Digital System Design(15A04504)

Prepared By

N PRAKASH BABU,
Associate Professor,
Department of ECE.

COURSE OUTCOMES

C315.1 Explain the operation CMOS, TTL logic families ECL logic families and interfacing between them.

C315.2 Demonstrate the concept in HDL and Model digital logic circuits using hardware description languages like VHDL

C315.3 Design and implement various Combinational circuits using basic IC structures

C315.4 Design and implement various sequential circuits using VHDL

C315.5 Develop VHDL programs for various complex combinational and Sequential circuits using VHDL

WAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY ANANTAPUR

Tech III-ISem. (ECE) **L T P C**

3 1 0 3

15A04504- DIGITAL SYSTEM DESIGN

IT-I

OS LOGIC: Introduction to logic families, CMOS logic, CMOS logic families; BIPOLAR
GIC AND INTERFACING**: Bipolar logic, Transistor logic, TTL** families, CMOS/TTL
rfacing, low voltage CMOS logic and interfacing, Emitter coupled logic, Comparison
c families, Familiarity with standard 74-series and CMOS 40- series-ICs – Specificat

T-II

WARE DESCRIPTION LANGUAGES: HDL Based Digital Design, The VHDL

vare Description Language–Program Structure, Types, Constants and Arrays,
ions and procedures, Libraries and Packages, Structural design elements,
low design elements, Behavioral design elements, The Time Dimension,
ation, Test Benches, VHDL Features for Sequential Logic Design, Synthesis

T-III

BINATIONAL LOGIC DESIGN PRACTICES: Description of basic structures like

ders, Encoders, Comparators, Multiplexers (74 –series MSI); Design of
lex Combinational circuits using the basic structures; Designing Using
inational PLDs like PLAs, PALs ,PROMs CMOS PLDs; Adders & sub tractors,
Combinational multipliers; VHDL models for the above standard building bl

T-IV

ENTIAL MACHINE DESIGN PRACTICES: Review of design of State machines;

ard building block ICs for Shift registers, parallel / serial conversion , shift
er counters, Ring counters; Johnson counters, LFSR counter ; VHDL models f
bove standard building block ICs. Synchronous Design example using standa

T –V

n Examples (using VHDL): Barrel shifter, comparators, floating-point encod
dual parity encoder. **Sequential logic Design: Latches & flip flops, PLDs,**
nters, shift register and their VHDL models.

Books:

n F.Wakerly ,“Digital Design Principles and Practices” 4th edition,
on Education., 2009

rles H.Roth,Jr., “Fundamentals of Logic Design” 5th edition , CENGAGE
ng 2012.

erences:

Morris Mano and Michael D. Cilleti., “Digital Logic Design” 4th edition
on Education., 2013

phen Brown and ZvonkoVranesic, “Fundamentals of digital logic with
design” 2nd edition McGraw Hill Higher Education.

hasker, “A VHDL PRIMER” 3rd edition Eastern Economy Edition, PHI
ng,2010.

UNIT-I
CMOS LOGIC

roduction to logic families, CMOS logic, CMOS steady state elec

navior, CMOS dynamic electrical behavior, CMOS logic families.

olar Logic and Interfacing; Bipolar logic, Transistor logic, TTL fam

OS/TTL interfacing, low voltage CMOS logic and interfacing, em

upled logic, comparison of logic families, Familiarity with standard 74-s

d CMOS 40- series-ICs - specifications.

INTRODUCTION

he Start of the Modern Electronics E

een, Shockley, and Brattain at Bell
s - Brattain and Bardeen invented
the bipolar transistor in 1947.

The first germanium bipolar trans
Roughly 50 years later, electron
account for 10% (4 trillion dollar
the world GDP.

Electronics Milestones

Braun invents the solid-state rectifier.

DeForest invents triode vacuum tube.

27

First radio circuits developed from diodes and triodes.

Lilienfeld field-effect device patent filed.

Bardeen and Brattain at Bell Laboratories invent bipolar transistors.

Commercial bipolar transistor production at Texas Instruments.

Bardeen, Brattain, and Shockley receive Nobel prize.

1958	Integrated circuit developed by Kilb Noyce
1961	First commercial IC from Fairchild Semiconductor
1963	IEEE formed from merger or IRE and
1968	First commercial IC opamp
1970	One transistor DRAM cell invented b Dennard at IBM.
1971	4004 Intel microprocessor introduce
1978	First commercial 1-kilobit memory.
1974	8080 microprocessor introduced.
1984	Megabit memory chip introduced.
2000	Alferov, Kilby, and Kromer share Nob prize

INTRODUCTI

volution of Electronic Devices

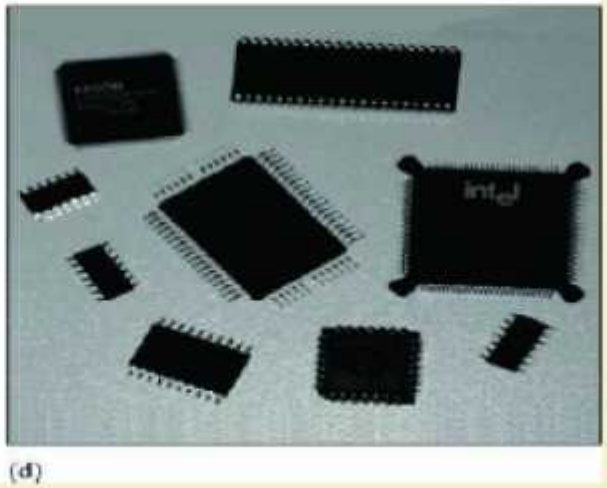
cuum
ubes

nd MSI
grated
rcuits


(a)


(b)

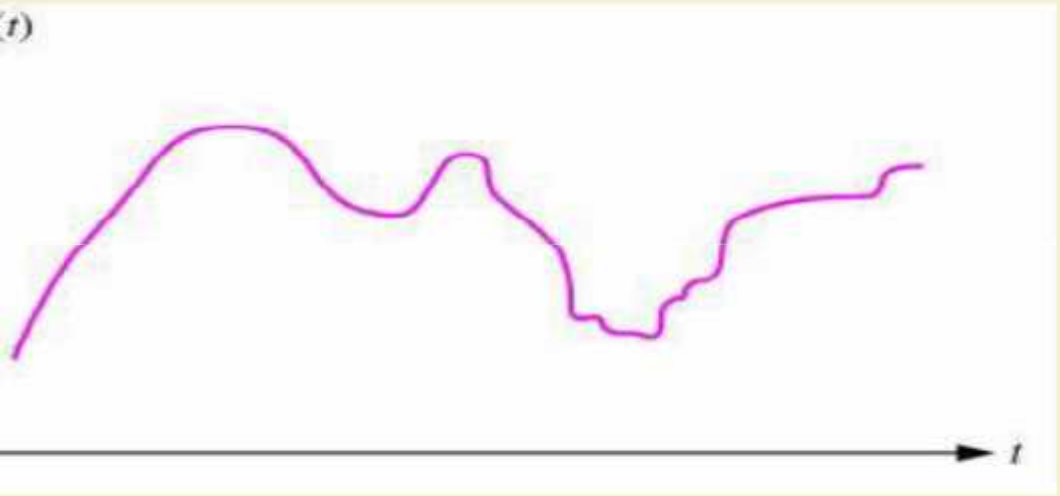

(c)

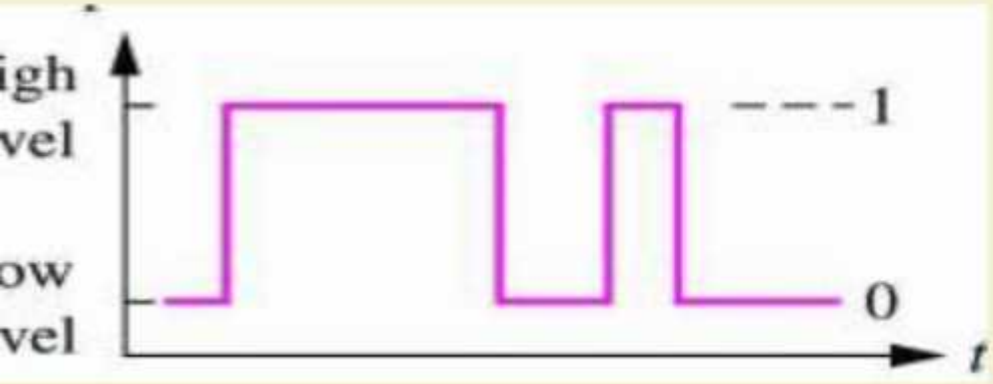

(d)

Discrete
Transisto

VLSI
Surface-M
Circuit

Signal Types





- Analog signals take on contin values - typically current or voltage.
- Digital signals appear at discr levels. Usually we use binary signals which utilize only two levels.
- One level is referred to as log 1 and logical 0 is assigned to other level.

INTRODUCTI

Physical Representation of bits in Different Applications

Technology	State Representing Bit	
	0	1
Pneumatic logic	Fluid at low pressure	Fluid at high pressure
Relay logic	Circuit open	Circuit closed
Complementary metal-oxide semi conductor (CMOS) logic	0–1.5 V	3.5–5.0 V
Transistor-transistor logic (TTL)	0–0.8 V	2.0–5.0 V
Fiber optics	Light off	Light on
Dynamic memory	Capacitor discharged	Capacitor charged
Nonvolatile, erasable memory	Electrons trapped	Electrons released
Bipolar read-only memory	Fuse blown	Fuse intact
Bubble memory	No magnetic bubble	Bubble present
Magnetic tape or disk	Flux direction "north"	Flux direction "south"
Polymer memory	Molecule in state A	Molecule in state B
Read-only compact disc	No pit	Pit
Rewriteable compact disc	Dye in crystalline state	Dye in non-crystalline state

INTRODUCTI

en discussing electronic logic circuits such as **CMOS and TTL**, digital designers oft

words “**LOW**” and “**HIGH**” in place of “**0**” and “**1**” .

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

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

ere are two logic types

1. Positive logic
2. Negative logic

igning **0 to LOW** and **1 to HIGH** seems most natural, and is called **positive logic**.

e opposite assignment, i.e. **1 to LOW** and **0 to HIGH**, is not often used, and is called **n**

ic.

INTRODUCTI

Gates:

gic gates are the basic building blocks of any **digital system**. It is an electronic circuit hav

more than one input and only one output.

gic gates are named as **AND gate, OR gate, NOT gate** etc.

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

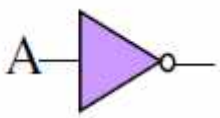
OR gate produces a **1** if and only if **one or more of its inputs are 1.**

NOT gate, usually called an inverter, produces an output value that is the **opposite of it**

lue.

NOT

A	C
0	1
1	0

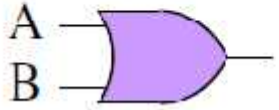


AND

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

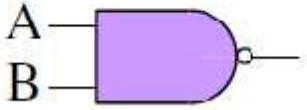
OR

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



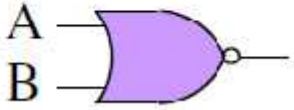
NAND

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



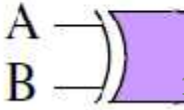
NOR

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



XO

A	B
0	0
0	1
1	0
1	1



INTRODUCTI

l Logic Family:

c family is a collection of different integrated circuit chips that have similar input, outp

al circuit characteristics, but that perform different logic functions.

ps from the same family can be interconnected to perform any desired logic function.

the other hand, chips from differing families may not be compatible; they may use d

wer supply voltages or may use different input and output conditions to represent logic va

of Logic Families:

r Logic Families

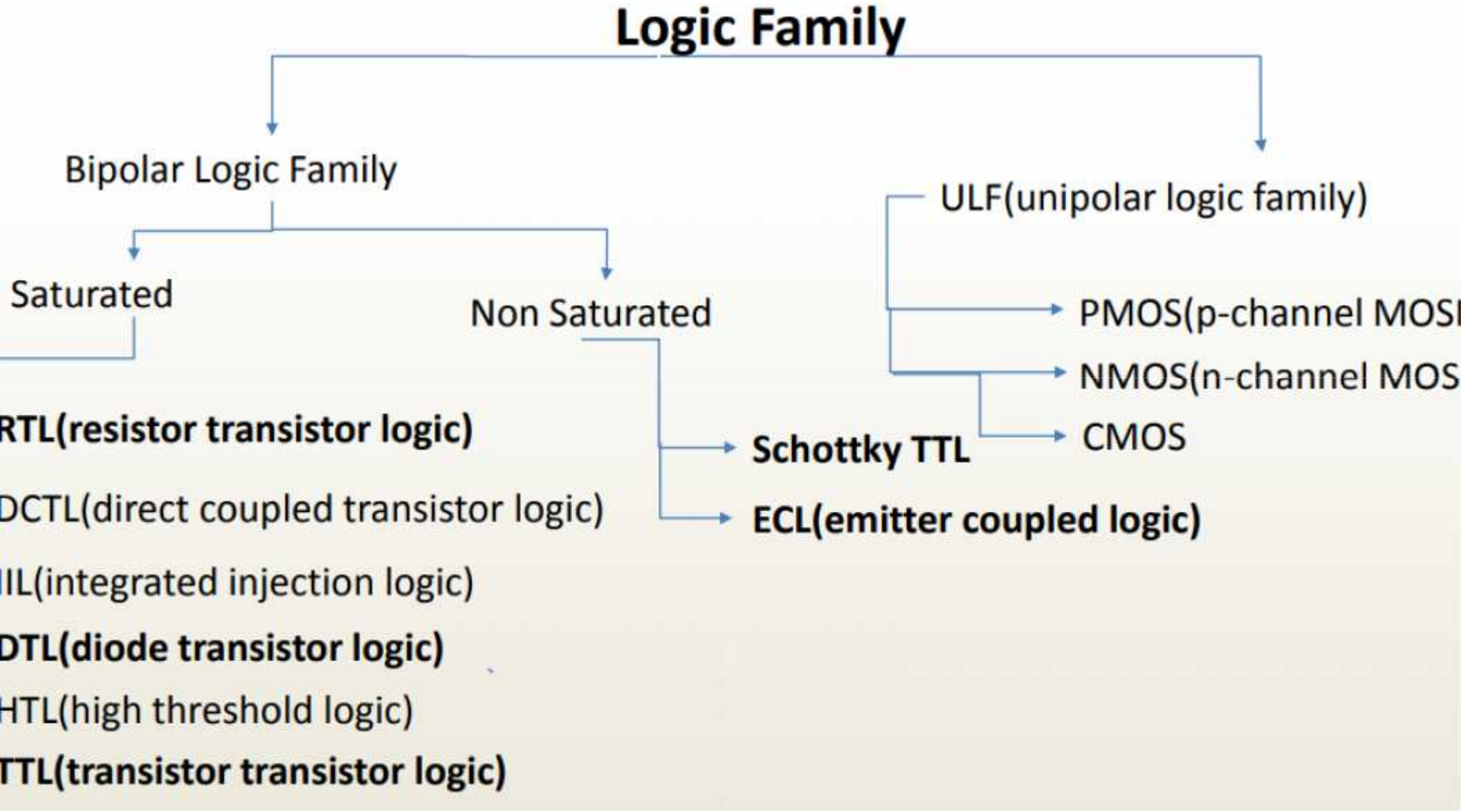
- **Diode logic (DL)**
- **Resistor-Transistor logic (RTL)**
- **Diode-Transistor logic (DTL)**
- **Transistor-Transistor logic (TTL)**
- **Emitter coupled logic (ECL)**

Unipolar Logic Familes

1. **NMOS**
2. **PMOS**
3. **CMOS**

INTRODUCTI

l Logic Family:



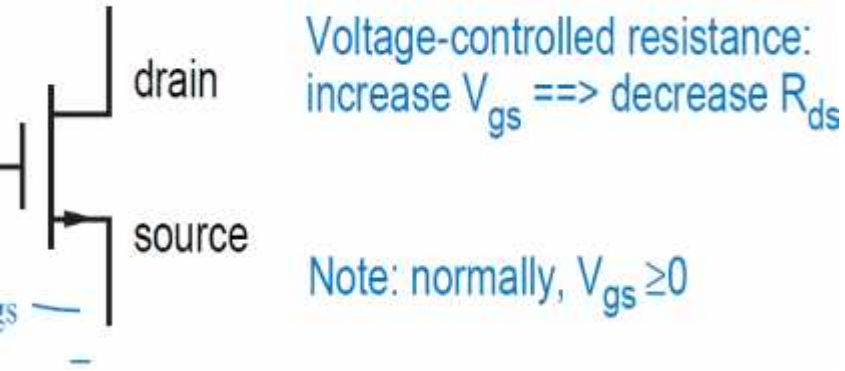
Family

MOS (**C**omplementary **M**etal-**O**xide-**S**emiconductor) technology is used predominantly to
ital circuitry.

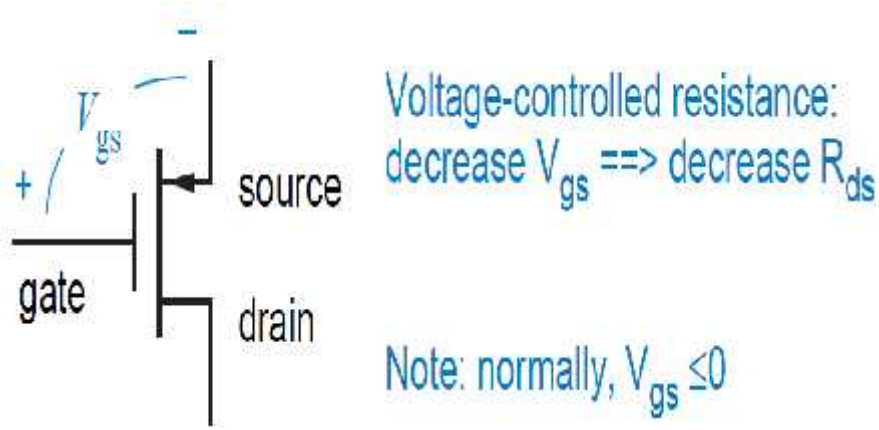
e fundamental building blocks of CMOS circuits are **P-type** and **N-type** MOS transistors.

MOS transistor can be modeled as a 3-terminal device that acts like a voltage- con
sistance. There are two types of MOS transisotrs as shown in below

e MOS transistors



P-type MOS transistors



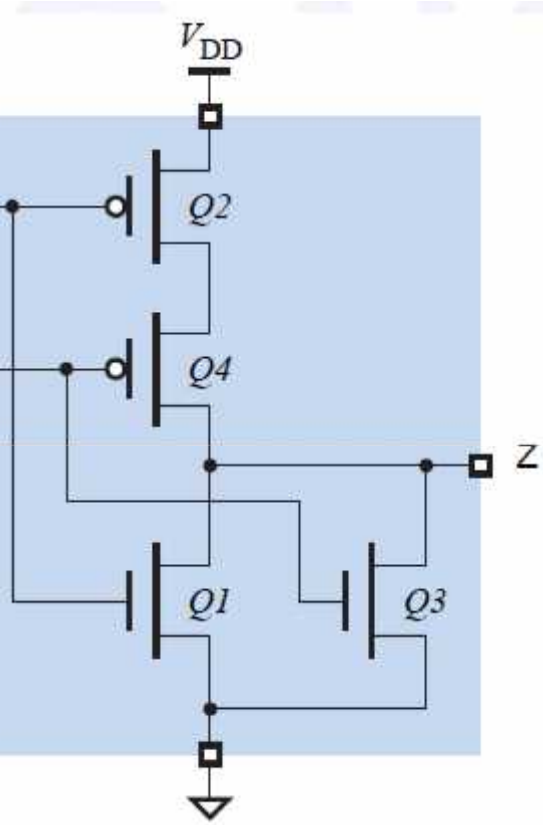
Vin

The MOS tran
a voltage-con
resistance

Far

NAND and NOR Gates

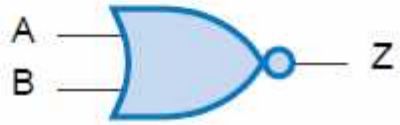
NOR GATE



(b)

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

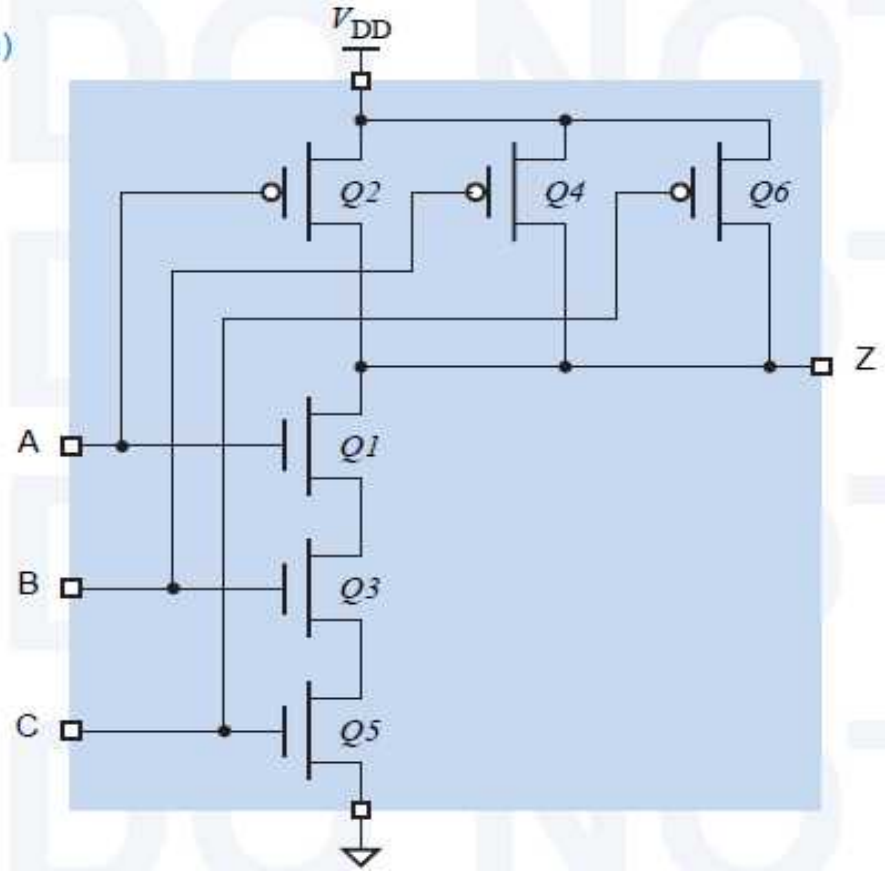
(c)



OS 2-input
R gate:
circuit diagram;
function table;
ogic symbol.

UT CMOS NAND GATE

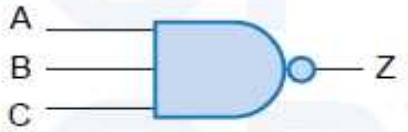
(a)



(b)

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

(c)



CMOS 3-input NAND gate: (a) circuit diagram; (b) function table; (c) logic symbol.

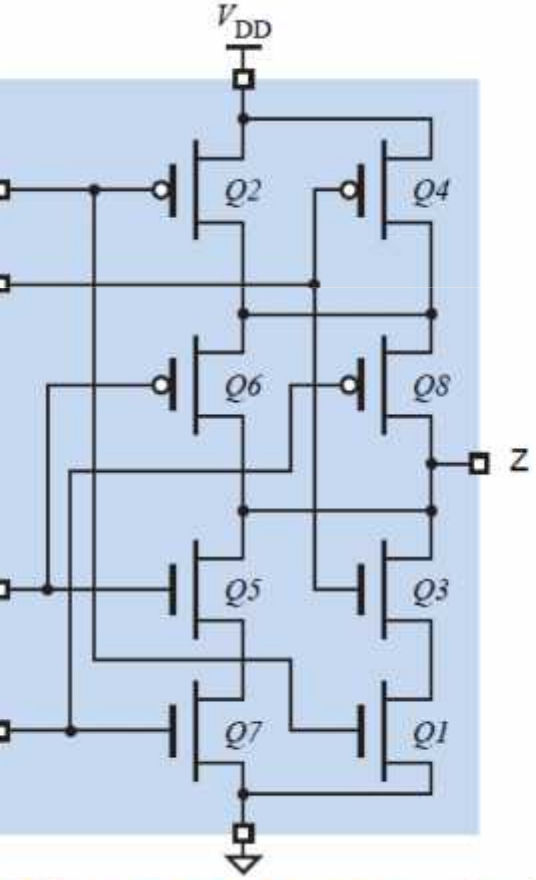
AND-OR-INVERT and OR-AND-INVERT Gates

Far

circuits can perform two levels of logic with just a single “level” of transistors.

AND-OR-INVERT (AOI)

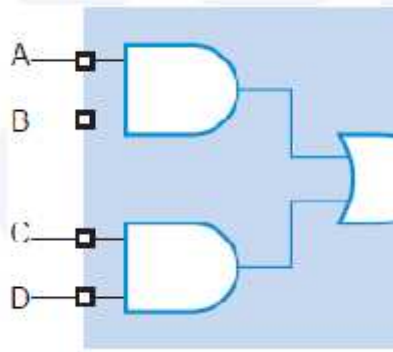
$$Z = \overline{A.B + C.D}$$



(b)

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

OS AND-OR-INVERT gate: (a) circuit diagram; (b) function tab



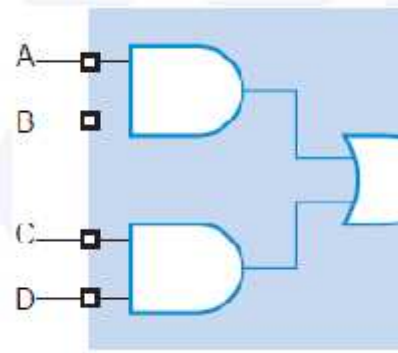
Logic Diagram

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

Far

circuits can perform two levels of logic with just a single “level” of transistors.

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



Logic Diagram

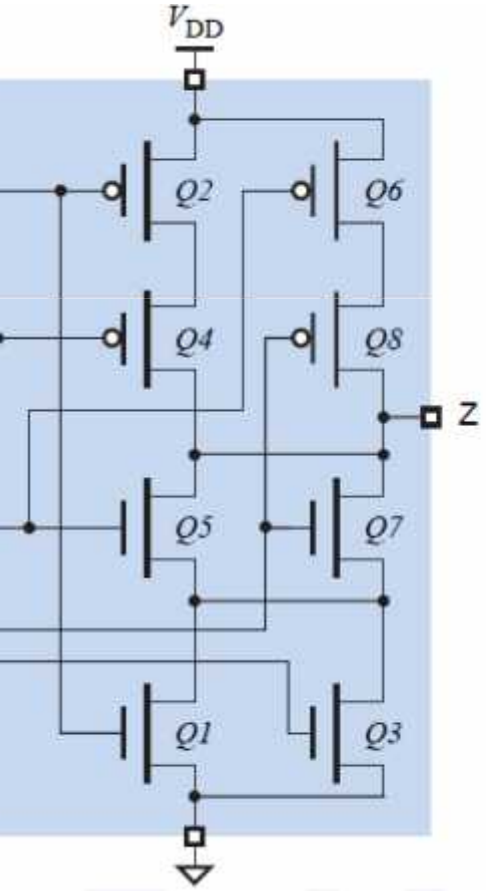
AND-OR-INVERT and OR-AND-INVERT Gates

Far

circuits can perform two levels of logic with just a single “level” of transistors.

