4 Marks)
2b. With a circuit diagram, explain the working of peak detector circuit (6 Marks)
2c. Explain the following:
i) Current-to-Voltage converter
ii) Voltage-to-Current converter (6 Marks)

3a. Compare positive logic vs negative logic. (3 Marks)
3b. Explain preparation of test bench. (5 Marks)
3c. Find minimal SOP of following Boolean functions using kmap. Give implementation using NAND gates only: (4 Marks)
i) f(A,B,C,D)= ∑m(0,1,3,5,6,11,13)+d(4,7,8)
3d. Simplify the following boolean function using entered variable map method: (4 marks)
Y=F(A, B, C) =A'B'C'+ A'B'C+ A'BC+ AB'C (4 Marks)

4a. Simplify the following Boolean function using Quine McClusky method: (6 Marks)
Y=f(A,B,C,D)= ∑m(0,1,2,3,10,11,12,13,14,15)
4b. Explain static-0 hazard and its cover. (6 Marks)
4c. What are the three different models for writing a module body in Verilog HDL. Write verilog code for following circuit. (4 Marks)



5a. Design a 32-to-1 multiplexer using two 16-to-1 multiplexers and one 2-to-l multiplexer. (2 Marks)
5b. Explain seven-segment decoder. (6 Marks)
5c. What is magnitude comparator? Explain 1-bit magnitude comparator with logic diagram and truth table. (6 Marks)
5d. What is PLA? How does PLA differ from PAL? (2 Marks)

MODEL PAPER-4 (CONT.)

6a. Explain different arithmetic building blocks with diagram for each. (6 Marks)
6b. Explain the operation of a flip-flop as a bistable device. (4 Marks)
6c. Explain edge-triggered RS flip-flop with its truth table & logic diagram. (6 Marks)

7a. Define i) Hold time ii) Set up time iii) Flip flop iv) Characteristic equation. (4 Marks)
7b. Using a behavioral model, write a verilog code for a SR flip-flop. (2 Marks)
7c. Name & explain the four basic types of shift registers and draw a block diagram for each. (4 Marks)
7d. Explain the working of 8-bit SIPO shift register (74164) with logic diagram? (6 Marks)

8a. Explain working of 8-bit ring counter with logic diagram & truth table. (5 Marks)
8b. Write verilog code for a shift register of 6 bits constructed using D flip-flops. (3 Marks)
8c. Differentiate between ripple and synchronous counter? (3 Marks)
8d. Explain the working of synchronous 4-bit up-down counter. (5 Marks)

9a. Design a modulo-4 irregular counter with following counting sequence using D flip-flop:00→10→11→01. (4 Marks)
9b. Write a verilog code for a modulo-8 up counter. (4 Marks)
9c. For a 5-bit resistive divider, determine the following:
(a) The weight assigned to the LSB.
(b) The weight assigned to the second and third LSB.
(c) The change in output voltage due to a change in the LSB, the second LSB, & the third LSB.
(d) The output voltage for a digital input of 10101.
Assume 0 =0V and 1 =+10V. (4 Marks)
9d. Explain accuracy and resolution for DAC. (4 Marks)

10a. With block diagram, explain operation of 2-bit simultaneous ADC. (7 Marks)
10b. With block diagram, explain operation of single-slope ADC. (7 Marks)
10c. What overall accuracy could one reasonably expect from construction of a 10-bit ADC? (2 Marks)