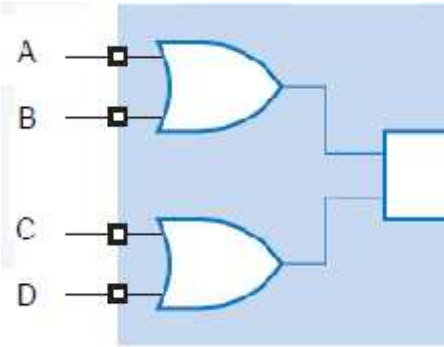
OR-AND-INVERT (OAI)

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



(b)

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



Logic Diagram

R-AND-INVERT gate: (a) circuit diagram; (b) function table.

Far

cteristics of Logic Families:

gic voltage levels

noise margins

ain and Fanout

eed

wer Consumption

ise

ctrostatic discharge

Far

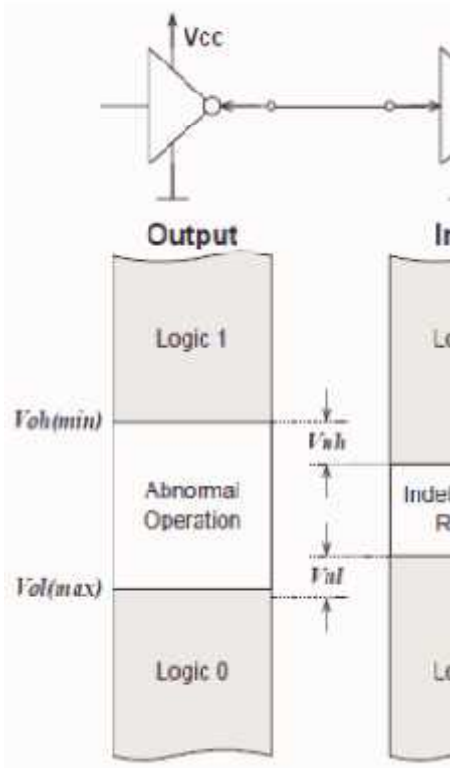
gic Levels and Noise Margins

MOS devices operating under normal conditions are guaranteed to produce output voltag

thin well-defined LOW and HIGH ranges. And they recognize LOW and HIGH input voltag

er somewhat wider ranges.

ere are four logic levels in the logic families.



The **minimum output voltage** in the **HIGH** state.

The **minimum input voltage** guaranteed to be recognized as a **HIGH**.

The **maximum input voltage** guaranteed to be recognized as a **LOW**.

The **maximum output voltage** in the **LOW** state.

Far

ower-supply voltage *V*CC and ground are often called the ***power supply rails***.

MOS levels are typically a function of the power-supply rails:

*V*CC – 0.1 V

Hmin 70% of *V*CC

30% of *V*CC

OLmax ground + 0.1 V

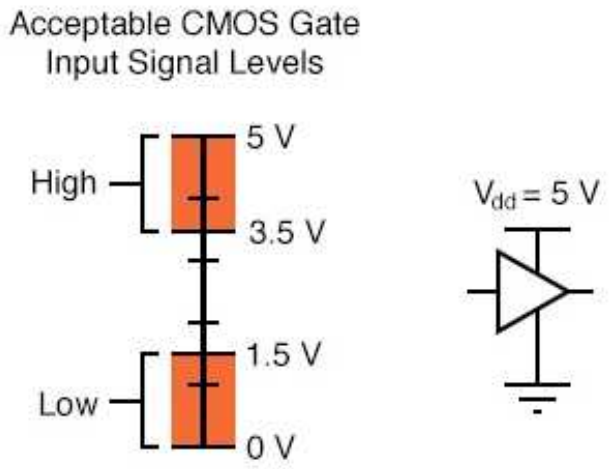
for example Vcc=5V then the CMOS logic levels are

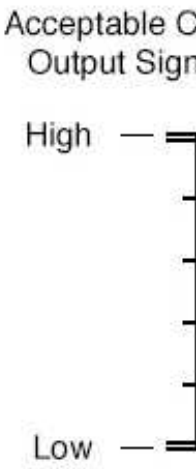
V_{OHmin} =4.9V

V_{IHmin} =3.5V

V_{ILmax} =1.5V

V_{OLmax} =0.1V



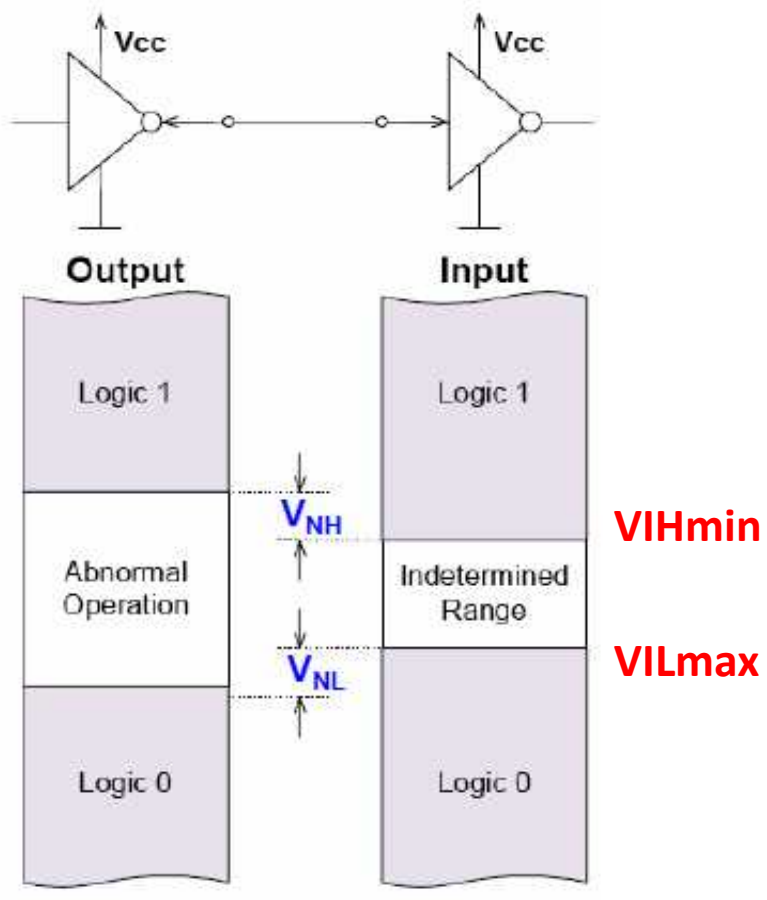


Steady-State Electrical Behavior

Margins

e circuit's ability to tolerate noise signals is referred to as the **noise immunity**, a quar

easure of noise immunity is called **noise margin**.



HI state noise margin:

$$V_{NH} = V_{OH}(\min) - V_{IH}(\min)$$

LO state noise margin:

$$V_{NL} = V_{IL}(\max) - V_{OL}(\max)$$

Noise margin:

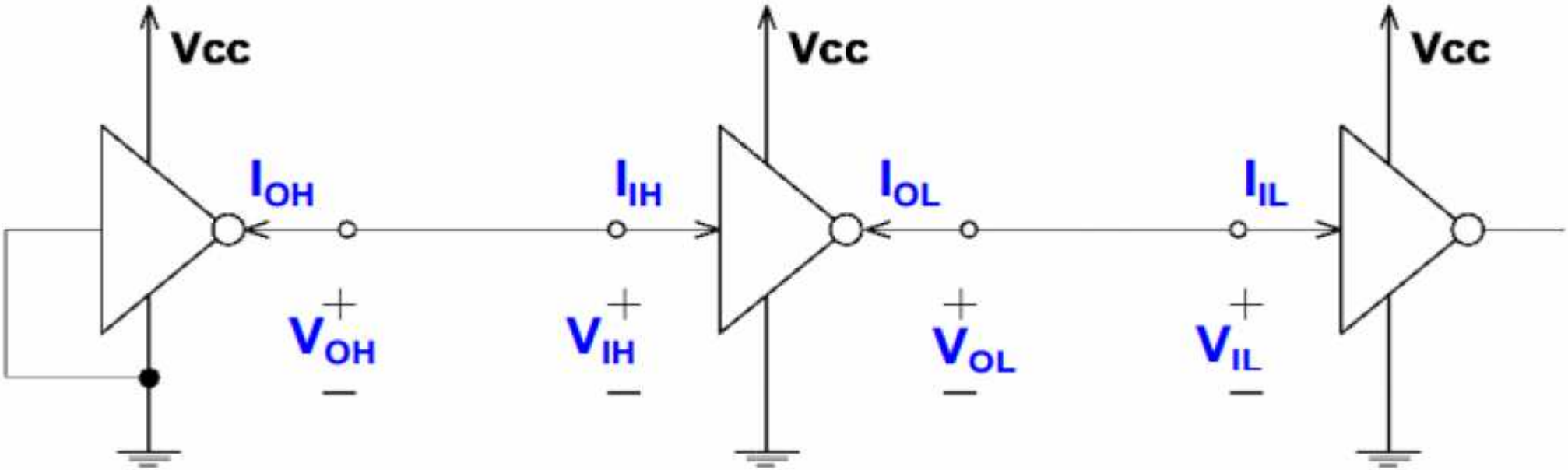
$$V_N = \min(V_{NH}, V_{NL})$$

Fan

$_{OH}$ – Current flowing into an output in the logical “1” state under specified load onditions

$_{OL}$ – Current flowing into an output in the logical “0” state under specified load onditions

$_{H}$ – Current flowing into an input when a specified HI level is applied to that nput I_{IL} – Current flowing into an input when a specified LO level is applied to hat input



Far

Steady-State Electrical Behavior

cuit Behavior with Resistive Loads

CMOS gate inputs have very high impedance and consume very little current from the that drive them.

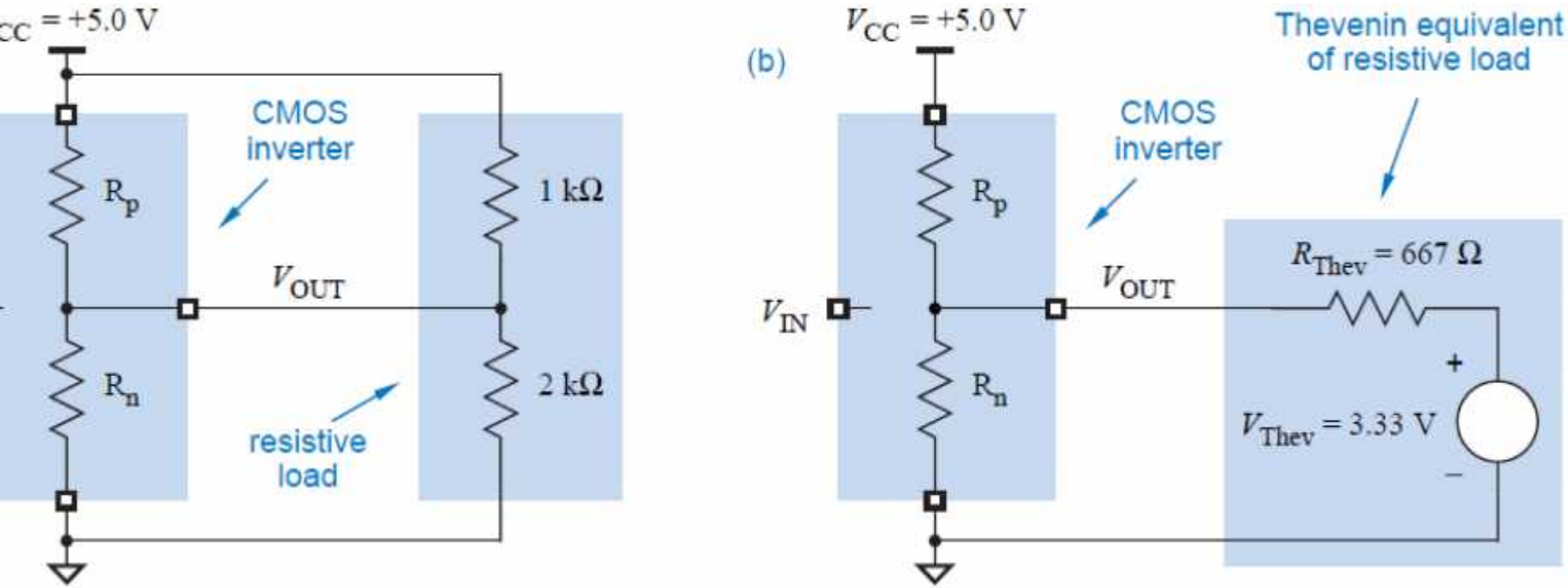
The other devices such as discrete resistors, LEDs and relays are connected to a CMOS we say that it is a **resistive loads or a DC load**.

When the output of a CMOS circuit is connected to a resistive load, the output behavic nearly as ideal.

Thus, in the **LOW state**, the output voltage may be somewhat **higher than 0.1 V**, and **HIGH state** it may be **lower than 4.4 V**.

Far

tive model of a CMOS inverter with a resistive load



(a) Showing actual load circuit (b) Thevenin equivalent model

normal operation, one resistance is high (>1 M ohms) and the other is low (perhap
ms), depending on whether the input voltage is HIGH or LOW.

load in this circuit consists of two resistors attached to the supply rails; a real circui
e any resistor values.

any case, a resistive load, consisting only of resistors and voltage sources, can alwa
deled by a Thévenin equivalent network

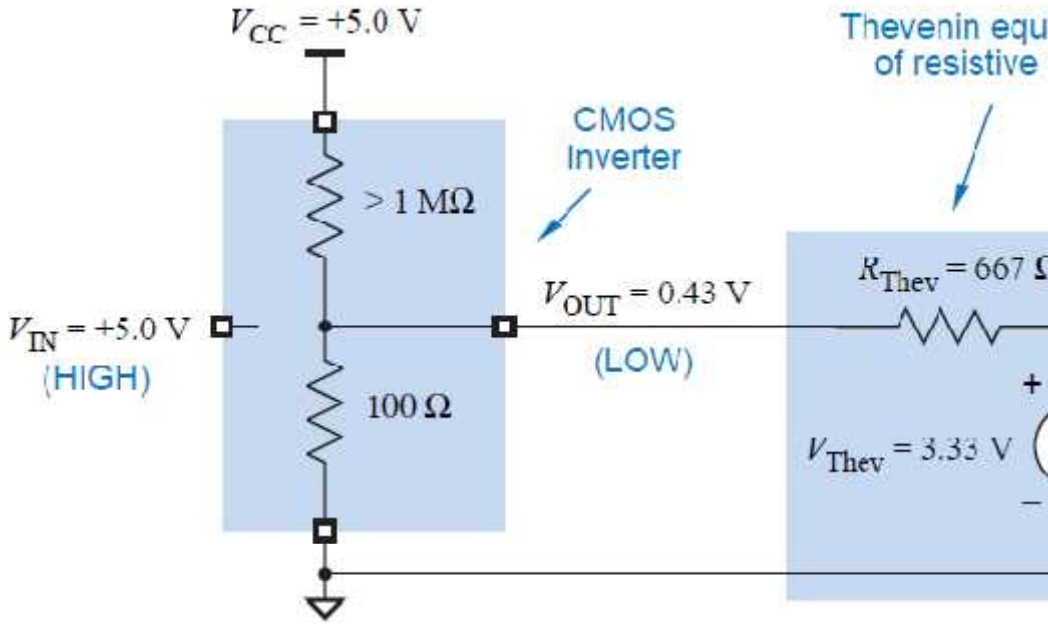
Far

Resistive model for CMOS LOW output with resistive load

en the CMOS inverter has a HIGH input,
output should be LOW; Since PMOS is off
h resistance) and NMOS is ON (**Less On**
stance)

resulting output voltage can be calculated
ollows $V_{OUT} = 3.33\ \text{V} \cdot [100/(100 + 667)]$

$= 0.43\ \text{V}$

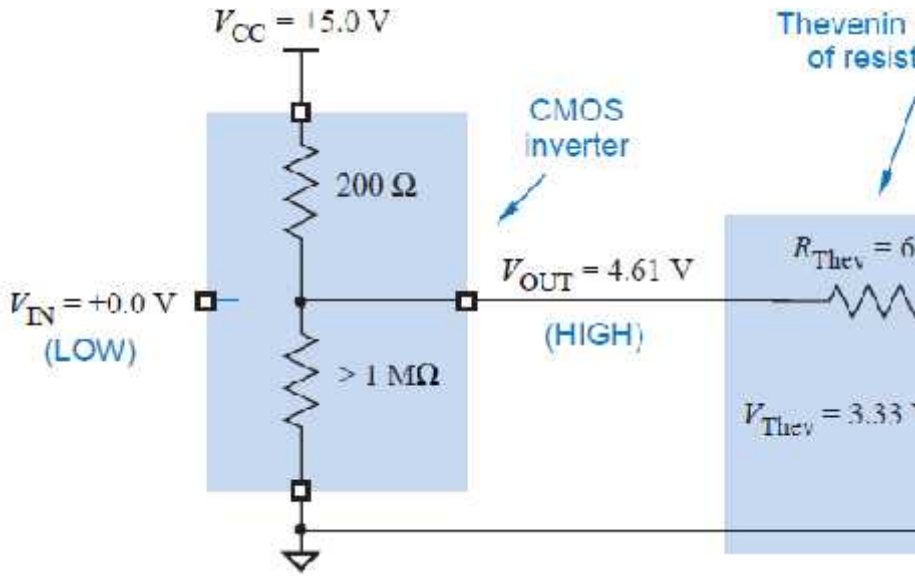


stive model for CMOS HIGH output with resistive load

arly, when the inverter has a LOW input, the output
ld be HIGH.
actual output voltage can be predicted with the model in
re.Assume that the PMOS "on" resistance is 200 Ω , the
lting output voltage can be calculated as follows:

$UT = 3.33\ \text{V} + (5\ \text{V} - 3.33\ \text{V}) \cdot [667/(200 + 667)]$

$= 4.61\ \text{V}$



Fan

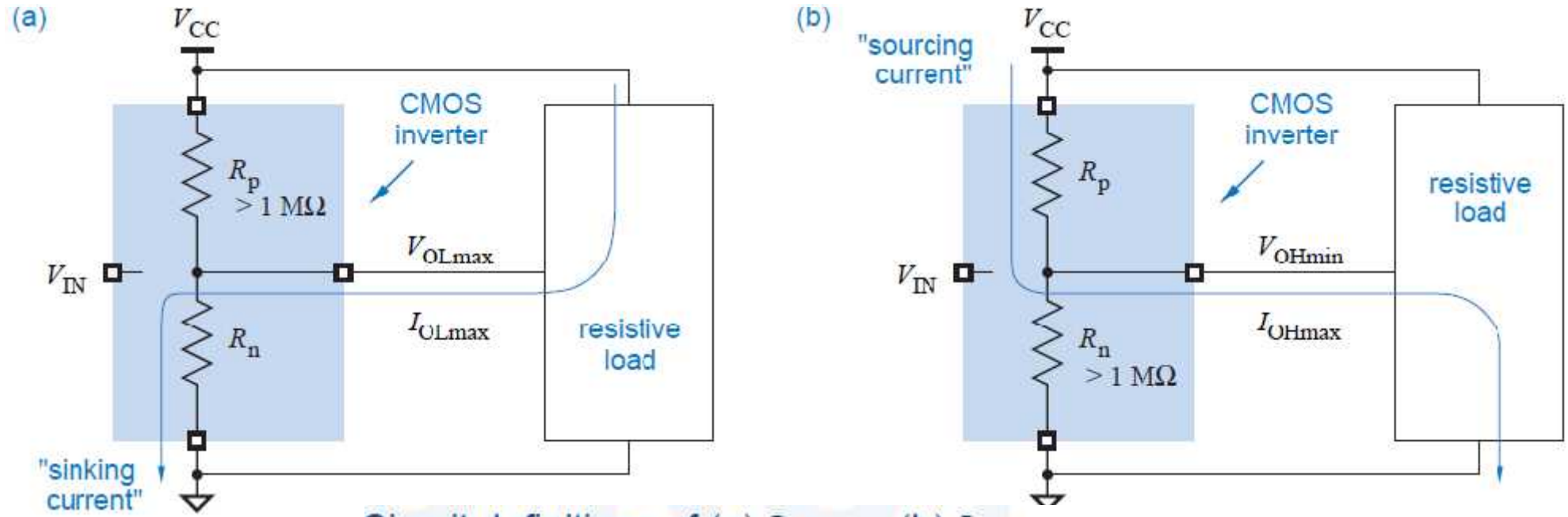
nufacturers specify a maximum load for the output in each state (HIGH or LOW), and guar

case output voltage for that load.

e load is specified in terms of current:

I_{OLmax} : The maximum current that the output can sink in the LOW state while still maintaining an output voltage no greater than **V_{OLmax}**.

I_{OHmax} : The maximum current that the output can source in the HIGH state while still maintaining an output voltage no less than **V_{OHmin}**.



Circuit definitions of (a) I_{OLmax}; (b) I_{OHmax}.

g Current and Sourcing Current

Far

g Current:

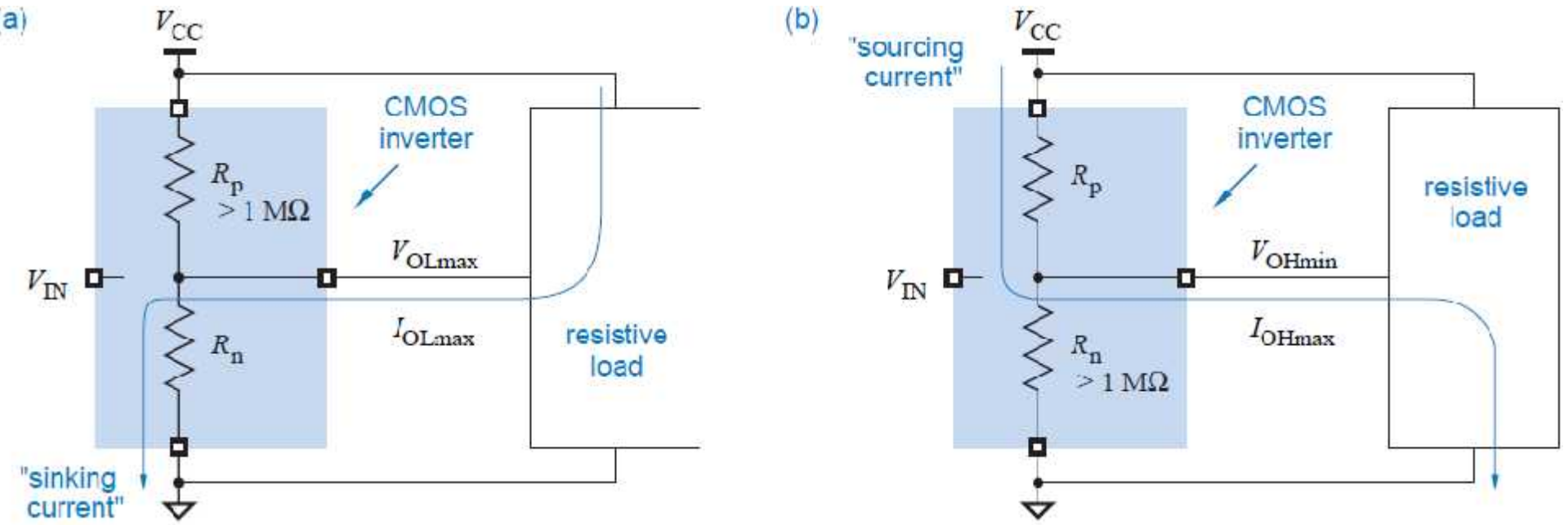
ce output is said to sink current when current flows from the power supply, through th

rough the device output to ground.

ng Current:

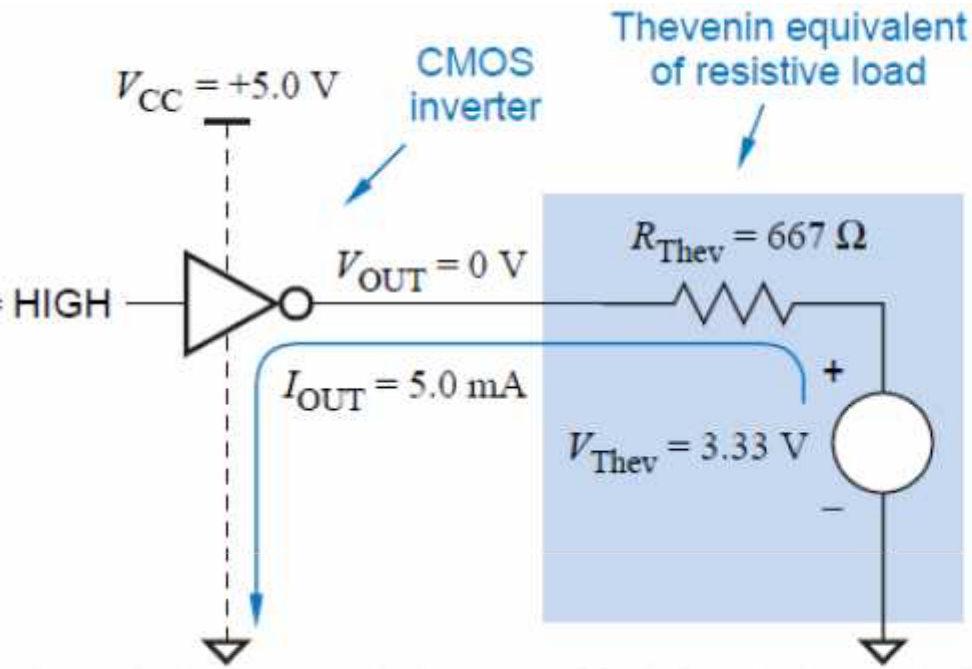
utput is said to source current when current flows from the power supply, out of the

t, and through the load to ground.

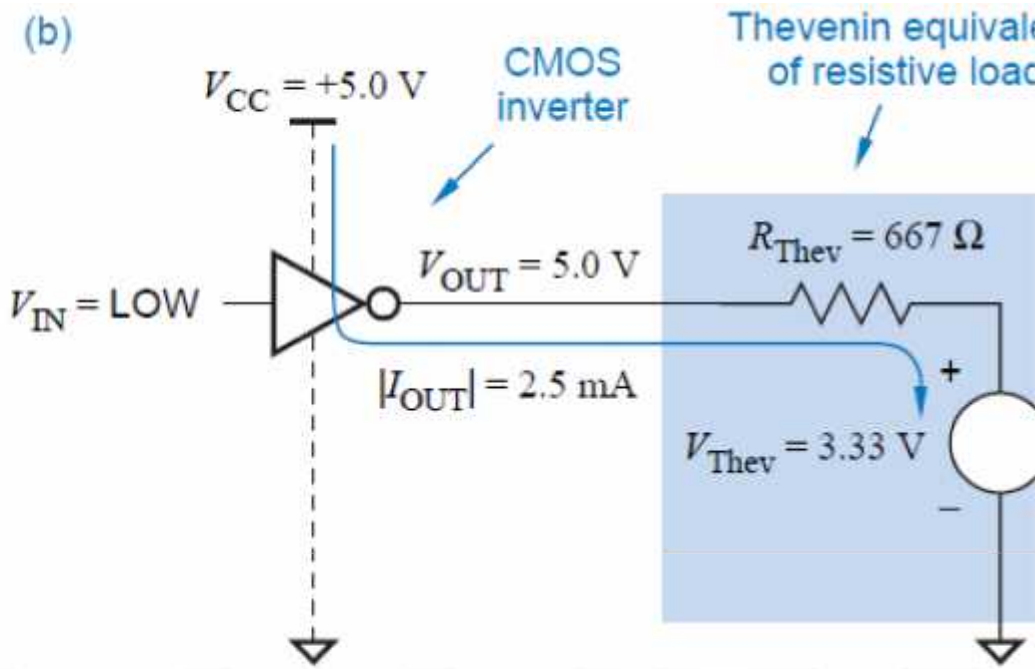


ation of Sinking Current and Sourcing Current

Far



(a) Output Low



(b) Output High

nk current = 3.33/667=5 mA

Source current = 1.67/667=2.5 m

uit Behaviour with Non-Ideal Inputs

Far

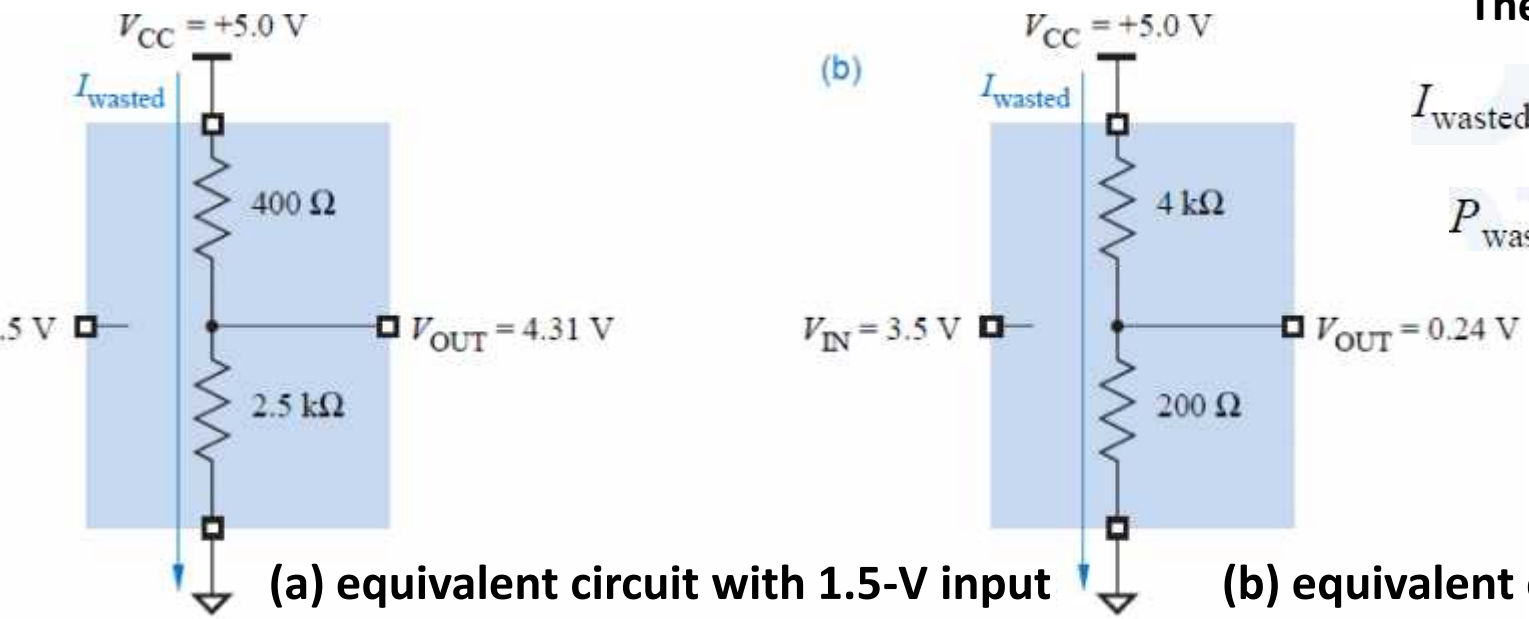
far, we have assumed that the HIGH and LOW inputs to a CMOS circuit are ideal voltag
se to the power supply rails (5V and 0V).
wever, the behavior of a real CMOS inverter circuit depends on the input voltage as we
characteristics of the load.
he input voltage is **not close to the power-supply rail**, then the "on" transistor may not
" and its resistance may increase.
ewise, the "off" transistor may not be fully "off' and its resistance may be quite a bit le
e megohm.
ese two effects combine to move the output voltage away from the power supply rail.

inverter with nonideal input voltages

The current flow with the 1.5V In

$$I_{\text{wasted}} = 5.0\ \text{V}/400\ \Omega + 2.5\ \text{k}\Omega =$$

$$P_{\text{wasted}} = 5.0\ \text{V} \cdot I_{\text{wasted}} = 8.62$$

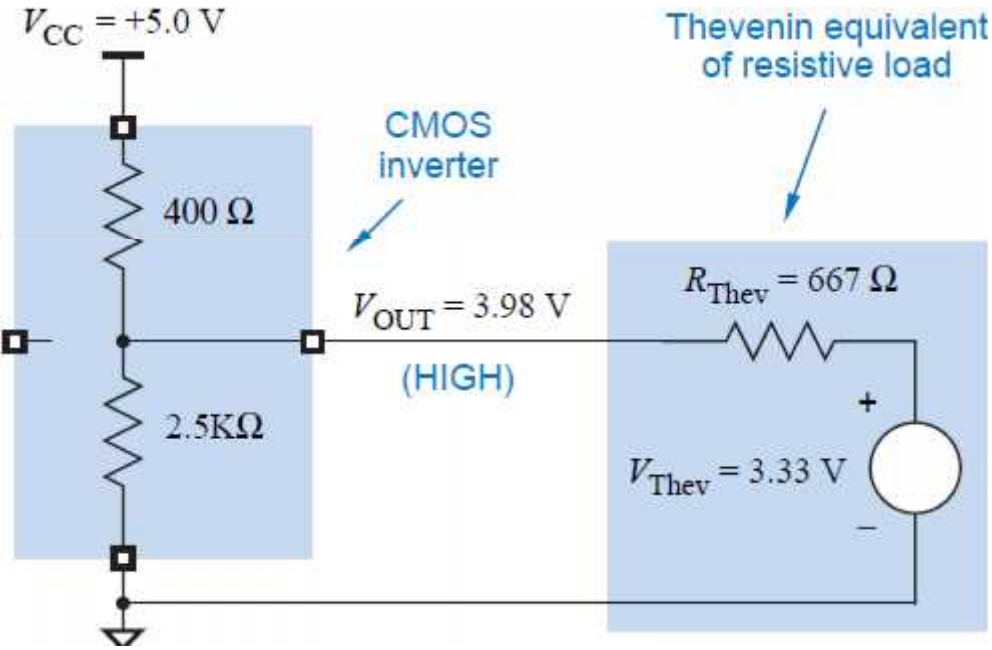


(a) equivalent circuit with 1.5-V input **(b) equivalent circuit with 3.5-V input**

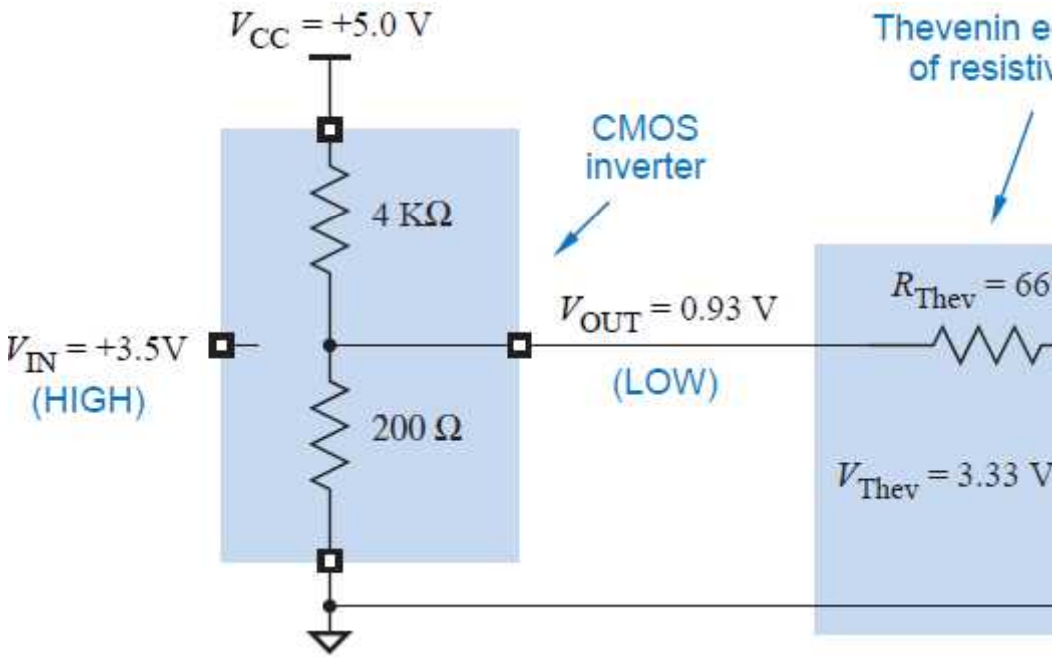
uit Behaviour with Non-Ideal Inputs

Far

OS inverter with load and nonideal
-V input



CMOS inverter with load and noni
3.5-V input



a **1.5-V input**, the output at **3.98 V** is still within the valid range for a HIGH signal, but it is far f
of **5.0 V**.

arly, with a **3.5-V input**, the LOW output is **0.93 V, not 0 V.**

Fan

out

e fanout of a logic gate is the number of inputs that the gate can drive without excee

rst-case loading specifications.

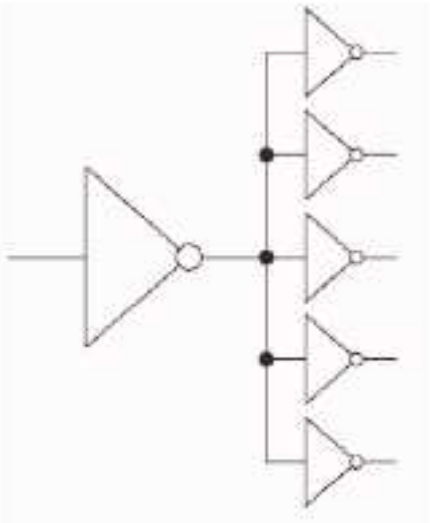
e fanout depends not only on the characteristics of the output, but also on the inputs t

ving.

nout must be examined for both possible output states, HIGH and LOW.

te that the HIGH-state and LOW-state fanouts of a gate are not necessarily equal.

general, the **overall fanout** of a gate is **the minimum of its HIGH-state and LOW-state fan**



$$\textbf{fanout} = \min\left(\frac{I_{OH}}{I_{IH}}, \frac{I_{OL}}{I_{IL}}\right)$$

Fan

cts of Loading

g an output beyond its rated fanout has several effects:

he LOW state, the output voltage (V_{OL}) may increase beyond V_{OLmax}.

he HIGH state, the output voltage (V_{OH}) may fall below V_{OHmin}.

pagation delay to the output may increase beyond specifications.

tput rise and fall times may increase beyond their specifications.

e operating temperature of the device may increase, thereby reducing the reliability vice and eventually causing device failure.

Far

used Inputs

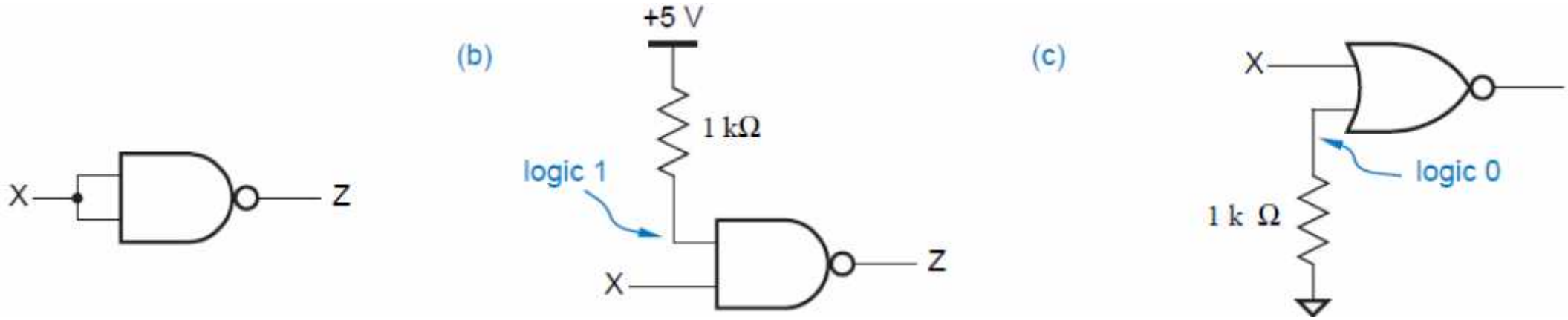
used CMOS inputs should never be left unconnected (or floating).

unused **AND or NAND** input should be tied to **logic 1**, as in (b), and

unused **OR or NOR** input should be tied to logic 0, as in (c).

high-speed circuit design, it's usually better to use method (b) or (c) rather than (a)

reases the capacitive load on the driving signal and may slow things down



Unused inputs: (a) tied to another input; (b) NAND pulled up; (c) NOR pulled down.

Dynamic Electrical Behaviour

Far

he CMOS Dynamic electrical behaviour we will study the followung parameters

- **Transition Time**
- **Propagation Delay**
- **Power Consumption**

h the **speed** and the **power consumption** of a CMOS device depend to a large extent o

amic characteristics of the device and its load, that is, what happens when the output c

ween states.

eed depends on two characteristics, **transition time** and **propagation delay**

nsition Time

mount of time that the output of a logic circuit takes to change from one state to an

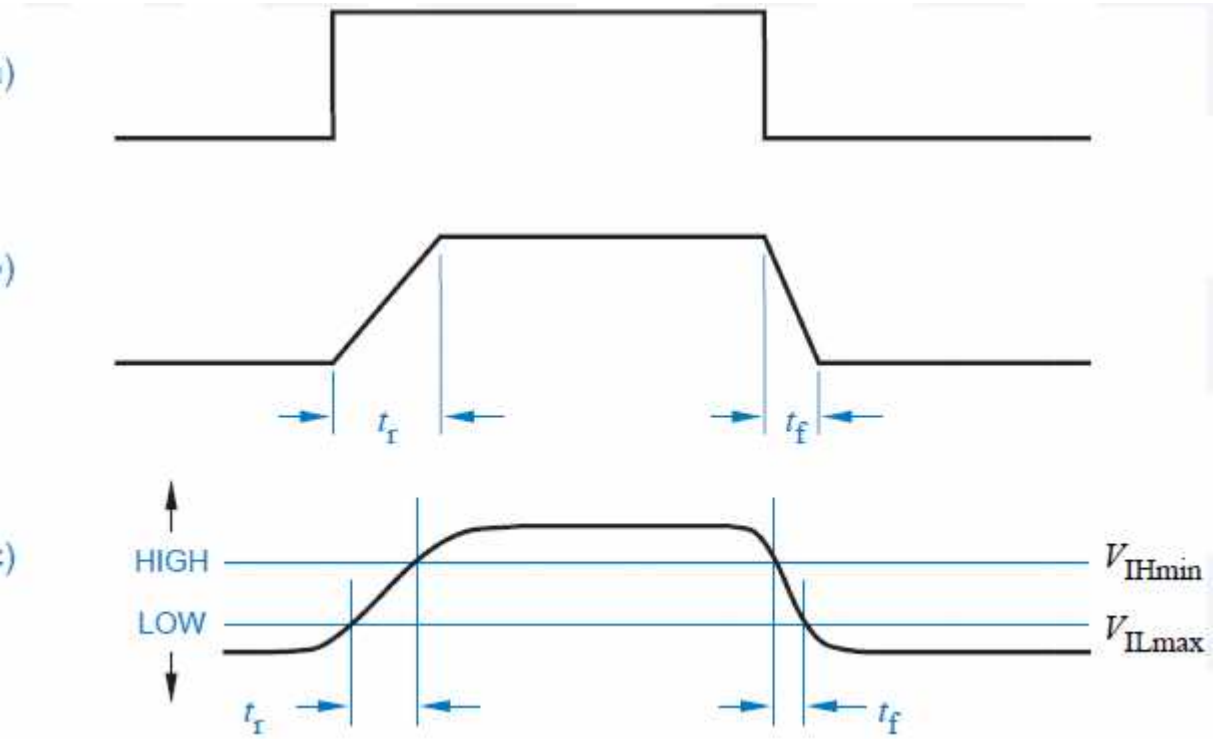
the transition time.

Dynamic Electrical Behaviour
nsition Time

ansition time will depends on the two parameters

se Time (T_r): It is the time taken by the output to change from **LOW to HIGH**

ll Time (T_f) : It is the time taken by the output to change from **HIGH to LOW**



(a) Ideal case of zero-time switching

(b) Realistic

(c) Actual Timing

Dynamic Electrical Behaviour

Far

se and fall times of a CMOS output depend mainly on two factors, the "**on" transistor res**
e **load capacitance**.

arge capacitance increases transition times; since this is undesirable, the stray capacit

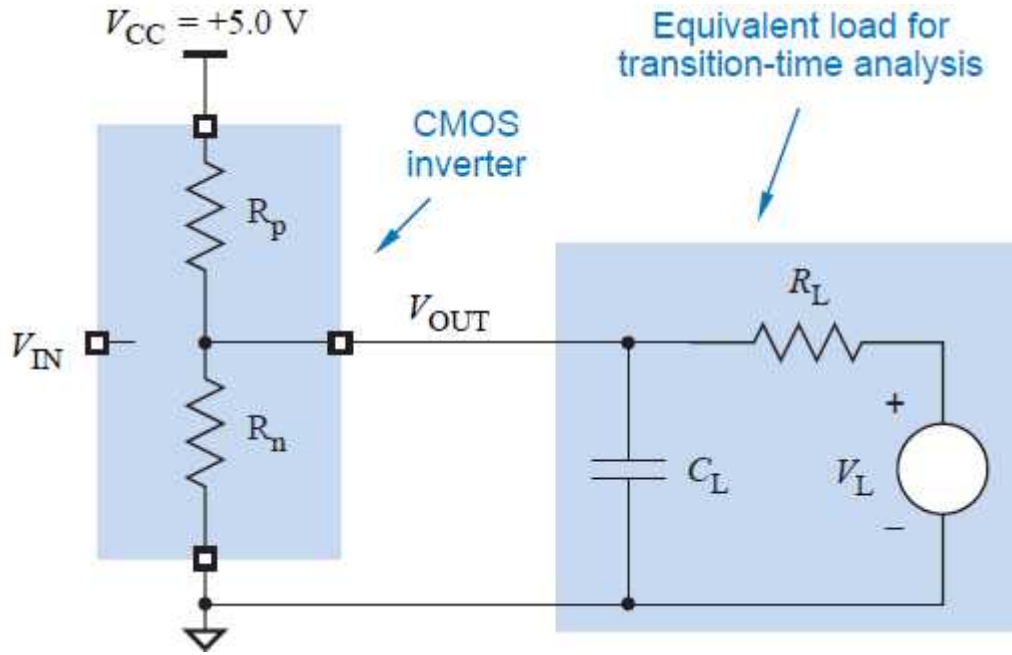
sent in every circuit due to following sources

The outout circuits , output transistors, internal wiring and packaging have some capa

associated with them. This is the order of 2-10 pF

The wiring connections between output and other input pads adds capacitance of the c

1 pF.



Equivalent circuit for analysing transition times of CMOS output

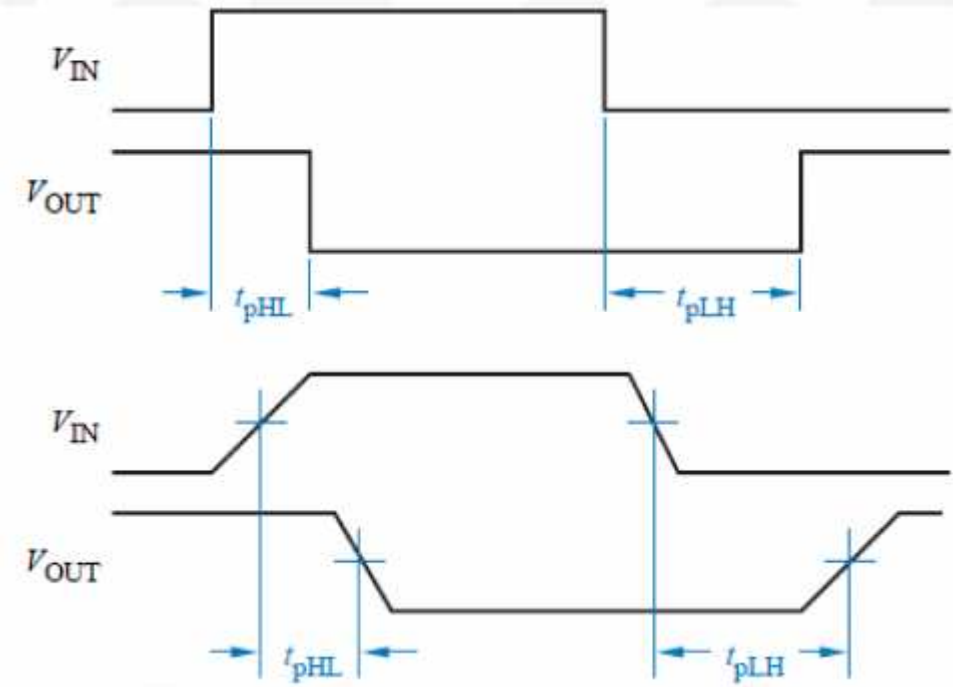
Dynamic Electrical Behaviour

pagation Delay

Far

ropagation delay (tp) of a signal path is the amount of time that it takes for a change in th

to produce a change in the output signal.



pagation delays
a CMOS inverter:
ignoring rise and
times; (b) measured at
points of transitions.

The Propagation delay is determined by two fa

1. t_{PHL} :The time between an input change corresponding output change when the o changing from HIGH to LOW.

2.t_{pLH} :The time between an input change corresponding output change when the o changing from LOW to HIGH.

- The values of propagation times (t_{PLH},t_{PHL}) ma same and both will vary depending on the c load.
- When both are equal, the larger value is co as a propagation delay time for logic gate.

Dynamic Electrical Behaviour

wer Consumption

ower consumption in the gates are two types

Static Power dissipation

Dynamic Powr dissipation

e power consumption of a CMOS circuit whose output is **not changing** is called **static**

sipation or quiescent power dissipation.

e power consumption of a CMOS circuit **during transition** is known as **dynamic**

sipation.

ower consumption during the transition is given by $P_T = C_{PD} \cdot V_{CC}^2 \cdot f$

e PT : The circuit's internal power dissipation due to output transitions.

VCC : The power supply voltage.

f : The transition frequency of the output signal. This specifies the number of power-consuming output transitions per second.

C_{PD} : The power dissipation capacitance, it specified by the manufacturer.

Dynamic Electrical Behaviour

wer Consumption

ower consumption in the gates are two types

Static Power dissipation

Dynamic Powr dissipation

e power consumption of a CMOS circuit whose output is **not changing** is called **static**

sipation or quiescent power dissipation.

e power consumption of a CMOS circuit **during transition** is known as **dynamic**

sipation.

ower consumption during the transition is given by $P_T = C_{PD} \cdot V_{CC}^2 \cdot f$

e PT : The circuit's internal power dissipation due to output transitions.

VCC : The power supply voltage.

f : The transition frequency of the output signal. This specifies the number of power-consuming output transitions per second.

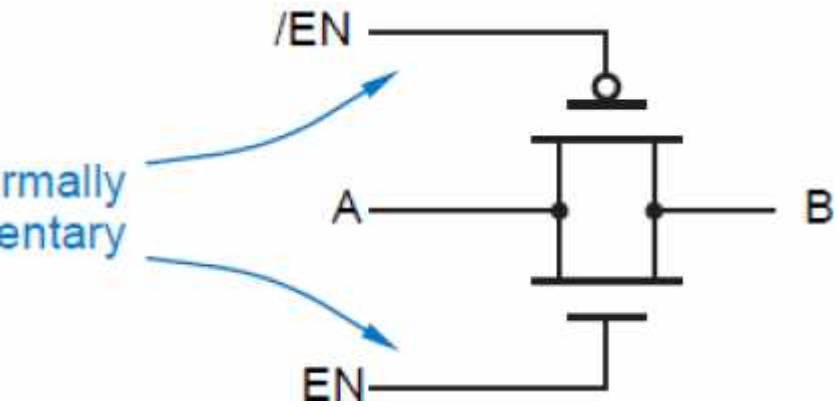
C_{PD} : The power dissipation capacitance, it specified by the manufacturer.

mission gate

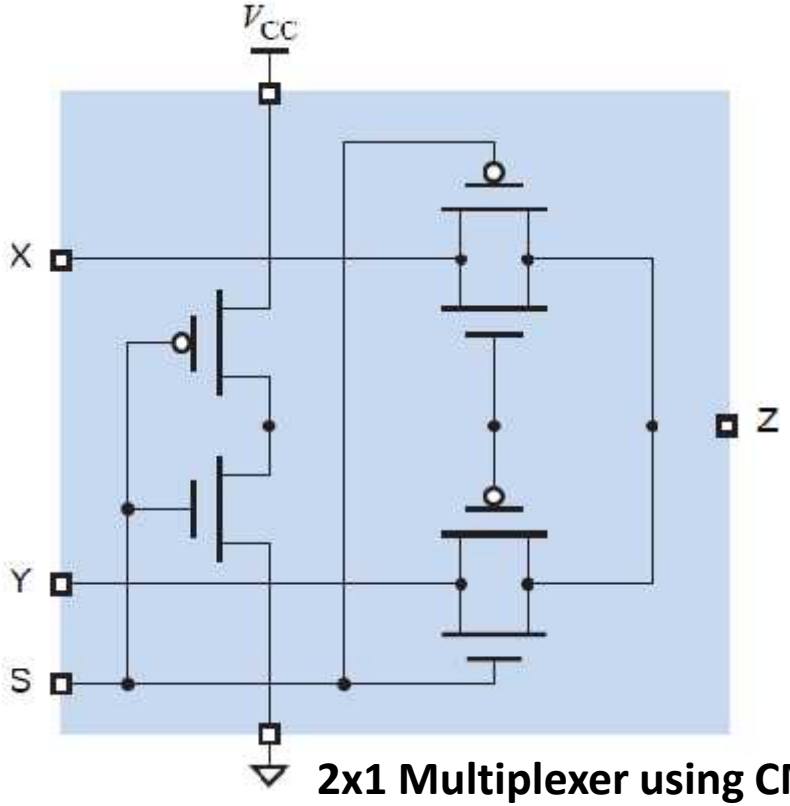
MOS and NMOS transistor pair can be connected together to form a logic-controlled swit
uit is called a CMOS transmission gate.

ransmission gate is operated so that its input signals EN and EN-bar are always at opposite
en EN is HIGH and EN-bar is LOW, there is a low impedance connection between points A
en EN is LOW and EN-bar is HIGH, points A and B are disconnected.

e propagation delay of the transmission gate is very low. Because of short delays and con
plicity, transmission gates are often used internally in larger scale CMOS devices s
ltiplexers and flip-flops.



MOS Transmission Gate



- When S is LOW, "input" is conn the Z "output";
- when S is HIGH connected to Z.

2x1 Multiplexer using CMOS transmission gat

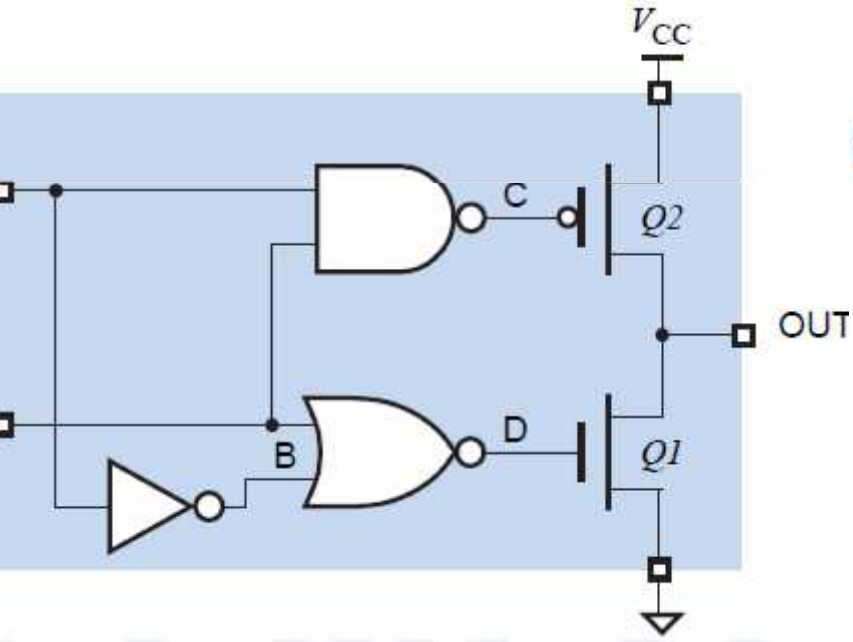
State Outputs

Far

output with **three possible states** is called a three-state output or, sometimes, a tri-state

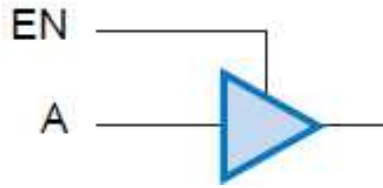
ee-state devices have an extra input, usually called “output enable” or “output disab

cing the device’s output(s) in the **high-impedance state (Hi-Z)**.



(b)

EN	A	B	C	D	Q1	Q2	OUT
L	L	H	H	L	off	off	Hi-Z
L	H	H	H	L	off	off	Hi-Z
H	L	L	H	H	on	off	L
H	H	L	L	L	off	on	H



CMOS Three State Buffer **Functional Table** **Symbol**

Drain Outputs

Far

e PMOS transistors in CMOS output structures are said to provide **active pull-up**, sin

ively pull up the output voltage on a LOW-to-HIGH transition.

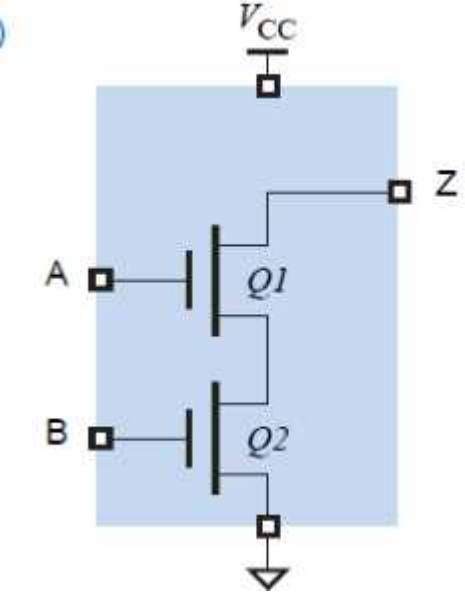
ese transistors are omitted in gates with **open-drain outputs**.

e drain of the topmost NMOS transistor is left unconnected internally, so if the output is n

"open.

open-drain output requires an external pull-up resistor to provide passive pull-up

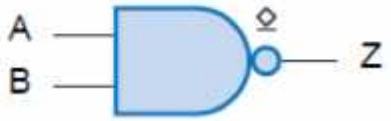
GH level.



(b)

A	B	Q1	Q2	Z
L	L	off	off	open
L	H	off	on	open
H	L	on	off	open
H	H	on	on	L

(c)



Open-drain CMOS NAND gate: (a) circuit diagram; (b) function table; (c) logic symbol

Logic

Far

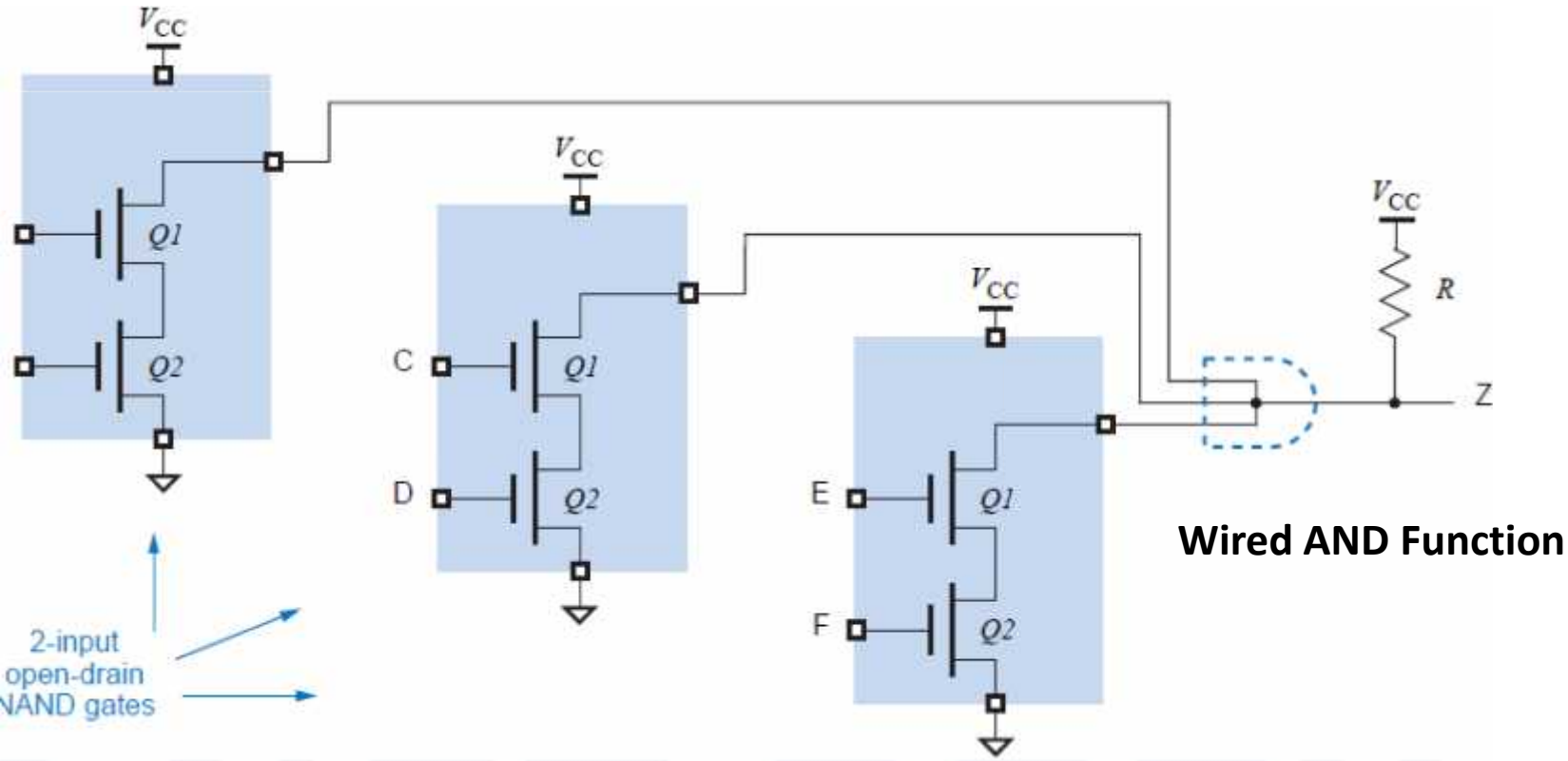
he outputs of several open-drain gates are tied together with a single pull-up resistor, the

ic is performed.

AND function is obtained, since the wired output is HIGH if and only if all of the individu

puts are HIGH.

example, a three-input wired AND function is shown in Figure.



Wired AND Function

Logic Familes

Far

e CMOS Logic Families are

- 4000 serires CMOS
- HC (High Speed CMOS)
- HCT (High Speed CMOS, TTL compatible)
- VHC (Very High Speed CMOS)
- VHCT (Very High Speed CMOS, TTL compatible)
- FCT (Fast CMOS, TTL compatible)
- FCT-T (Fast CMOS, TTL compatible with V_{OH})

0 serires CMOS

e first commercially successful CMOS family was **4000-series** CMOS.

00-series circuits offered the benefit of low power dissipation, they were fairly slow and w

y to interface with the most popular logic family of the time, bipolar TTL.

Logic Familes

CMOS devices that have part numbers of the form **74FAMnn**

where “FAM” is an alphabetic family mnemonic and nn is a numeric function designator.

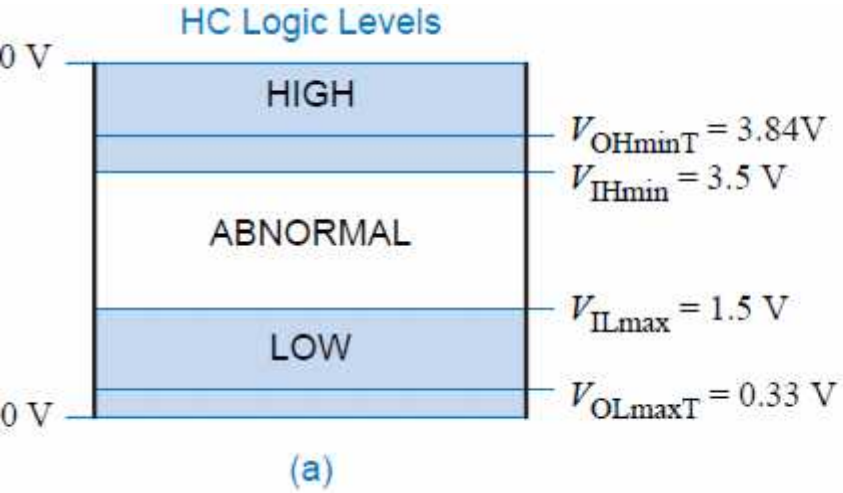
prefix “74” is simply a number that was ued by popular supplier of TTL devices, ruments.

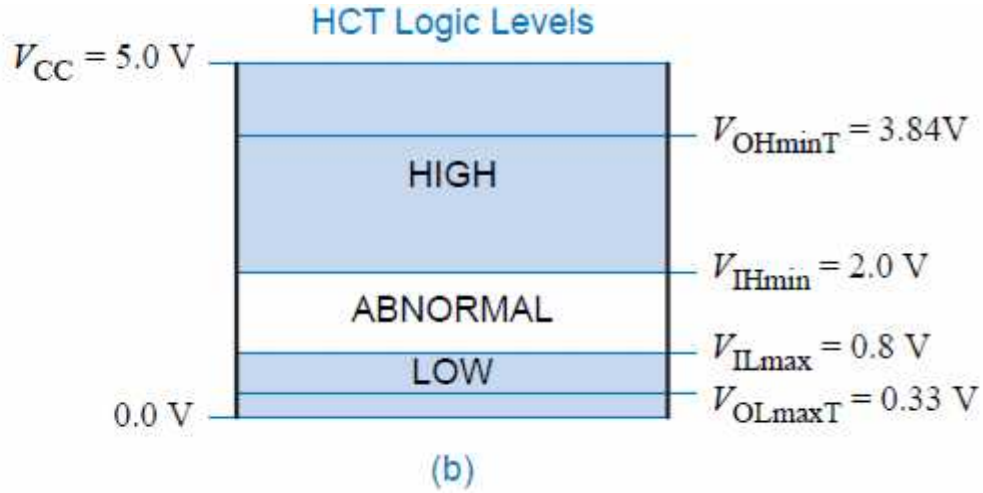
prefix “54” is used for military applications (wider range of temperature and power supp

High Speed CMOS) & HCT (High Speed CMOS, TTL compatible)

first two 74-series CMOS families are HC and HCT.

and HCT both have higher speed and better current sinking and sourcing capability.





Logic Leve
HC and HC

Logic Familes

and VHCT (Vey High Speed CMOS) & HCT (Very High Speed CMOS, TTL compatible)

se families are about twice as fast as HC/HCT while maintaining backwards compatibili

r predecessors.

like HC/HCT, VHC/VHCT outputs have symmetric output drive. That is, an output can

rce equal amounts of current.

and FCT-T

key benefit of the FCT family was its ability to meet or exceed the speed and the outpu

ability of the best TTL families while reducing power consumption and maintaini

npatibility with TTL.

FCT family had the **drawback** of producing a full 5-V CMOS V_{OH}, creating enormous

ipation and circuit noise.

-T is to reduce the HIGH-level output voltage, thereby reducing both power consumpti

tching noise while maintaining the same high operating speed as the original FCT.

Bipolar Logic Family

olar logic families use semiconductor **diodes and bipolar junction transistors** as th
lding blocks of logic circuits.

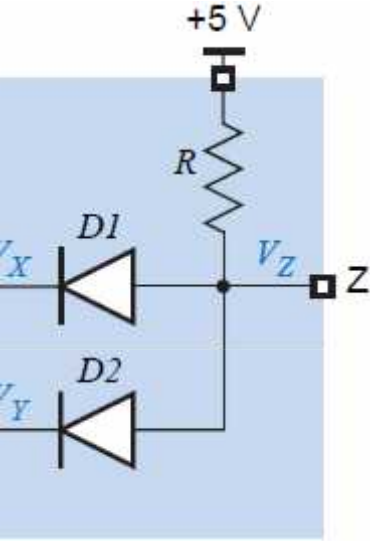
e simplest bipolar logic elements use diodes and resistors to perform logic operations;
ed diode logic. BJT Family can be classified as..

- **Diode logic (DL)**
- **Resistor-Transistor logic (RTL)**
- **Diode-Transistor logic (DTL)**
- **Transistor-Transistor logic (TTL)**
- **Emitter coupled logic (ECL**

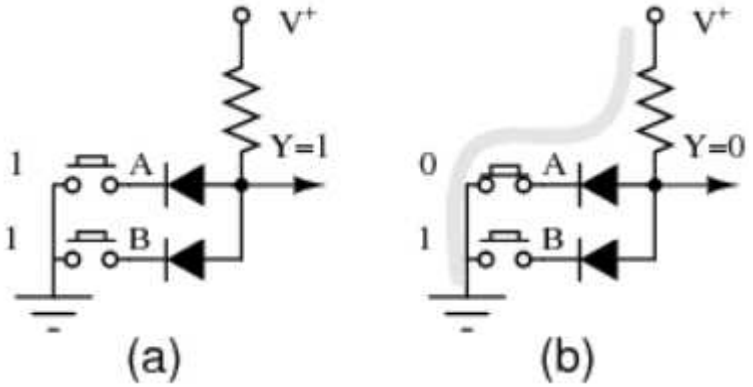
de logic (DL)

de logic family, all the logic is implemented using diodes and resistors.

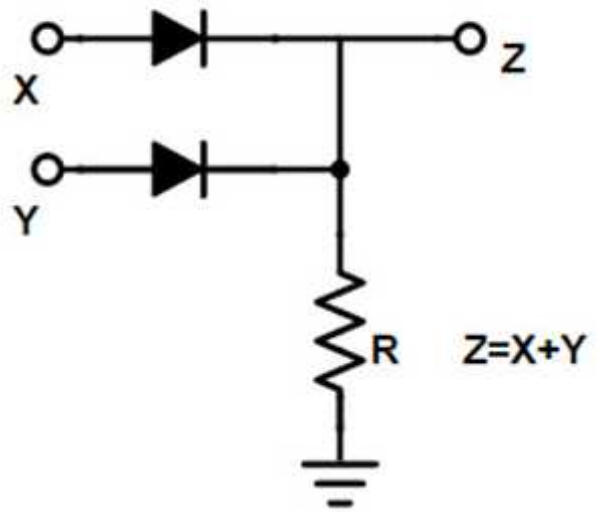
ND GATE



V_X	V_Y	V_Z
w	low	low
w	high	low
gh	low	low
gh	high	high



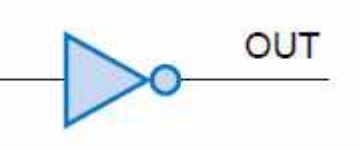
OR GATE

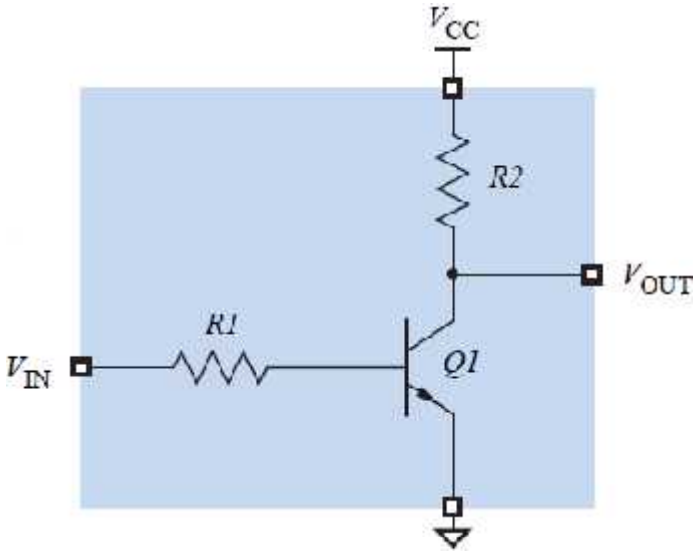


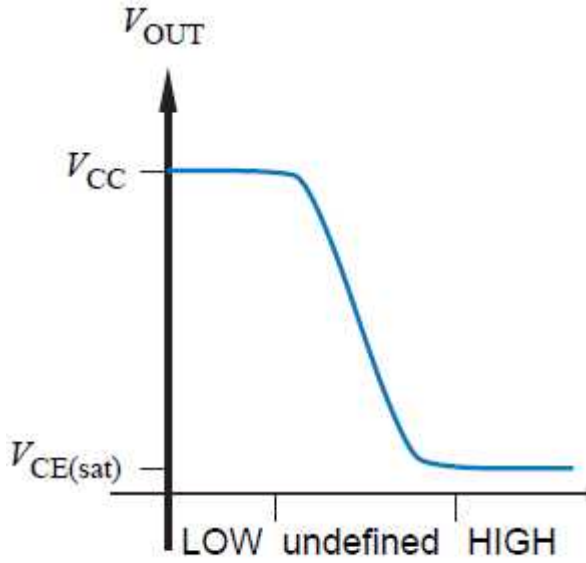
Inputs		Outputs
X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1

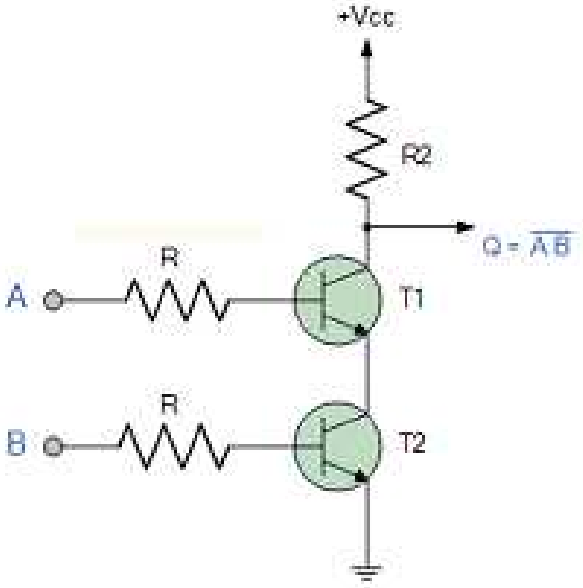
istor Transistor logic (RTL)

ATE



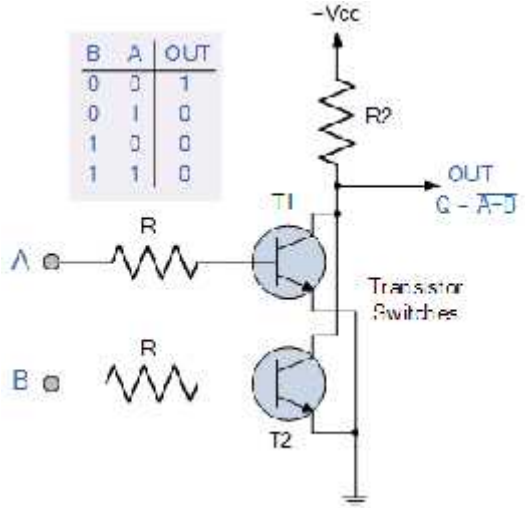






D GATE

B	A	OUT
0	0	1
0	1	0
1	0	0
1	1	0



NOR GATE

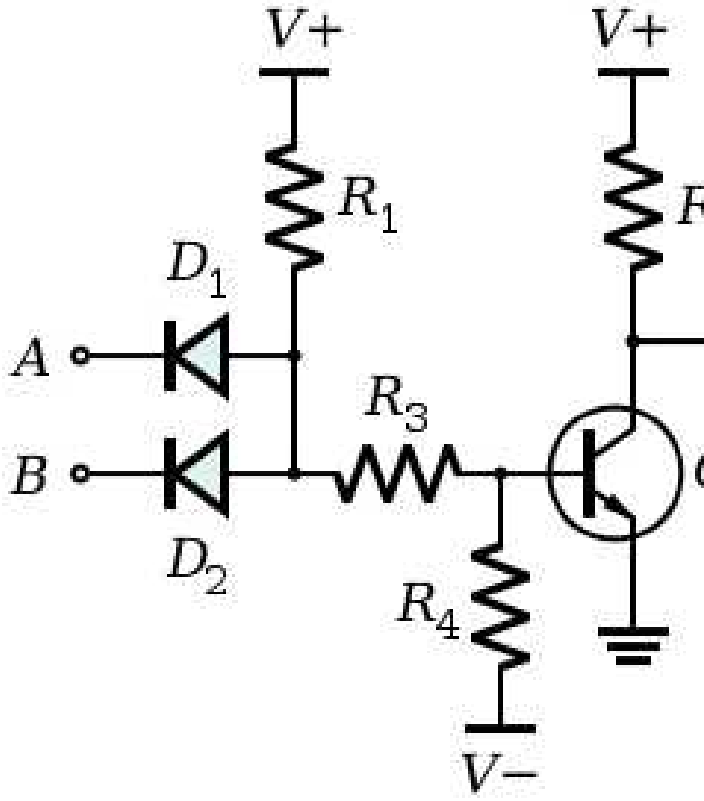
de Transistor logic (DTL)

natic of basic two-input **DTL NAND gate**.

– and V– shift the positive output voltage of the input DL stage below the ground (to cut
stor at low input voltage).

oth inputs A and B are high (logic 1) then the diodes
and D2 are reverse biased. Resistors R1 and R3 will then
pply enough current to turn on Q1. There will be a small
sitive voltage on the base of Q1. **the transistor Q1 is ON**
put Q is LOW.

ither or both inputs are low, then at least one of the
ut diodes conducts and pulls the voltage at the anodes
value less than about 2 volts. R3 and R4 then act
voltage divider that makes Q1's base voltage negative
consequently **turns off Q1**. **The output Q IS HIGH**



DTL NAND Gate

ansistor Transistor logic (TTL)

e most commonly used bipolar logic family is transistor-transistor logic.

e basic building block of this logic family is NAND gate.

ere are various sub-families of this logic family are

- Standard TTL (**74-series TTL**)
- Low Power TTL (**74L-TTL**)
- High-speed TTL ((**74H-TTL)**
- Schottky TTL (**74S-TTL**)
- Low-power Schottky TTL (**74LS**)
- Advanced Schottky TTL (**74AS**)
- Advanced Low- power Schottky TTL (**74ALS**)
- Fast TTL (**74F**)

Fa

- **Resistor values** in the original TT were changed to obtain two m families with different perfo characteristics.
- The 74H (High- speed TTL) fam **lower resistor** values to propagation delay at the expe increased power consumption.
- The 74L (Low-power TTL) fami **higher resistor values** to reduce consumption at the expense o agation delay.

sistor Transistor logic NANDGate (7400)

below figure shows the **low power** TTL NAND

output configuration with transistor Q4 in the up path and transistor Q3 in the pull-down is called a **totem pole output**.

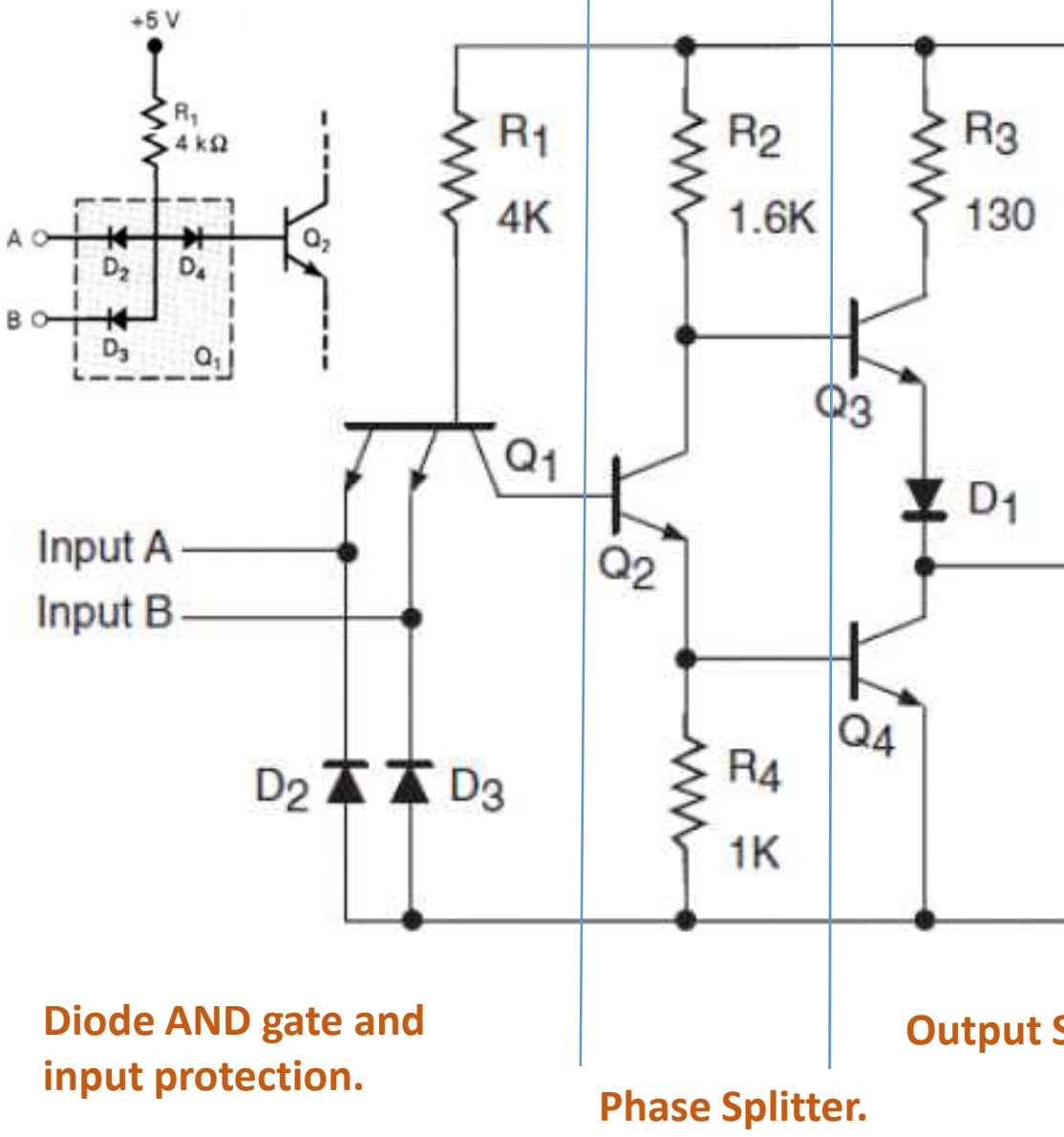
s called input transistor, which is **multi-ter transistor**, that drive transistor Q2 h is used to control Q3 and Q4.

e D1 and D2 is used to **protect Q1** from anted negative voltages.

e D3 ensures when **Q4 is ON, Q3 is OFF**.

n one or both inputs low, base-emitter tion of Q1 is forward bias, so Q1 ON and ut at collector will be low making Q2 off. ff and Q3 & D1 on making **output HIGH**.

n inputs high, base-emitter junction of s reverse bias so Q1 OFF and output at ctor will be high making Q2 ON. Q4 ON Q3 & D1 OFF, so **output is LOW**.

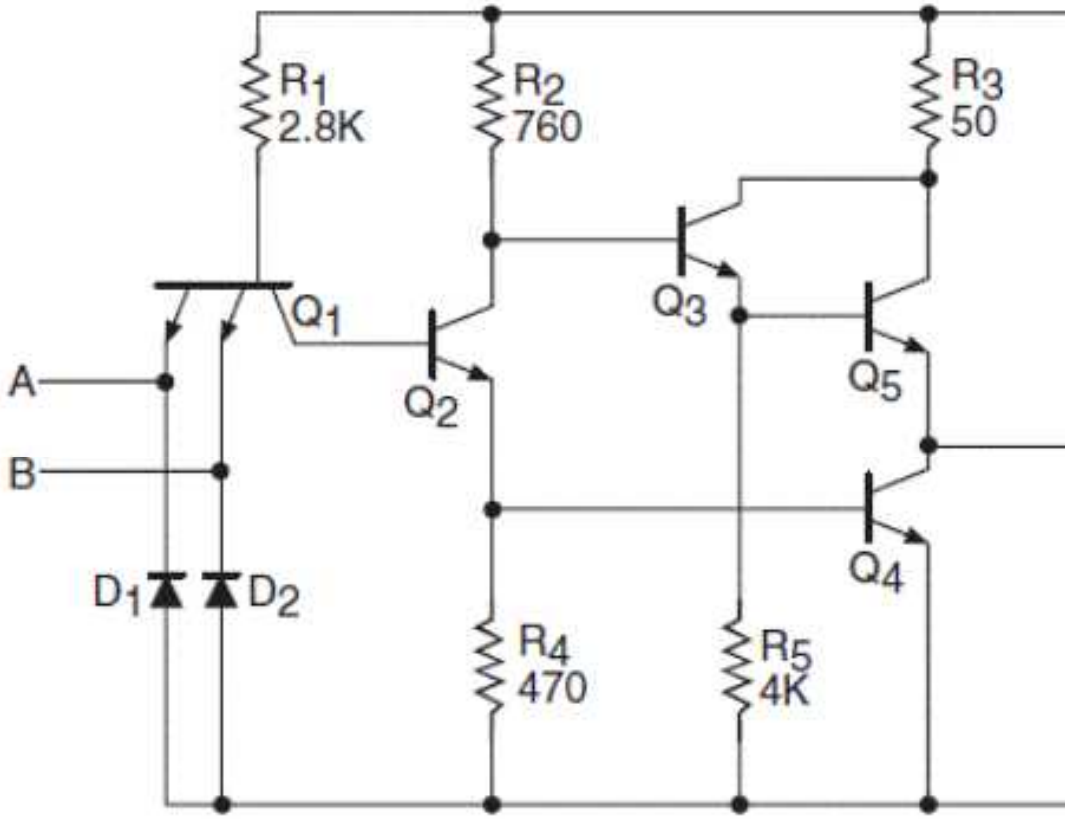


gh Speed NAND Gate (7400)

below drawn is very similar to that of a dard TTL **except Q3 transistor and D1** e combination.

ch has been replaced by an arrangement 3, Q5 and R5. This combination is called arlington pair combination.

speed of operation is higher and the er dissipation is also higher for this type TLs.

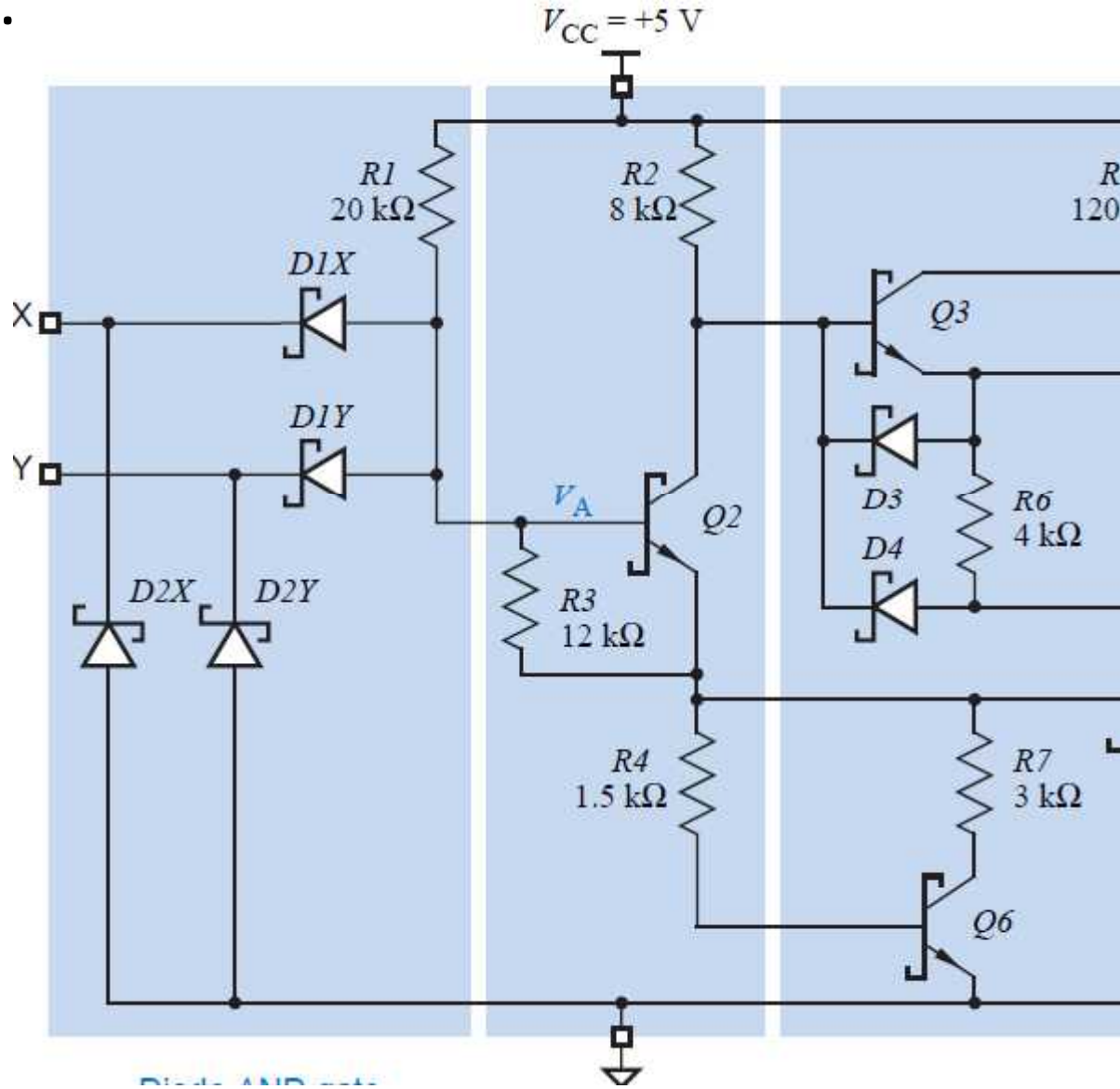


ottky TTL NAND Gate (74LS00)

uit's operation is best understood by dividing it into the three parts

- **Diode AND gate and input protection.**
- **Phase Splitter.**
- **Output Stage.**

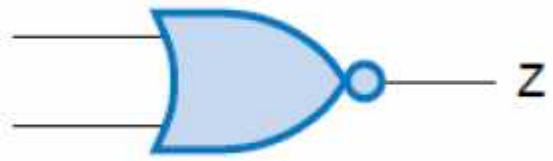
V_A	Q2	Q3	Q4	Q5	Q6	V_Z	Z
≤1.05	off	on	on	off	off	2.7	H
≤1.05	off	on	on	off	off	2.7	H
≤1.05	off	on	on	off	off	2.7	H
1.2	on	off	off	on	on	≤0.35	L

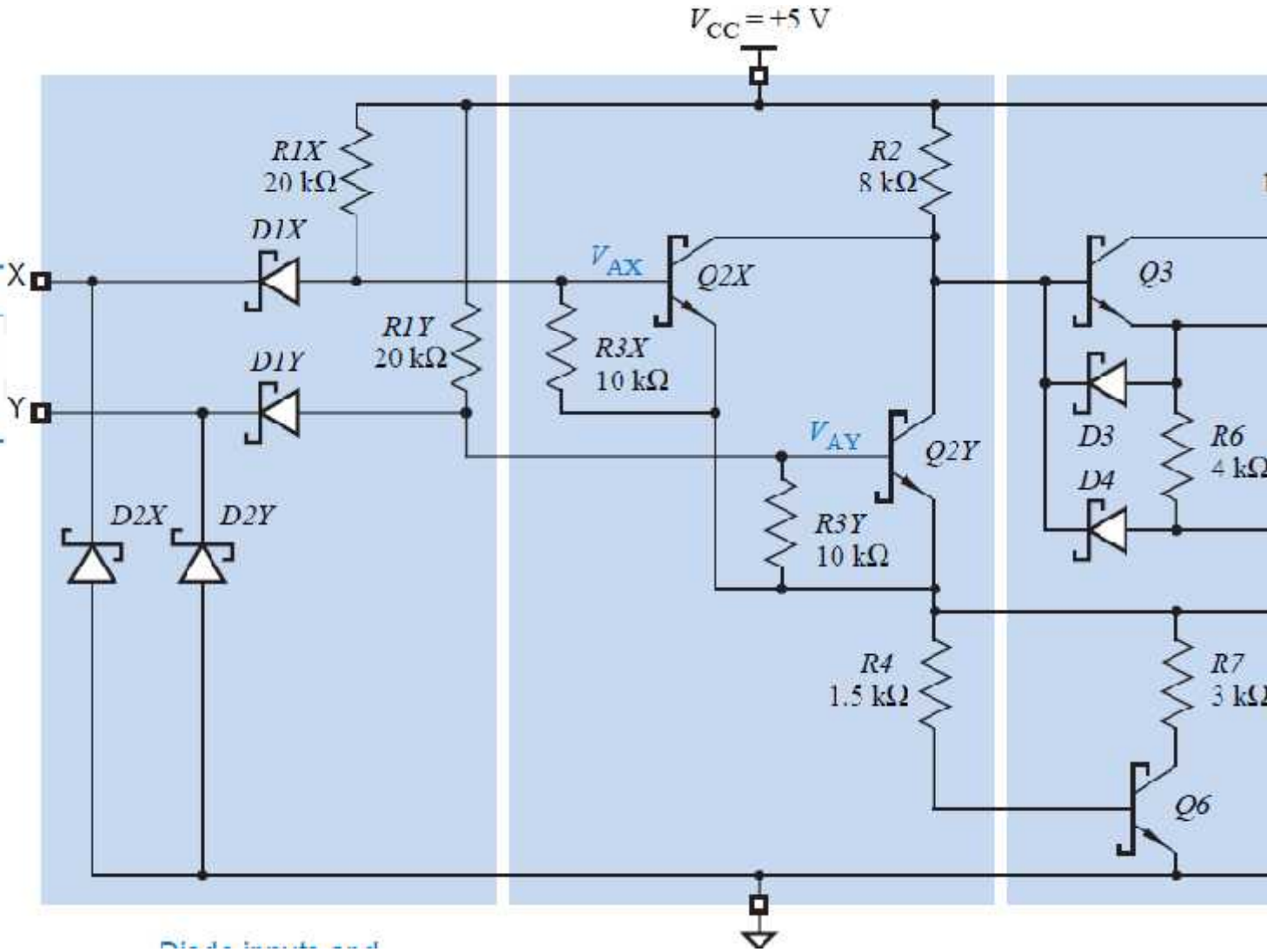


ottky TTL NOR Gate(74LS02)

uit's operation is best understood by dividing it into the three parts
de AND gate and input protection.
se Splitter.
tput Stage.

Q2X	V_{AY}	Q2Y	Q3	Q4	Q5	Q6	V_Z	Z
off	≤1.05	off	on	on	off	off	≥2.7	H
off	1.2	on	off	off	on	on	≤0.35	L
on	≤1.05	off	off	off	on	on	≤0.35	L
on	1.2	on	off	off	on	on	≤0.35	L

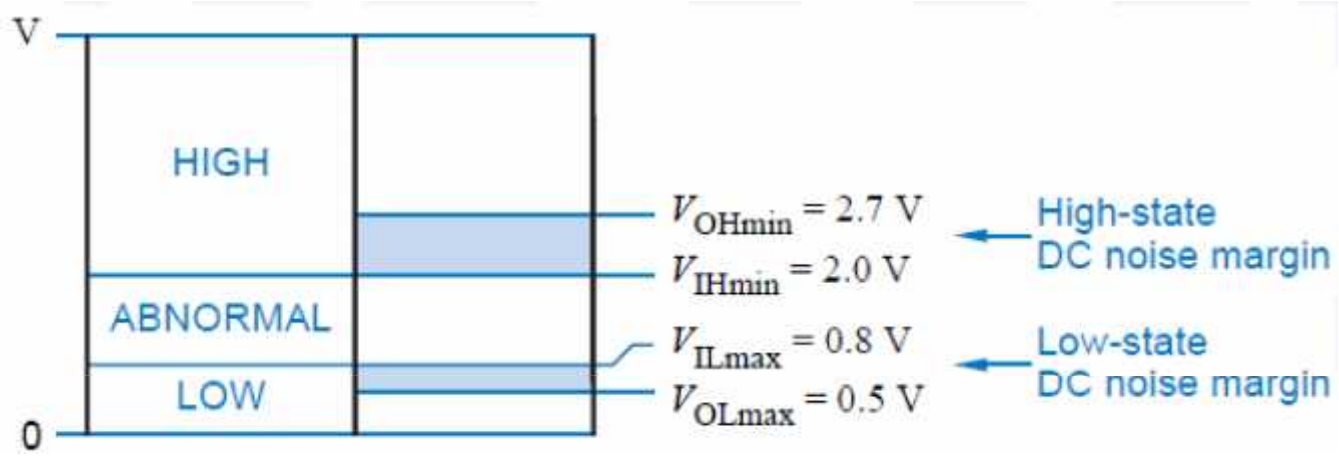




evels and Noise Margins

e TTL family the logic levels between **0 and 0.8 V to be LOW**, and levels between **2.0 and**

e HIGH.



DC noise margin in the the HIGH is .0.7 V

DC noise margin in the LOW state is only 0.3 V.

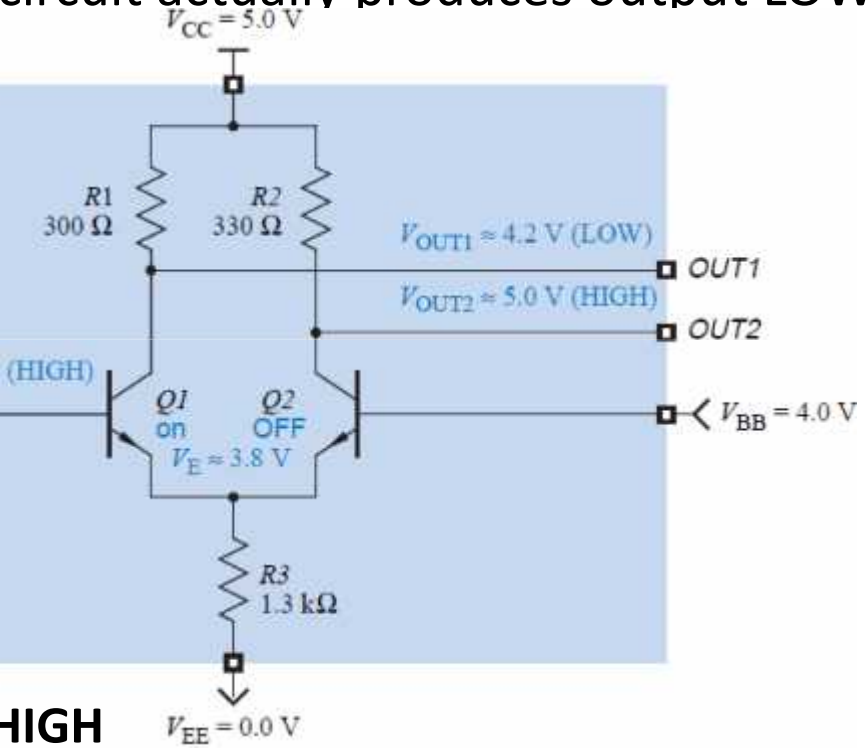
Fa

ter Coupled Logic (ECL)

key to **reducing propagation delay** in a bipolar logic family is to **prevent a gate's tran**
saturating.
does not produce a **large voltage swing** between the LOW and HIGH levels. Instead, it
l voltage swing.
ECL logic family is extremely fast, offering propagation delays as short as **1 ns.**

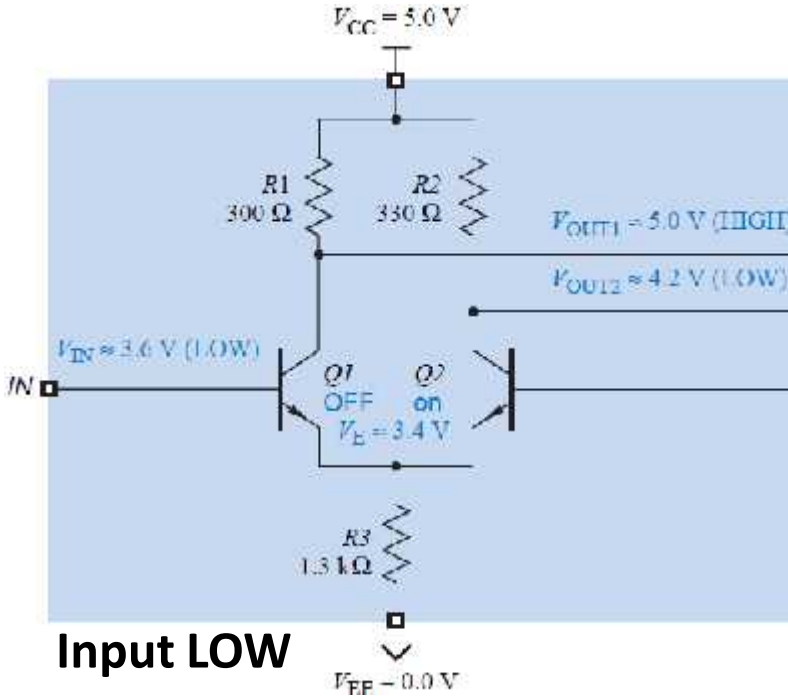
ic ECL Inverter/Buffer Gate

circuit has both an inverting output (OUT1) and a noninverting output (OUT2).
transistors are connected as a differential amplifier with a common emitter resistor.
nput LOW and HIGH levels are defined to be **3.6 and 4.4 V**.
circuit actually produces output LOW and HIGH levels that are **0.6 V higher** (4.2 and 5.0 V



HIGH

N is HIGH, transistor **Q1 is on**, but not saturated, sistor **Q2 is OFF**. Thus, **VOUT2** is pulled to **5.0 V** hrough R2, and it can be shown that the voltage oss R1 is about 0.8 V so that **VOUT1** is about **4.2 V**

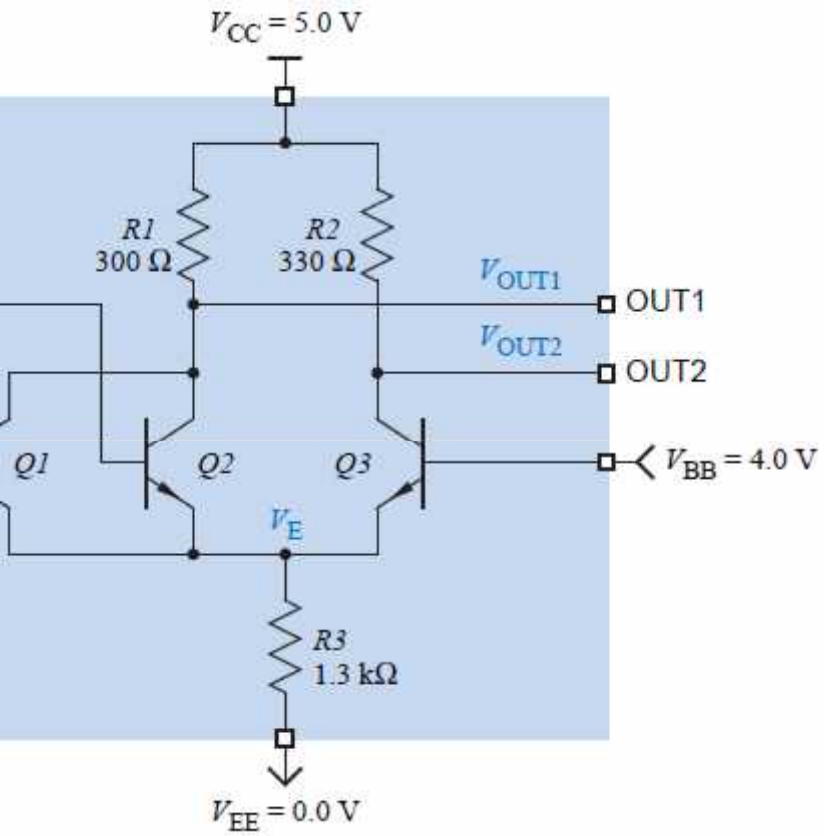


Input LOW

When **VIN is LOW**, transistor **Q2 is on**, but not sa and transistor **Q1 is OFF**.
Thus, **VOUT1** is pulled to **5.0 V** through R1, and it shown that **VOUT2** is about **4.2 V.**

two input OR/NOR Gate

circuit has both an inverting output (OUT1) and a noninverting output (OUT2).
transistors are connected as a differential amplifier with a common emitter resistor.



Circuit Diagram

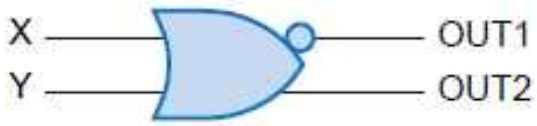
(b)

X	Y	V_X	V_Y	Q1	Q2	Q3	V_E	V_{OUT1}	V_{OUT2}	OUT1	OU
L	L	3.6	3.6	OFF	OFF	on	3.4	5.0	4.2	H	L
L	H	3.6	4.4	OFF	on	OFF	3.8	4.2	5.0	L	H
H	L	4.4	3.6	on	OFF	OFF	3.8	4.2	5.0	L	H
H	H	4.4	4.4	on	on	OFF	3.8	4.2	5.0	L	H

Functio

y input is HIGH, the corresponding input transistor is
e, and **VOUT1 is LOW (NOR output).**
e same time, Q3 is OFF, and **VOUT2 is HIGH (OR output).**

(c)



Shymbol

Interfac

TTL Interfacing

facing means connecting the outputs of one circuit to the inputs of another circuit tha
different electrical characteristics.

n two circuits have different electrical characteristics direct connection may not me ma
case driver and load circuits are connected through interface circuit.

e are several factors to consider in TTL/CMOS interfacing,

1. **Noise margin.**
2. **Fanout.**
3. **Capacitive loading.**

e **Noise Margin factor**, the driver output must satisfy the **voltage and current** requireme
oad circuit.

ving CMOS

TTL is driver and CMOS is a load circuit.

d Output Currents of

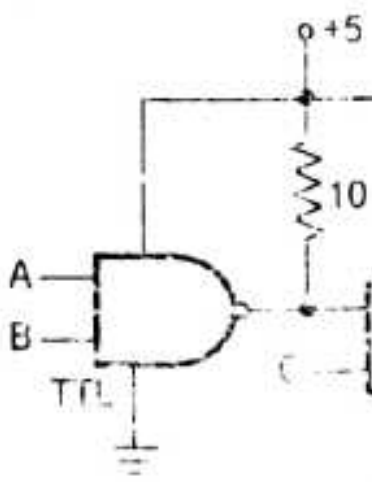
	CMOS		TTL		
	4000B	74HC/HCT	74	74LS	74AS
$I_{IH\ (max)}$	1 µA	1 µA	40 µA	20 µA	200 µA
$I_{IL\ (max)}$	1 µA	1 µA	1.6 mA	0.4 mA	2 mA
$I_{OH\ (max)}$	0.4 mA	4 mA	0.4 mA	0.4 mA	2 mA
$I_{OL\ (max)}$	0.4 mA	4 mA	16 mA	8 mA	20 mA

Interfa

n the table, it is observed that the input current values for CMOS are extremely low com
the output current capabilities of TTL series.
TTL has no problem meeting the CMOS input currents.
by comparing Voltages, We found that,

$$V_{OHmin} \text{ for TTL} << VI_{Hmin} \text{ for CMOS}$$

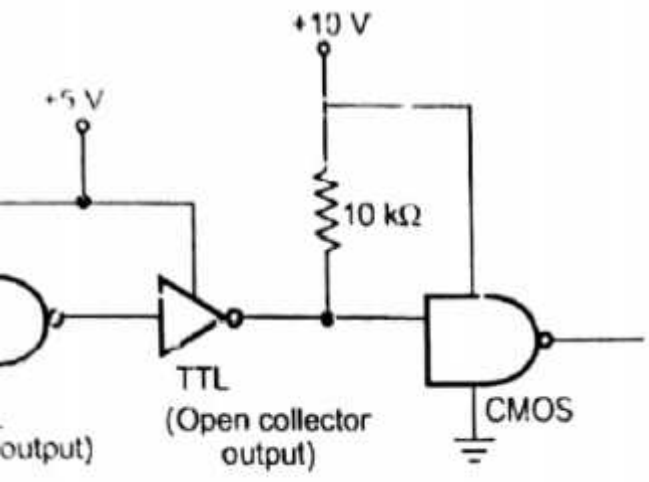
situations TTL outputs must be raised to an acceptable level for CMOS.



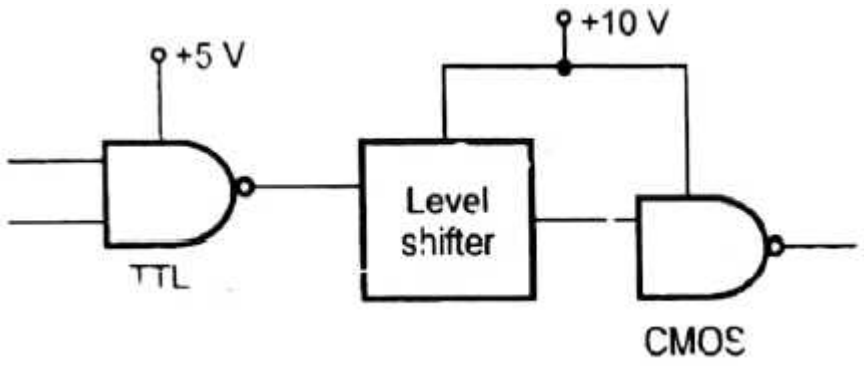
TTL driving C
with Pull-up

ving High Voltage CMOS

Buffer as Interface



Using Level shifter as Interface



Interfa

Driving TTL

ow table shows the CMOS outputs and TTL input characteristics

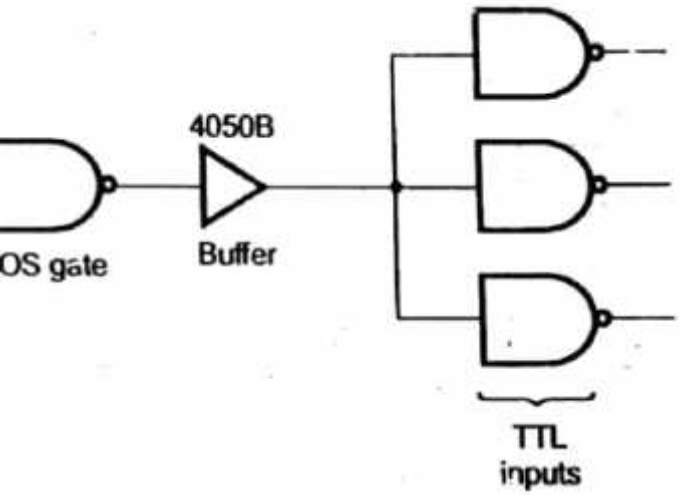
For CMOS (4000B)	For TTL
$V_{OH(min)}$: 4.95 V	$V_{IH(min)}$: 2.0 V
$V_{OL(max)}$: 0.05 V	$V_{IL(max)}$: 0.8 V
$I_{OH(max)}$: 0.4 mA	$I_{IH(max)}$: 40 µA
$I_{OL(max)}$: 0.4 mA	$I_{IL(max)}$: 1.6 mA

H State:

, no special consideration is required for CMOS driving TTL in the HIGH state.

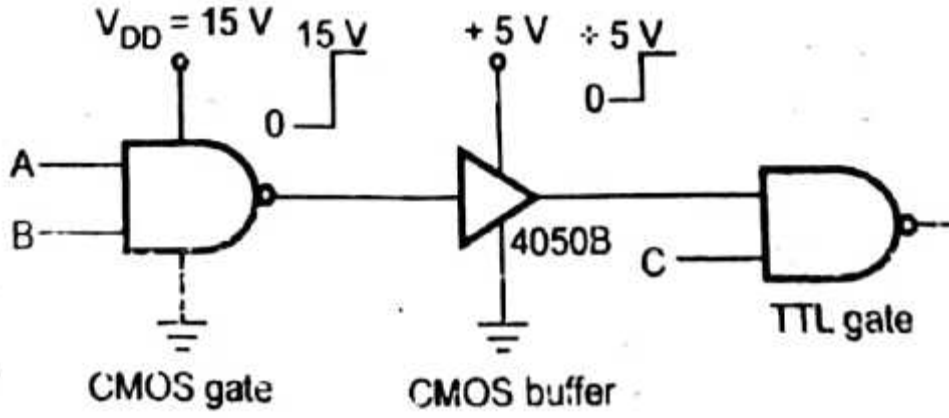
V State:

rent requirement in the LOW state are not satisfied.



iving TTL in LOW state using Buffer

High Voltage CMOS driving TTL



Interfac

important factors may consider in IC industry to move towards **lower power- supply vo**

MOS devices:

Cutting power-supply voltage reduces **dynamic power dissipation** more than proportiona

The oxide insulation between a CMOS transistor's gate and its source and drain is gettin

thinner, and thus **incapable of insulating voltage potentials** as "high" as 5 V.

C, an IC industry standards group, selected **3.3V ± 0.3V**, **2.5V ± 0.2V,** and **1.8 V ± 0.15V**

"standard" logic power-supply voltages.

v voltage,such as **2.5 V**, is supplied to operate the chip's **internal gates**, or ***core logic***.

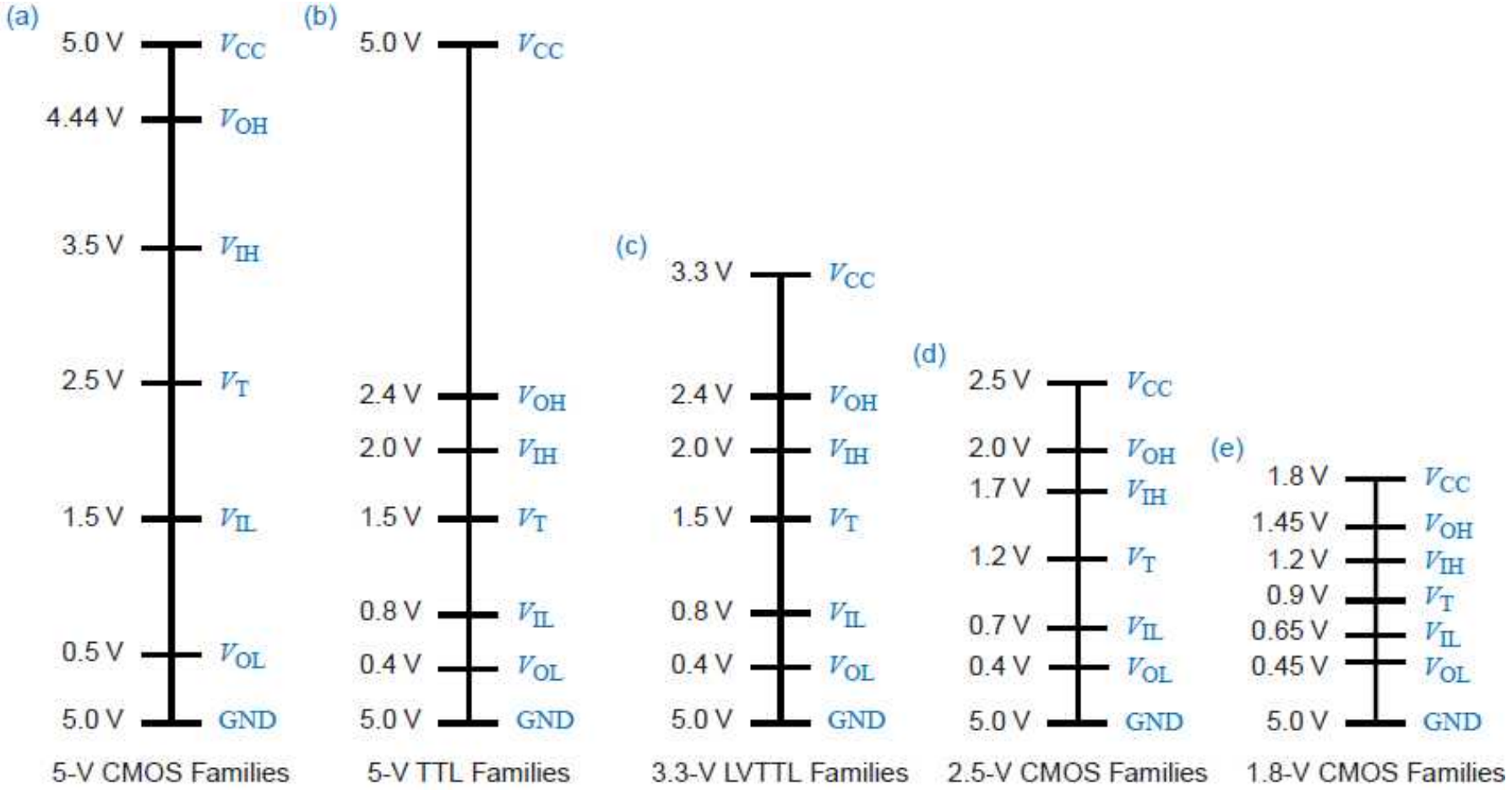
gher voltage, such as **3.3 V,** is supplied to operate the ***external input and output circuits***, c

VTTL and LVCMOS Logic

Interfac

JEDEC standard for 3.3-V logic actually defines **two sets of levels**.

MOS (low-voltage CMOS) levels are used in pure CMOS applications where outputs hav
oads (less than 100 uA), so V_{OL} and V_{OH} are maintained within 0.2 V of the power-supply ra
L (low-voltage TTL) levels are used in applications where outputs have significant DC loa
an be as high as 0.4 V and V_{OH} can be as low as 2.4 V.



UNIT-II

HARDWARE DESCRIPTION LANGUAGES

Based Digital Design, The VHDL Hardware Description Language–Program Structure, Types, Co

rrays, Functions and procedures, Libraries and Packages, Structural design elements, Dataflow

nts, Behavioral design elements, The Time Dimension, Simulation, Test Benches, VHDL Featu

ntial Logic Design, Synthesis

INTRODUCTION TO VHDL

- VHDL is a language that is used to describe the behavior of digital circuit designs

Very High Speed Integrated Circuit
Hardware
Description
Language

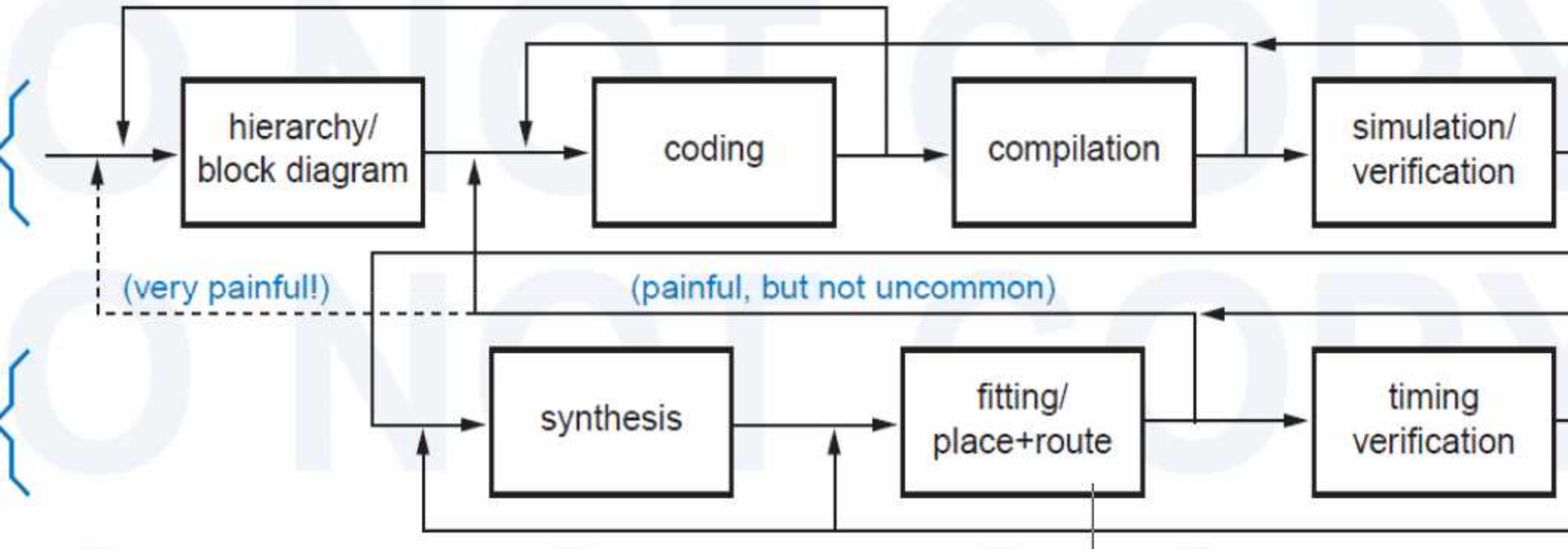
- VHDL designs can be simulated and translated into a form suitable for hardware implementation

- Hierarchical use of VHDL designs permits the rapid creation of complex digital circuit designs

HISTORY OF VHDL

- VHDL was developed by the VHSIC (Very High Speed Integrated Circuit) Program in the late 1970s and early 1980s
  - The VHSIC program was funded by the U.S. Department of Defense
  - Existing tools were inadequate for complex hardware designs
- The evolution of VHDL has included the following milestones:
  - In 1981, VHDL was first proposed as a hardware description language
  - In 1986, VHDL was proposed as an IEEE standard
  - In 1987, the first VHDL standard (IEEE-1076-1987) was adopted
  - In 1993, a revised VHDL standard (IEEE-1076-1993) was adopted
  - In 2002, the current VHDL standard (IEEE-1076-2002) was adopted
- VHDL is now used extensively by industry and academia for the purpose of simulating and synthesizing digital circuit designs

HDL Based Digital Design



HDL based design flow

Program Structure

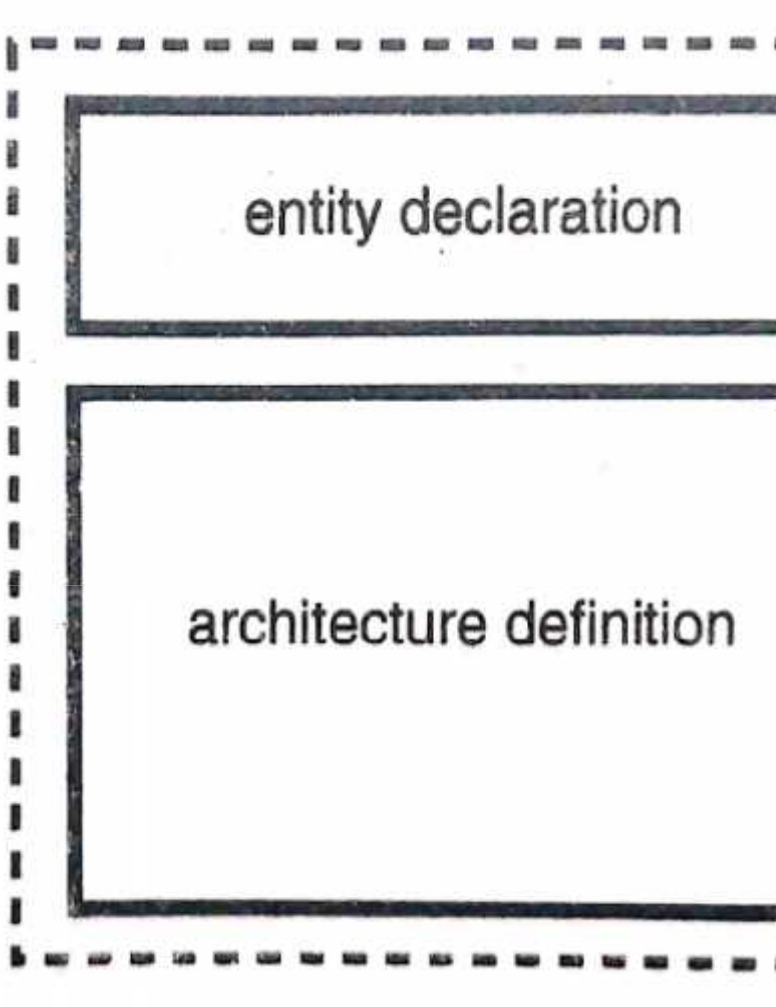
he VHDL program structure having the vo blocks, which are:

- **Entity**
- **Architecture**

ntity: entity is simply a declaration of a odule's inputs and outputs.

rchitecture: Architecture is a detailed escription of module's internal behavior r structure.

n the text file of a VHDL program, the ntity declaration and architecture eclaration is separated.



VHDL program file structure

Program Stru

y declaration:

ic entity declaration has the syntax shown in below

e purpose of the entity declaration is to define
external interface signals or ports in port
laration
ddition to the keywords *entity*, *is*, *port*, *end* ,an
ity declaration has the following elements
ity-name: A user selected identifier
nal-names: A comma-separated list of one or
re user selected identifiers to name the external
erface
de: One of four reserved words, specifying the
nal directions
the signal is an input to the entity
: The signal is an output of the entity
ut: the signal can be used as input or output of
entity
ffer: the signal is an output of the entity
nal-type: A built-in or user-defined signal type

```
entity entity-name is
  port (signal-names : mode signa
        signal-names : mode signa
        ...

        signal-names : mode signa
end entity-name;
```

VHDL entity declaratio

Program Stru

itecture declaration:

ic architecture declaration has the syntax shown in below

architecture-name is a user-selected
tifier, usually related to the entity
e.
entity-name must be the same as one
n previously in the entity
aration.

al declaration

signal-names : signal type

ble declaration

ble variable-names : variable type

```
architecture architecture-name of entity
  type declarations
  signal declarations
  constant declarations
  function definitions
  procedure definitions
  component declarations
begin
  concurrent-statement
  ...
  concurrent-statement
end architecture-name;
```

VHDL architecture declaration

Types, Constants and Arrays

:

ne type specifies the set or range of values that the object can take on an
ere is also typically set of operators (such as add, AND..)
HDL has few predefined types such as..

bit	character	severity_level
bit_vector	integer	string
boolean	real	time

VHDL predefined types

integer Operators		*boolean* Operators	
+	addition	and	AND
-	subtraction	or	OR
*	multiplication	nand	NAND
/	division	nor	NOR
mod	modulo division	xor	Exclusive OR
rem	modulo remainder	xnor	Exclusive NOR
abs	absolute value	not	complementation
**	exponentiation		

Predefined operator for VHDLs integer and boolean types

Types, Constants and Ar

he most commonly used types in typical VHDL programs are *user defin*
pes.

- *enumerated type*
- *standard user defined logic type (std_logic)*

he most common of the user defined type is *enumerated type*, which are
fined by listing their values.
he predefined types *boolean* and *character* are enumerated type
numerated type and **constant** declaration has the following syntax

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

subtype subtype-name is type-name start to end;
subtype subtype-name is type-name start downto end;

constant constant-name: type-name := value;
```

VHDL predefined types

andard user defined logic type (std_logic):

```
type STD_ULOGIC is ( 'U',   -- Uninitialized
                     'X',   -- Forcing  Unknown
                     '0',   -- Forcing  0
                     '1',   -- Forcing  1
                     'Z',   -- High Impedance
                     'W',   -- Weak     Unknown
                     'L',   -- Weak     0
                     'H',   -- Weak     1
                     '-'    -- Don't care
                   );
subtype STD_LOGIC is resolved STD_ULOGIC;
```

Definition of VHDL *std_logic* type

Types, Constants and Ar

ys

nother very important category of used defined types are *array types*
ne syntax for VHDL array declaration as shown in below

```
pe type-name is array (start to end) of element-type;
pe type-name is array (start downto end) of element-type;
pe type-name is array (range-type) of element-type;
pe type-name is array (range-type range start to end) of element-type;
pe type-name is array (range-type range start downto end) of element-type
```

VHDL array declaration

the first two versions, *start* and *end* are integers that define the possibl
nge of the array index
the last three versions, all or a subset of the values of an existing type
nge type) are the range of the array index.

Types, Constants and Ar

ys

xamples of array declaration

```
type monthly_count is array (1 to 12) of integer;
type byte is array (7 downto 0) of STD_LOGIC;

constant WORD_LEN: integer := 32;
type word is array (WORD_LEN-1 downto 0) of STD_LOGIC;

constant NUM_REGS: integer := 8;
type reg_file is array (1 to NUM_REGS) of word;

type statecount is array (traffic_light_state) of integer;
```

VHDL array declaration

Functions and Procedures

VHDL function accepts a number of ***arguments*** and returns the ***results.***
VHDL procedure is similar to a function, except it does not return a result.
ile a function call can be used in the place of an expression, a procedure call can be
ne place of statement.

unction may define its own local types,
stants, variables and nested functions and
cedures.
e keywords *begin* and *end* enclose a series
equential statements that are executed
en the function is called.

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

syntax for function definition

Functions and Procedures

```
ion function-name (
signal-names : signal-type;
signal-names : signal-type;
...
signal-names : signal-type
urn return-type is
 declarations
stant declarations
able declarations
ction definitions
cedure definitions

uential-statement

uential-statement
unction-name;
```

```
architecture Inhibit_archf of Inhibit

function ButNot (A, B: bit) return bit
begin
  if B = '0' then return A;
  else return '0';
  end if;
end ButNot;

begin
  Z <= ButNot(X,Y);
end Inhibit_archf;
```

Example of the function declaration of inhibit gate

Libraries and Packages

VHDL ***library*** is a place where the VHDL compiler stores the information abo
ticular design project, including intermediate files used in the analysis, simulation
thesis of the design.

a given VHDL design, the compiler automatically creates and use a library na
rk .

e designer can specify the name of such library using a ***library*** *clause* at the begin
he design file.

example IEEE library cab be specified as

library *ieee*; (**library** *work*;)

e VHDL ***package*** is a file containing definitions of objects that can be used in c
grams.

e kind of objects that can be put into a package include signal, type, consta
ctions, procedure and component declarations.

nals that are defined in a package are "**global**" signals, available to any VHDL e
t uses the package.

Libraries and Packages

esign can “**use**” a package by including a ***use*** *clause* at the beginning of the de

.

example, to use all of the definitions in the IEEE standard 1164 package, it ca

tten as

use *ieee.std_logic_1164.all*;

Here *ieee* is the name of the library,

within this library “*std_logic_1164*” contains the desired definitions.

The suffix *all* tells the compiler to use all of the definitions in this file

tead of “*all*” you can write the name of the particular object to use just its definitio

Ex: **use** *ieee.std_logic_1164.std_ulogic*;

Libraries and Packages

```
ckage package-name is
ype declarations
ignal declarations
onstant declarations
component declarations
function declarations
procedure declarations
d package-name;
ckage body package-name is
type declarations
constant declarations
function definitions
procedure definitions
d package-name;
```

All of the objects declared between “package” first “end” statement are visible in any design f uses the package.

All of the objects declared between “pack body” are local.

Syntax of a VHDL package definition

VHDL Design Elements

ere are three design elements in the VHDL (Modeling
yles)

tructural Design

Data Flow Design

Behavioural Design

VHDL Design Elements

ctural Design

he body of the VHDL architecture is a series of concurrent statements.
n VHDL, each concurrent statement executes simultaneously with the
oncurrent statements in the same architecture body.
A VHDL architectures that uses ***components*** is often called a **struct**
escription or structural design, because it defines the pr
nterconnection structure of signals and entities that realize the entity.
A pure structural design is equivalent to a schematic or net list for the ci

```
bel: component-name port map(signal1, signal2, ..., signaln);

bel: component-name port map(port1=>signal1, port2=>signal2, ..., portn=>signaln
```

Syntax of VHDL component statements

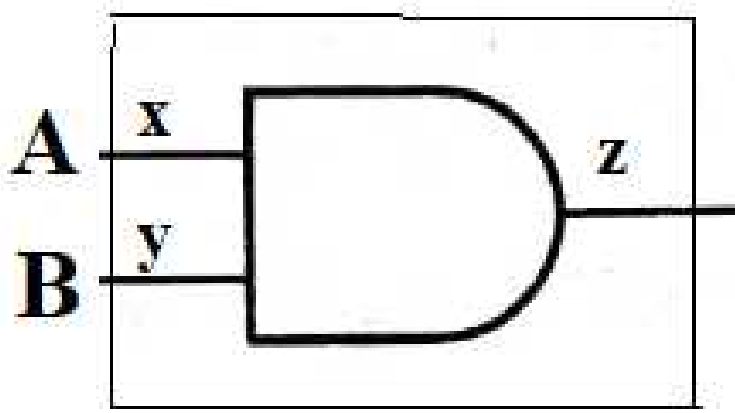
e ***port map*** keywords introduce a list that associates ports of the named entity
nals in the current architecture.
e list may be written in either of **two different styles**.
e first is a **positional style**; as in conventional programming languages, the signals
are associated with the **entity's ports in the same order** that they appear in the e
inition.
e second is an **explicit style**; each of the entity's ports is connected to a signal usi
>" operator and these associations may be listed in any order

```
mponent component-name
port (signal-names : mode signal-type;
      signal-names : mode signal-type;
      ...
      signal-names : mode signal-type);
d component;
```

Syntax of VHDL component declar

ructural VHDL program for basic gates

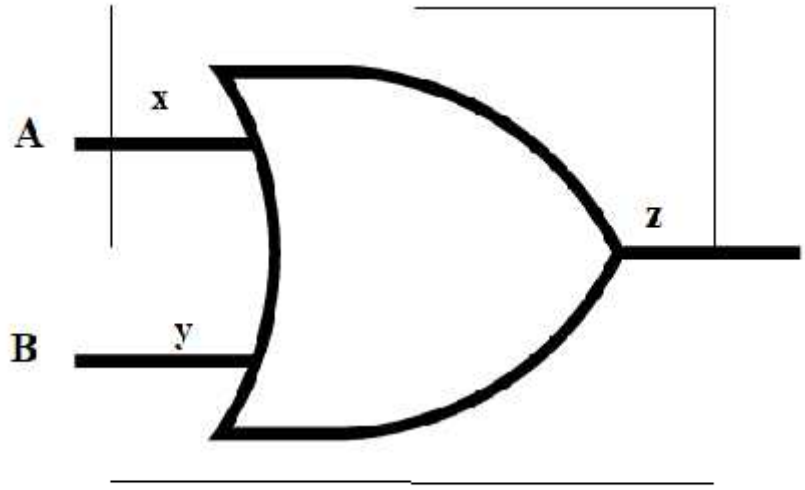
```
brary IEEE;
se IEEE.std_logic_1164.all;
ntity andgate is
ort (A,B: in std_logic;
        C: out std_logic);
nd andgate;
rchitecture logic_gates of andgate is
omponent AND_gate
ort (x,y: in std_logic;
       z:out std_logic);
nd component;
egin
: AND_gate port map (A,B,C);
nd logic_gates;
```



AND Gate

ructural VHDL program for basic gates

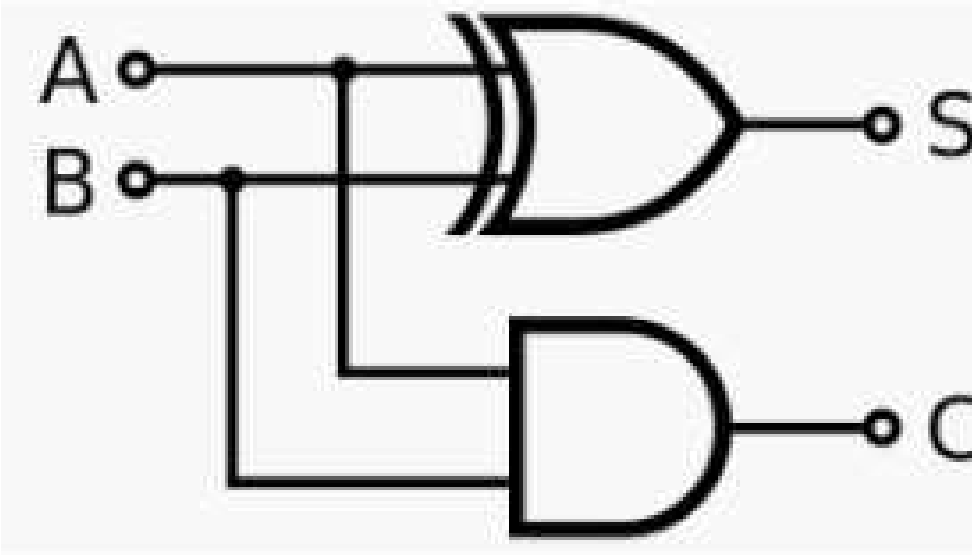
```
brary IEEE;
se IEEE.std_logic_1164.all;
ntity orgate is
ort (A,B: in std_logic;
        C: out std_logic);
nd orgate;
rchitecture logic_gates of orgate is
omponent OR_gate
ort (x,y: in std_logic;
       z:out std_logic);
nd component;
egin
: OR_gate port map (A,B,C);
nd logic_gates;
```



OR Gate

ctural VHDL program for Half Adder

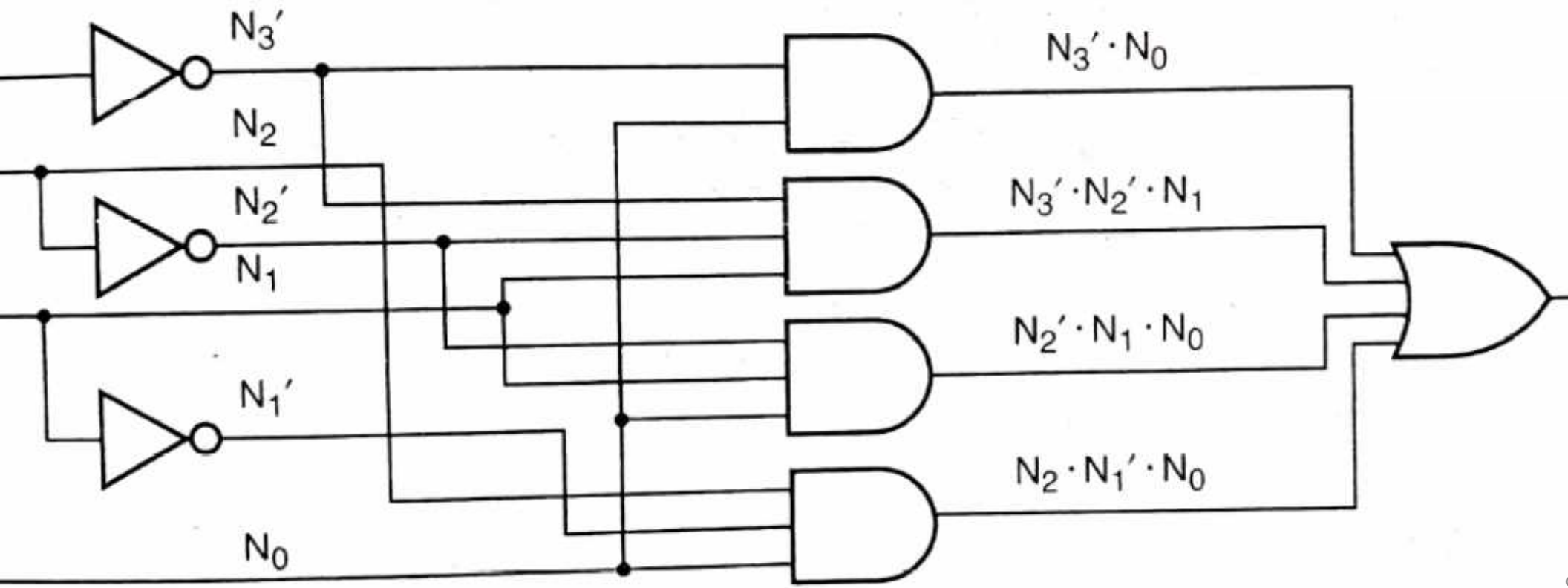
```
ry IEEE;
EEE.std_logic_1164.all;
y half_adder is
(A,B: in std_logic;
   S,C: out std_logic);
nalf_adder;
tecture half_adder_arch of half_adder is
ponent XOR_gate
(x,y: in std_logic;
  z:out std_logic);
component;
ponent AND_gate
(x1,y1: in std_logic;
  z1:out std_logic);
component;
n
OR_gate port map (A,B,S);
ND_gate port map (A,B,C);
nalf_adder_arch;
```



Structural VHDL program for prime number detector

$$F = \Sigma_{N3,N2,N1,N0}(1,2,3,5,7,11,13)$$

$$F = N_3' \cdot N_0 + N_3' \cdot N_2' \cdot N_1 + N_2' \cdot N_1 \cdot N_0 + N_2 \cdot N_1' \cdot N_0$$



```
 IEEE.std_logic_1164.all;
rary unisim;
 unisim.vcomponents.all;

ty prime is
port ( N: in STD_LOGIC_VECTOR (3 downto 0);
       F: out STD_LOGIC );
prime;

itecture prime1_arch of prime is
al N3_L, N2_L, N1_L: STD_LOGIC;
al N3L_N0, N3L_N2L_N1, N2L_N1_N0, N2_N1L_N0: STD_LOGIC;
onent INV port (I: in STD_LOGIC; O: out STD_LOGIC); end component;
onent AND2 port (I0,I1: in STD_LOGIC; O: out STD_LOGIC); end component;
onent AND3 port (I0,I1,I2: in STD_LOGIC; O: out STD_LOGIC); end component;
onent OR4 port (I0,I1,I2,I3: in STD_LOGIC; O: out STD_LOGIC); end component;
n
  INV port map (N(3), N3_L);
  INV port map (N(2), N2_L);
  INV port map (N(1), N1_L);
  AND2 port map (N3_L, N(0), N3L_N0);
  AND3 port map (N3_L, N2_L, N(1), N3L_N2L_N1);
  AND3 port map (N2_L, N(1), N(0), N2L_N1_N0);
  AND3 port map (N(2), N1_L, N(0), N2_N1L_N0);
  OR4 port map (N3L_N0, N3L_N2L_N1, N2L_N1_N0, N2_N1L_N0, F);
rime1_arch;
```

nerate:

n some applications, it is necessary to create **multiple copies** of a parti
tructure within an architecture

VHDL includes a ***generate* statement** that allows you to create
epetitive structures using a kind of "***for loop***," without having to writ
ll of the component instantiations individually.

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

Syntax of VHDL for-generate loop

erate:

```
ary IEEE;
IEEE.std_logic_1164.all;

ty inv8 is
port ( X: in STD_LOGIC_VECTOR (1 to 8);
       Y: out STD_LOGIC_VECTOR (1 to 8) );
inv8;

itecture inv8_arch of inv8 is
onent INV port (I: in STD_LOGIC; O: out STD_LOGIC); end component;
n
: for b in 1 to 8 generate
    U1: INV port map (X(b), Y(b));
  end generate;
inv8_arch;
```

VHDL entity and architecture for 8-bit inverter

eric:

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

Syntax of VHDL generic declaration within an entity declaration

eric:


```
ibrary IEEE;
se IEEE.std_logic_1164.all;

ntity businv is
   generic (WIDTH: positive);
   port ( X: in STD_LOGIC_VECTOR (WIDTH-1 downto 0);
          Y: out STD_LOGIC_VECTOR (WIDTH-1 downto 0) );
nd businv;

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

aflow Design

he body of the VHDL architecture is a series of concurrent statements.
n dataflow design, these concurrent statements allows to describe the ci
n terms of operations on signals and flow of signals in the circuit.
When such concurrent statements are used in a VHDL program ,the sty
alled a **"dataflow design".**
he concurrent signal assignment statements are used in this style.

```
gnal-name <= expression;

gnal-name <= expression when boolean-expression else
             expression when boolean-expression else
             ...
             expression when boolean-expression else
             expression;
```

Syntax of VHDL concurrent signal assignment statements

aflow Design

The another syntax of signal assignment using **selected signal** a
tatement.

```
th expression select
 signal-name <= signal-value when choices,
                signal-value when choices,
                ...
                signal-value when choices;
```

Syntax of VHDL selected signal assignment statements

taflow VHDL program for basic gates

AND GATE

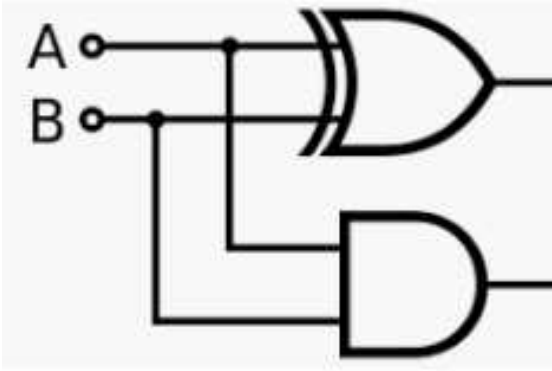
```
 IEEE;
EE.std_logic_1164.all;
 andgate is
A,B: in std_logic;
 C: out std_logic);
ndgate;
ecture and_dataflow of andgate is

AND B;
nd_dataflow;
```

OR GATE

```
library IEEE;
use IEEE.std_logic_1164.all;
entity orgate is
Port (A,B: in std_logic;
      C: out std_logic);
end orgate;
architecture or_dataflow of org
begin
C<= A OR B;
end or_dataflow;
```

aflow VHDL program for Half Adder

```
ary IEEE;
 IEEE.std_logic_1164.all;
ity half_adder is
t (A,B: in std_logic;
      S,C: out std_logic);
 half_adder;
hitecture half_adder_arch of half_adder is
in
= A XOR B;
= A AND B;
 half_adder_arch;
```



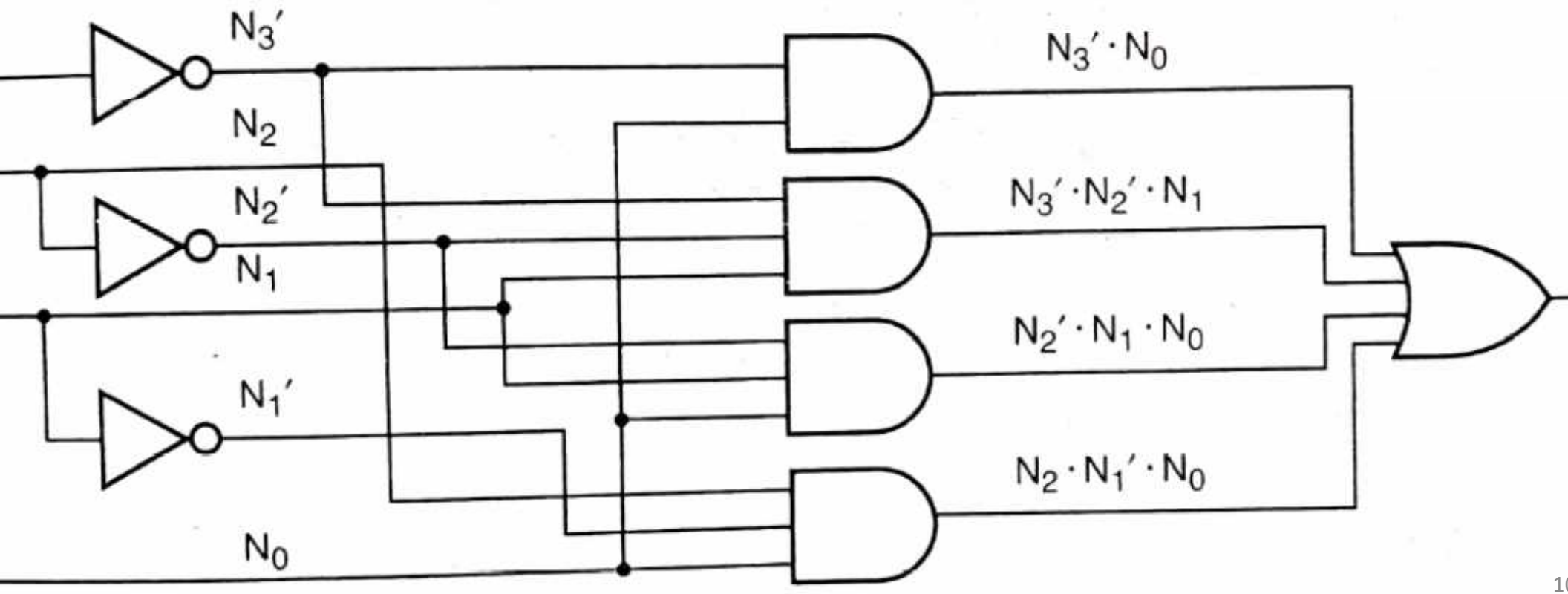
S = A ă B

C = AB

VHDL program for prime number detector

$$F = \Sigma_{N3,N2,N1,N0}(1,2,3,5,7,11,13)$$

$$F = N_3' \cdot N_0 + N_3' \cdot N_2' \cdot N_1 + N_2' \cdot N_1 \cdot N_0 + N_2 \cdot N_1' \cdot N_0$$



low VHDL program for prime number detector

$F = \Sigma_{N3,N2,N1,N0}(1,2,3,5,7,11$

$F = N_3' \cdot N_0 + N_3' \cdot N_2' \cdot N_1 + N_2' \cdot N_1 \cdot N_0$

```
brary IEEE;
e IEEE.std_logic_1164.all;
brary unisim;
e unisim.vcomponents.all;

tity prime is
  port ( N: in STD_LOGIC_VECTOR (3 downto 0);
         F: out STD_LOGIC );
d prime;
chitecture prime2_arch of prime is
gnal N3L_N0, N3L_N2L_N1, N2L_N1_N0, N2_N1L_N0: STD_LOGIC;
gin
N3L_N0     <= not N(3)                                    and N(0);
N3L_N2L_N1 <= not N(3) and not N(2) and     N(1)                ;
N2L_N1_N0  <=              not N(2) and     N(1) and N(0);
N2_N1L_N0  <=                   N(2) and not N(1) and N(0);
F <= N3L_N0 or N3L_N2L_N1 or N2L_N1_N0 or N2_N1L_N0;
d prime2_arch;
```

Behavioral Model

DL's key behavioral element is the **"process**."

rocess is a collection of "**sequential**" statements that executes in parallel with current statements and other processes.

HDL ***process statement*** can be used anywhere that a concurrent statement can be u

ocess statement is introduced by the keyword ***process***.

```
cess (signal-name, signal-name, ..., signal-name)
ype declarations
ariable declarations
onstant declarations
unction definitions
rocedure definitions
in
equential-statement
..
equential-statement
 process;
```

Syntax of a VHDL p
statements

```
oolean-expression then sequential-statements
if;

oolean-expression then sequential-statements
e sequential-statements
if;

oolean-expression then sequential-statements
if boolean-expression then sequential-statements

if boolean-expression then sequential-statements
if;

boolean-expression then sequential-statements
if boolean-expression then sequential-statements

if boolean-expression then sequential-statements
e sequential-statements
if;
```

Syntax of a VHDL i
statement

of a VHDL case statement

```
 expression is
n choices => sequential-statements

n choices => sequential-statements
ase;
```

```
dentifier in range loop
uential-statement

uential-statement
oop;
```

of a VHDL for loop statement

Syntax of a VHDL loop statem

```
loop
  sequential-statement
  ...
  sequential-statement
end loop;
```

```
while boolean-expression
  sequential-statement
  ...
  sequential-statement
end loop;
```

Syntax of a VHDL while loop state

havioral VHDL program for basic gates

AND GATE

```
EEE;
E.std_logic_1164.all;
ndgate is
B: in std_logic;
C: out std_logic);
lgate;
cture and_behv of andgate is

 (A, B)

 AND B='1' then
1';

0';

cess;
l_behv;
```

AND GATE (Using ca

```
library IEEE;
use IEEE.std_logic_1164.all;
entity andgate is
Port (A,B: in std_logic;
      X: in std_logic_vector (1 downto 0);
        C: out std_logic);
end andgate;
architecture and_behv of andgate is
begin
process (A, B, X)
begin
case  X is
when '00'   =>  C<= '0';
when '01'   =>  C<= '0';
when '10'   =>  C<= '0';
when others => C<= '1';
end case;
end process;
end and_behv;
```

Behavioral VHDL program for 4x1 Multip

```
y IEEE;
EE.STD_LOGIC_1164.ALL;
 Mux_4x1 is
t ( a,b,c,d: in  STD_LOGIC;
    sel : in  STD_LOGIC_VECTOR (1 downto 0);
    out1 : out  STD_LOGIC);
lux_4x1;
ecture Behavioral_4x1 of Mux_4x1 is

    process(a, b, c, d, sel)
    begin
    case sel is
            when "00" =>
                    out1 <= a;
            when "01" =>
                    out1 <= b;
            when "10" =>
                    out1 <= c;
            when others =>
                    out1 <= d;
    end case;
    end process;
ehavioral_4x1;
```

The Time Dimension

DL allows you to specify a **time delay** using the two different keyword

1. ***after***
2. ***wait***

any signal assignment statement, including sequential, concurrent, conditiona
ected assignments.

example

```
Z <= '1' after 4ns when X='1' and Y='0' else '0' after 3ns;
```

ere the output ‘Z’ that has 4 ns of delay on a 0-to-1 output transition, and only 3 ns
-to-0 transition.

t statement can be used to **suspend a process for a specified time period**

t statement can be three different forms**:**

it on sensitivity list; ---------- ***wait on*** clock, reset;

it for time-expression; ---------- ***wait for*** 10ns;

it until Boolean expression; ---------- ***wait until*** A=B;

it --------*This statements suspends the process indefinit*

SIMULATION

DL simulator can be used to observe the operation of the circuit after correctin
ax and semantics of VHDL program.

following steps are carried out when the VHDL simulator is applied for a V
gram.

Initialization:

- ✓ The simulator operation is started at **simulation time of zero**
- ✓ At this stage all signals are initialized to a default value.
- ✓ All the processes in the program are scheduled for execution

Simulation cycle:

- ✓ The simulator can not simulate all of the processes and concurrent state simultaneously.
- ✓ It can be pretend it using time-based “event list” and “signal sensitivity matrix”
- ✓ According to the scheduling of the processes for execution, one of the process is selec

Event List:

- ✓ During the process execution new values of the signals are assigned, but these new values directly assigned.
- ✓ They are placed on a list called “event list”.
- ✓ In the event list, at what time these new values to be assigned to the signals is also scheduled.
- ✓ Ex: after 10ns; wait for 20ns;

SIMULATION

Delta delay:

- ✓ When any signal is not placed on the event list, it is supposed to occur immed but practically it will not happen.
- ✓ It is actually scheduled to occur at the current simulation time plus infinitesi short time.
- ✓ This infinitesimally short time is called as delta time

Next simulation cycle:

- ✓ After completion of a first simulation cycle, the event list is scanned.
- ✓ According to scheduling, when change occurs in the signal value, the pr execution is started. This is the beginning of next simulation cycle.

Completion of the simulation:

- ✓ The above step is repeated until the event list is empty.
- ✓ When all the simulation cycles are executed by scanning the event list, assi new signal values according to the scheduling and execution of processes acc to the scheduling.
- ✓ It can say that the simulation is complete.

SYNTHESIS

The synthesis process converts the VHDL code into a set of primitives or compo
that can be assembled in a required technology.

The quality of the synthesised circuits greatly depends on the VHDL code writt
them.

For example

- ✓ When conditional statements are used in the process, sometimes for some combination an output can not be obtained.
- ✓ Generally the loops in the process are unwound. This creates multiple cop combinational logic to execute statements in the loop.
- ✓ The synthesis of serial control structures (e.g.if-elsif-elsif-else) can results corresponding serial chain of logic gates to test the conditions.

  In this case, a **case** or **select statement** is to be used to if condition mutually exclusive.

For some synthesis tools, the “coding style” is also an important factor to obtain results.

TEST BENCHES

A **test bench** specifies a sequence of inputs to be applied by the simulator to an
based design, such as a VHDL entity.

The entity being tested is often called the ***unit under test (UUT)***

- ✓ Like all VHDL programs, there is an entity declaration, but notice that it h
  inputs and outputs.
- ✓ The inputs to the UUT are created within the test bench and outputs are observ
  the simulator.
- ✓ The architecture definition makes a component declaration for the UUT, the U
  instantiated using local signals for its inputs and outputs.
- ✓ A process with no sensitivity list starts at simulation time ‘0’. Applies new
  combinations to the UUT at specified times.

UNIT-III

COMBINATIONAL LOGIC DESIGN PRACTICES

ption of basic structures like Decoders, Encoders, Comparators, Multiplexers (74 –series MSI); Des

ex Combinational circuits using the basic structures; Designing Using combinational PLDs like

PROMs CMOS PLDs; Adders & sub tractors, ALUs, Combinational multipliers; VHDL models

standard building block ICs.

Decoders

decoder is a ***multiple-input, multiple-output logic circuit that con***
***ded* inputs into coded outputs**, where the input and output code
fferent.

he input code generally has fewer bits than the output code, and there
e-to-one mapping from input code words into output code words.

he enable inputs, if present, must be asserted for the decoder to perfor
rmal mapping function.

he most commonly used input code is an n-bit binary code, where an
ord represents one of 2^n different coded values.

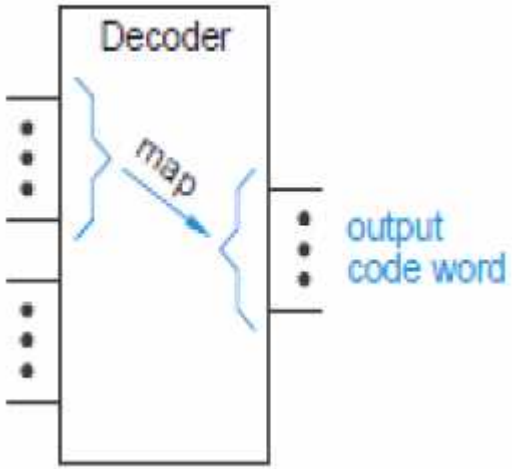
he most commonly used output code is a **1-out-of-m code**, which con
bits, where one bit is asserted at any time.

a 1-out-of-4 code with **active-high outputs**,

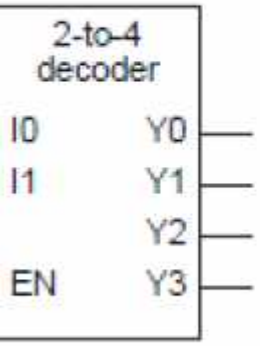
the code words are **0001, 0010, 0100, and 1000**.

With **active-low outputs**, the code words are **1110, 1101, 1011, and 0**

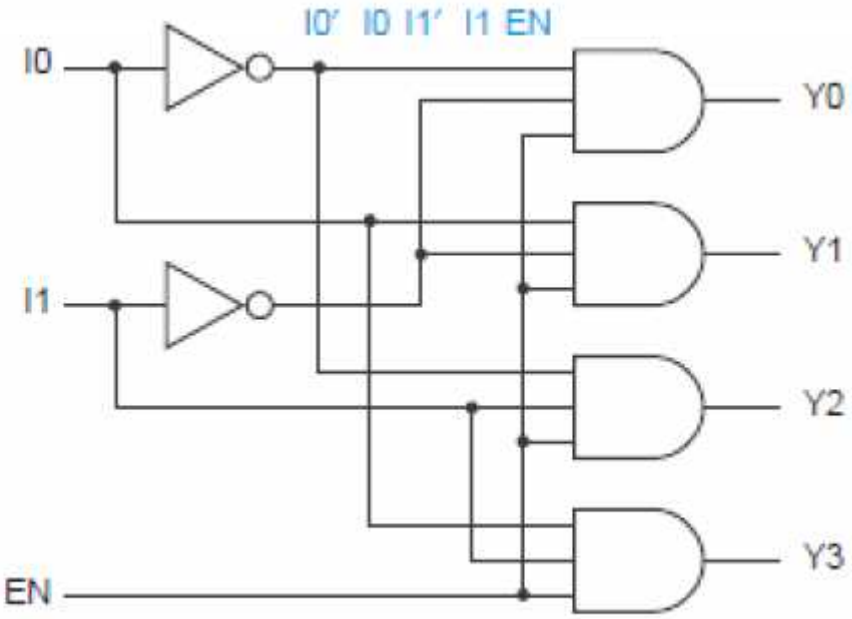
Decoder Circuit Structure and its truth table



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



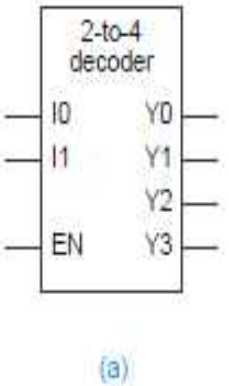
(a)



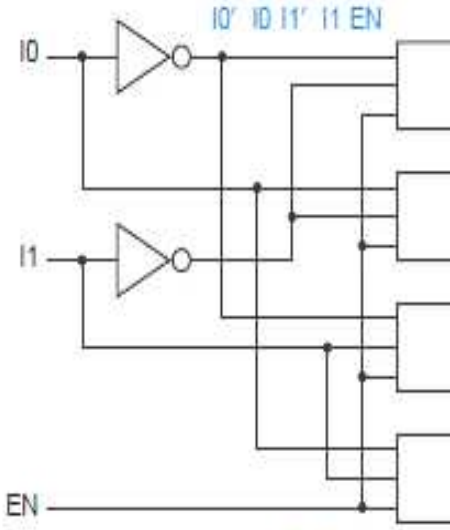
(b)

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

VHDL Program



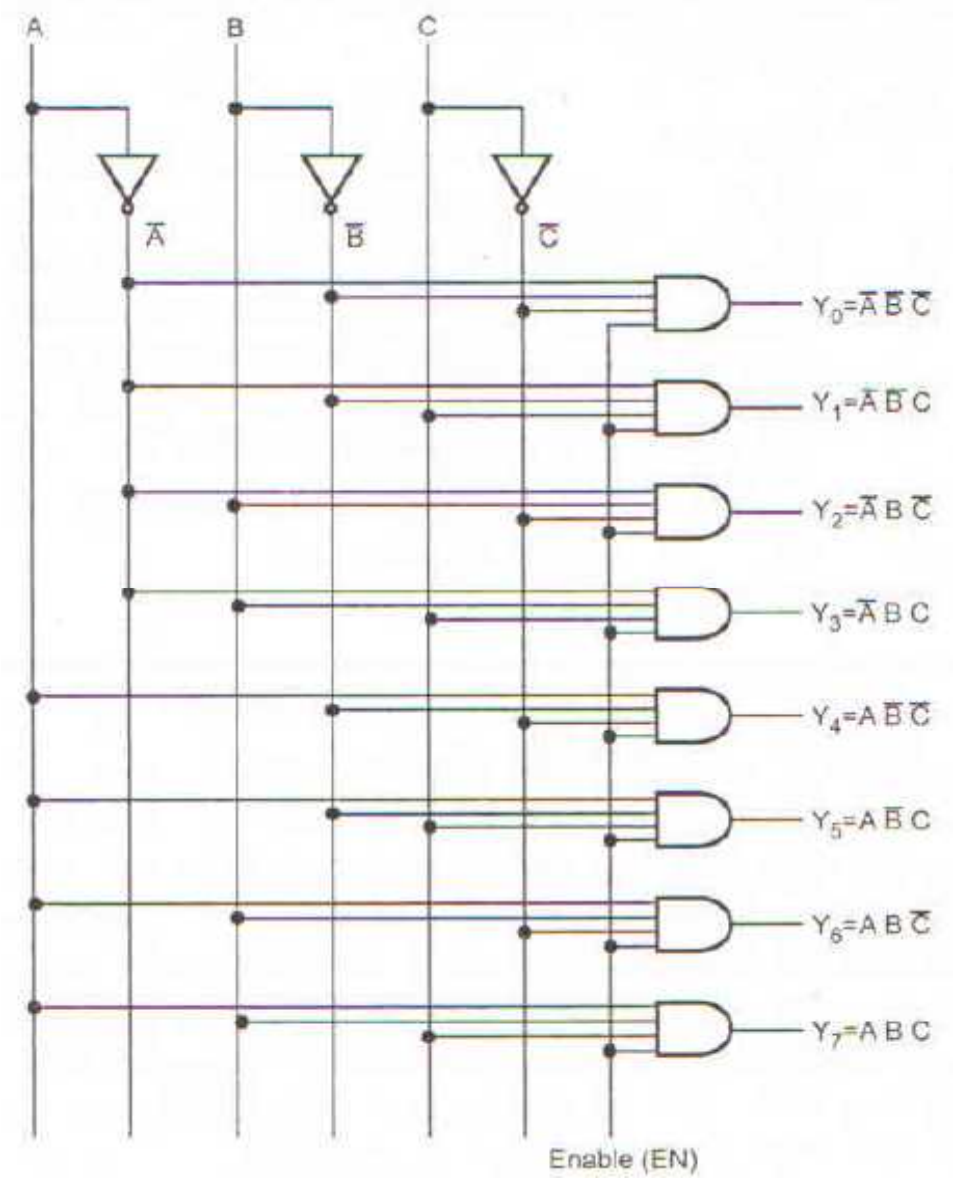
(a)



(b)

```
y IEEE;
EE.std_logic_1164.all;

 V2to4dec is
 (I0, I1, EN: in STD_LOGIC;
  Y0, Y1, Y2, Y3: out STD_LOGIC );
to4dec;

ecture V2to4dec_s of V2to4dec is
al NOTI0, NOTI1: STD_LOGIC;
onent inv port (I: in STD_LOGIC; O: out STD_LOGIC ); end component;
onent and3 port (I0, I1, I2: in STD_LOGIC; O: out STD_LOGIC ); end component;

 inv port map (I0,NOTI0);
 inv port map (I1,NOTI1);
 and3 port map (NOTI0,NOTI1,EN,Y0);
 and3 port map (   I0,NOTI1,EN,Y1);
 and3 port map (NOTI0,   I1,EN,Y2);
 and3 port map (   I0,   I1,EN,Y3);
to4dec_s;
```

The 3-to-8 Decoder



3x8 decoder logic diagram

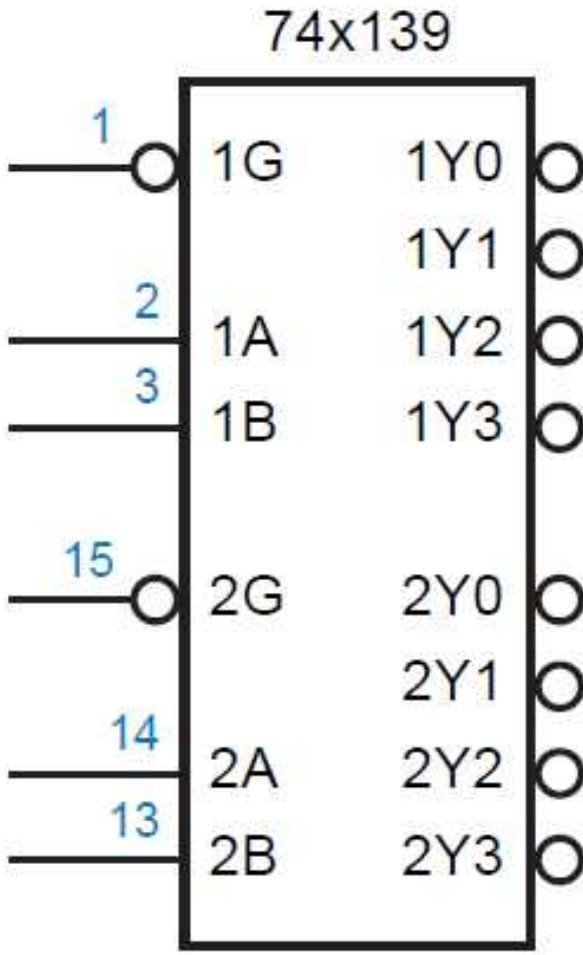
The 74x139 Dual 2-to-4 Decoder

o independent and identical **2-to-4 decoders** are tained in a single MSI *74x139*.

om the figure it is notice that the outputs and the ble input of the ’139 are **active-low**.

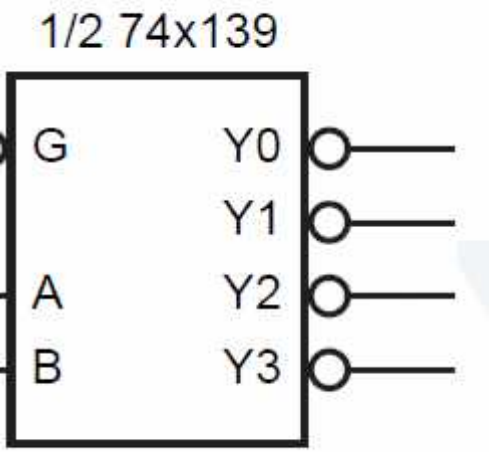
st MSI decoders were originally designed with ve-low outputs,.

ce TTL inverting gates are generally faster than inverting ones.



74x139 decoder logic sy

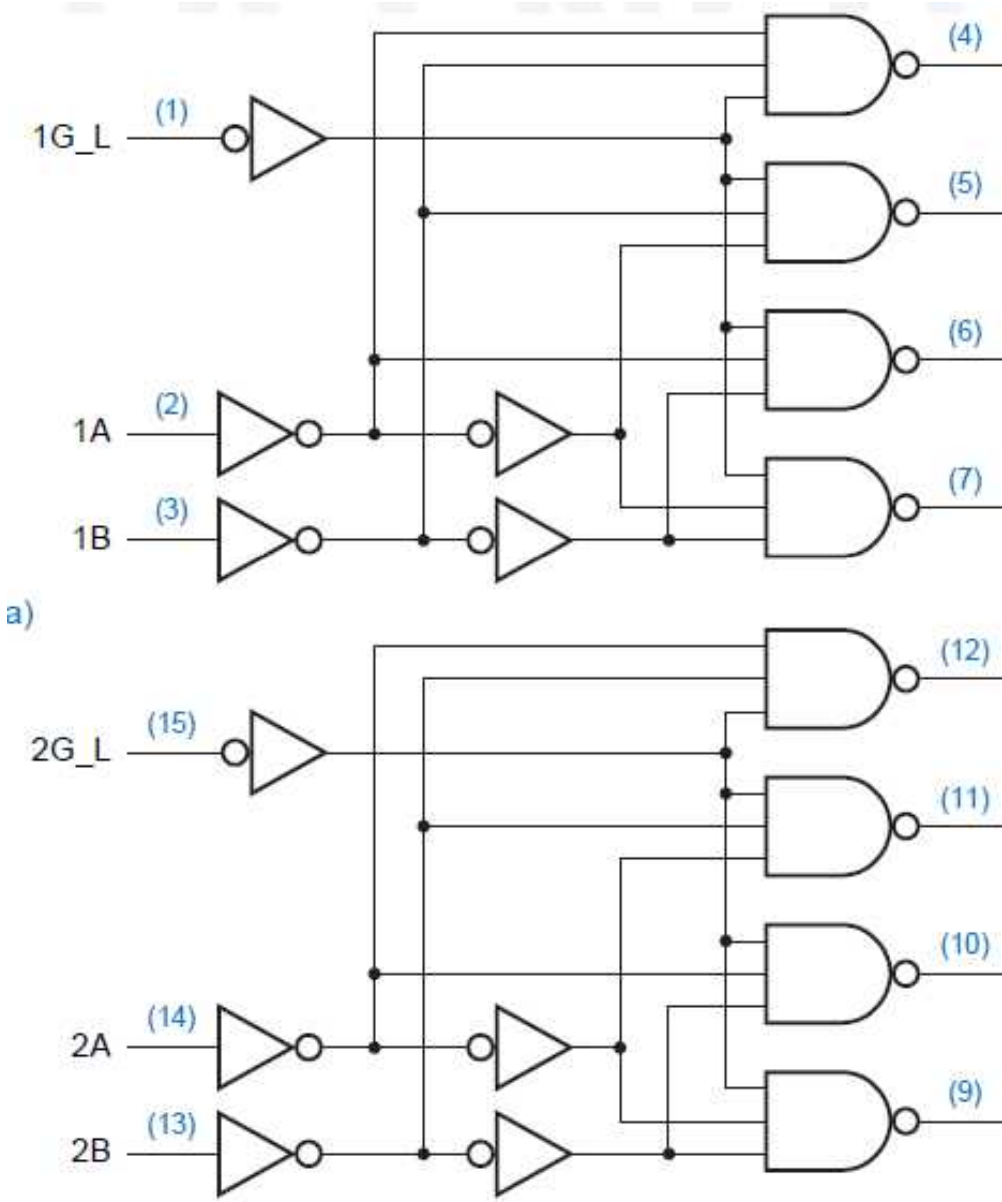
The 74x139 Dual 2-to-4 Decoder



gic symbol for one decoder

uts		Outputs			
B	A	Y3_L	Y2_L	Y1_L	Y0_L
x	x	1	1	1	1
0	0	1	1	1	0
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	1	1	1

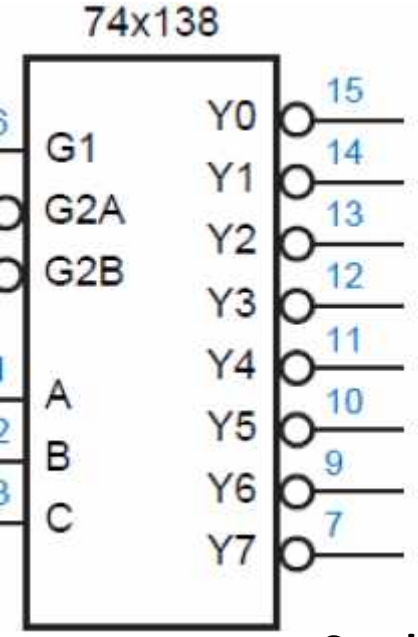
tional table



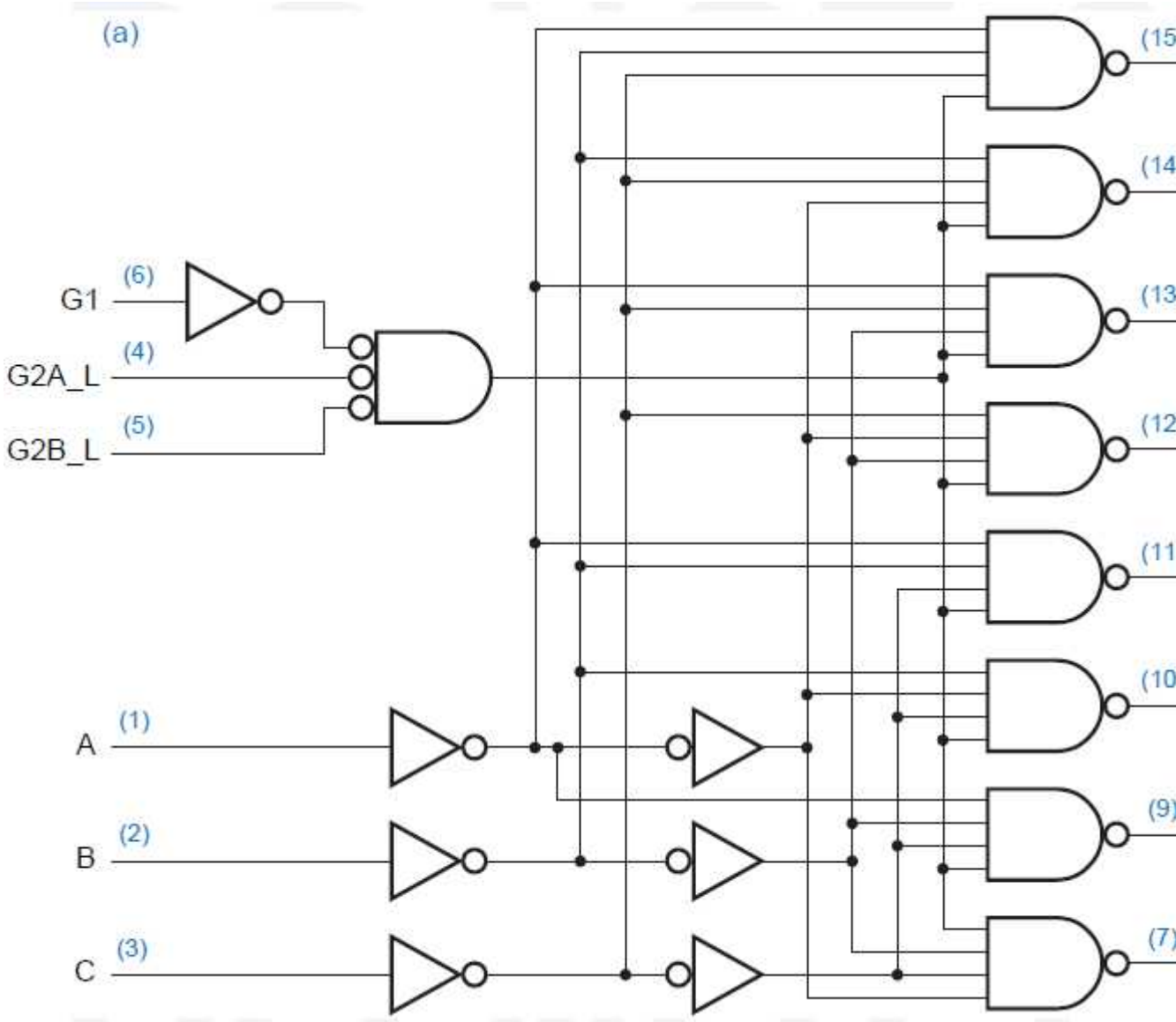
Logic diagram for 74x139 decoder

The 74x138 3-to-8 Decoder

74x138 is a commercially
able MSI 3-to-8 decoder.
the 74x139, the 74x138 has
e-low outputs, and it has
e enable inputs
/G2A, /G2B), all of which
be asserted for the
ted output to be asserted.



Symbol of 74x138 decoder



Logic diagram of 74x13

The 74x138 3-to-8 Decoder-VHDL Program

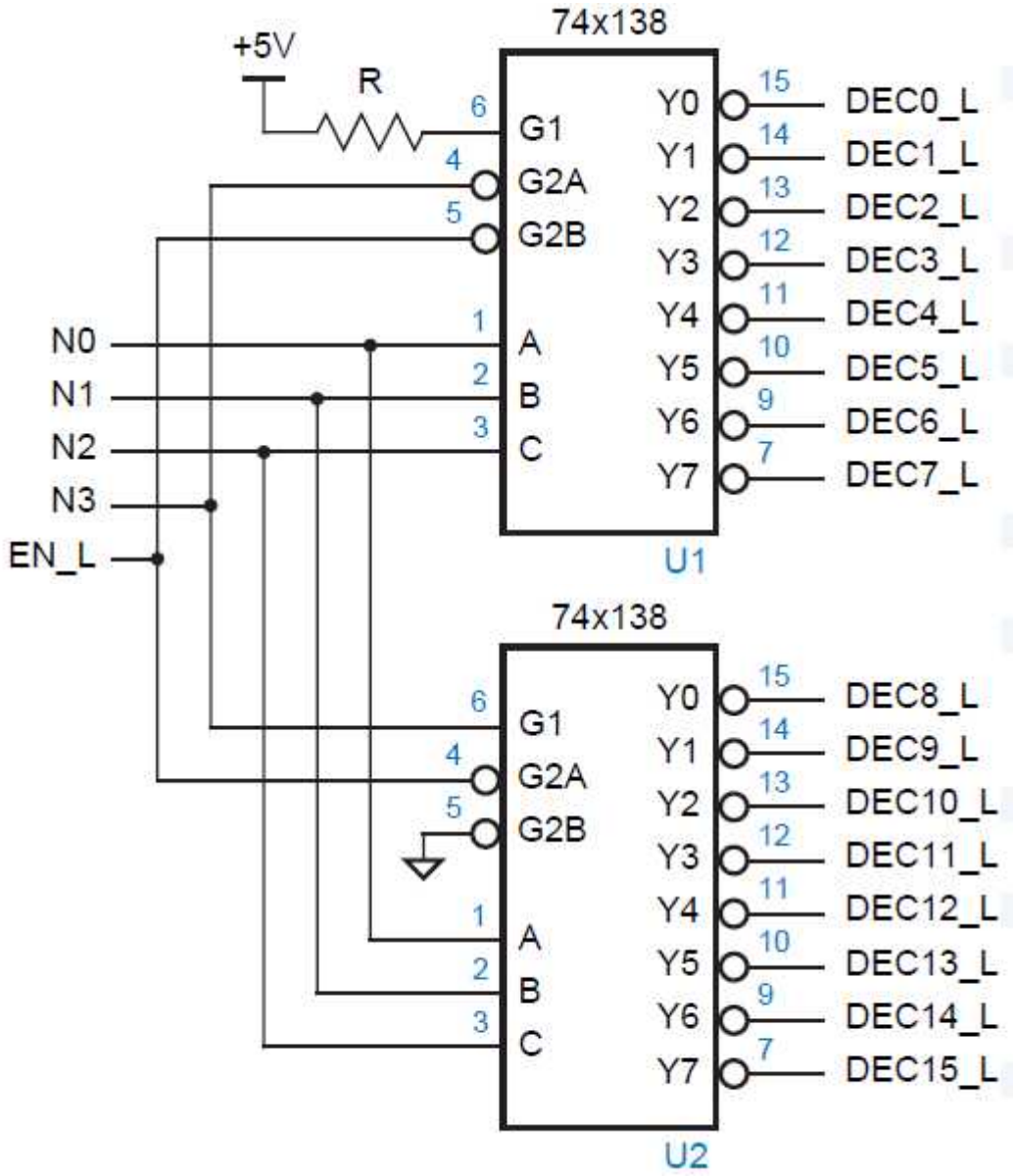
w model

```
y IEEE;
EE.std_logic_1164.all;

 V74x138 is
rt (G1, G2A_L, G2B_L: in STD_LOGIC;      -- enable inputs
  A: in STD_LOGIC_VECTOR (2 downto 0);   -- select inputs
  Y_L: out STD_LOGIC_VECTOR (0 to 7) );  -- decoded outputs
74x138;

ecture V74x138_a of V74x138 is
al Y_L_i: STD_LOGIC_VECTOR (0 to 7);

 A select Y_L_i <=
1111111" when "000",
0111111" when "001",
1011111" when "010",
1101111" when "011",
1110111" when "100",
1111011" when "101",
1111101" when "110",
1111110" when "111",
1111111" when others;
<= Y_L_i when (G1 and not G2A_L and not G2B_L)='1' else "11111111";
4x138_a;
```

Behavioral mod

```
Process(G1,G2A_L,G2B_L,A)
Begin
case A is
when "000" => Y_L_i <= “011111
when "001" => Y_L_i <= “101111
when "010" => Y_L_i  <= “110111
when "011" => Y_L_i  <= “111011
when "100" => Y_L_i <= “111101
when "101" => Y_L_i  <= “111110
when "110" => Y_L_i  <= “111111
when "111" => Y_L_i  <= “111111
when others => Y_L_i <= “111111
end case;
if (G1 and not G2A_L and not G2
      Y_L <= Y_L_i;
else
      Y <= “11111111";
end if;
end process;
```

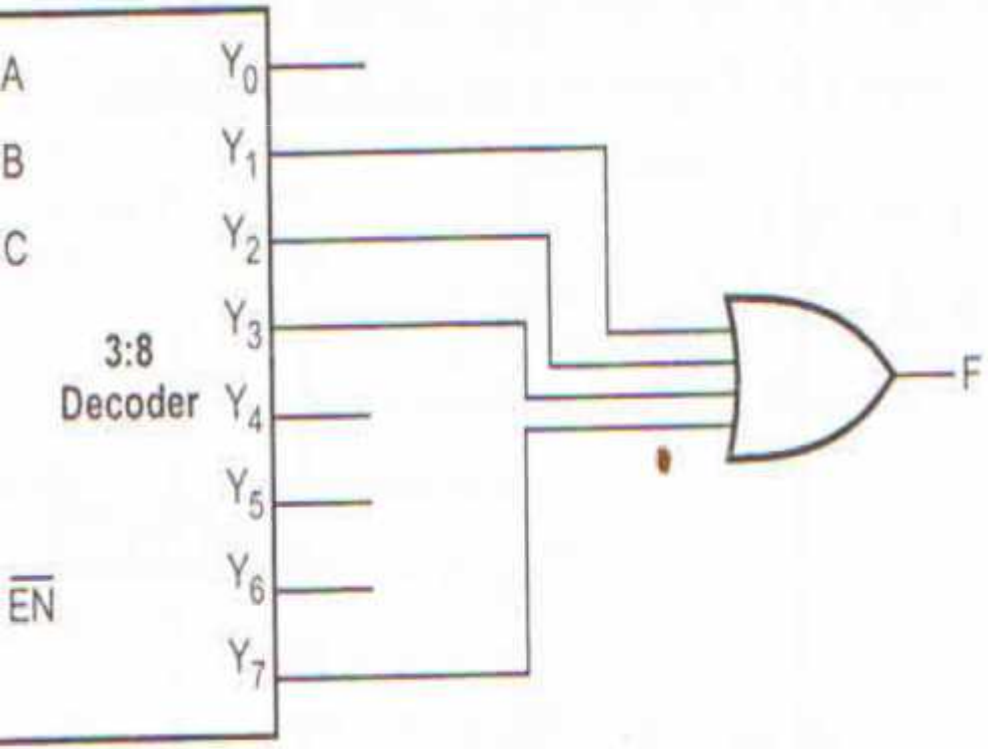
Design of a 4-to-16 decoder using 74x138s



Realization of multiple output function using decoder

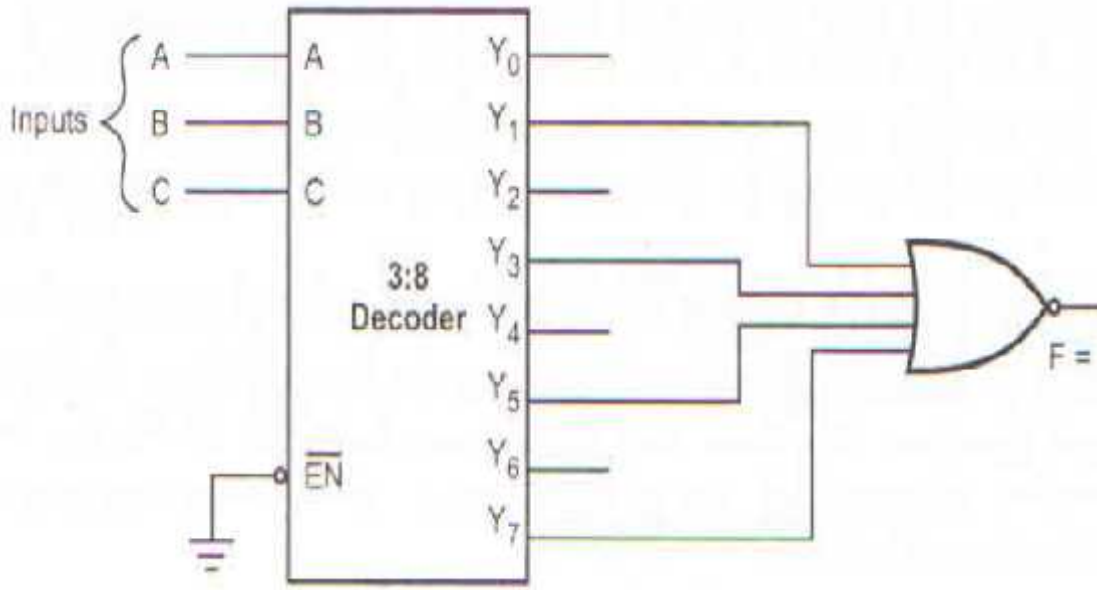
tive High Outputs
nction Implementation

$\sum M(1, 2, 3, 7)$



Active High Outputs
POS Function Implementatio

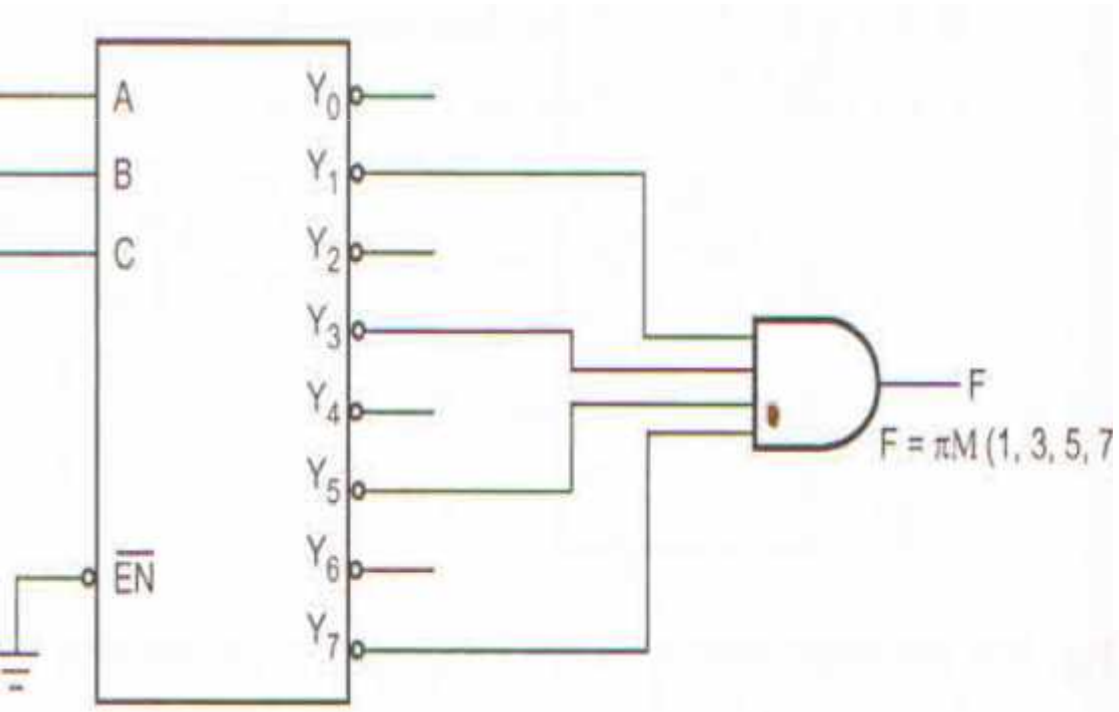
$f = \pi M(1, 3, 5, 7)$



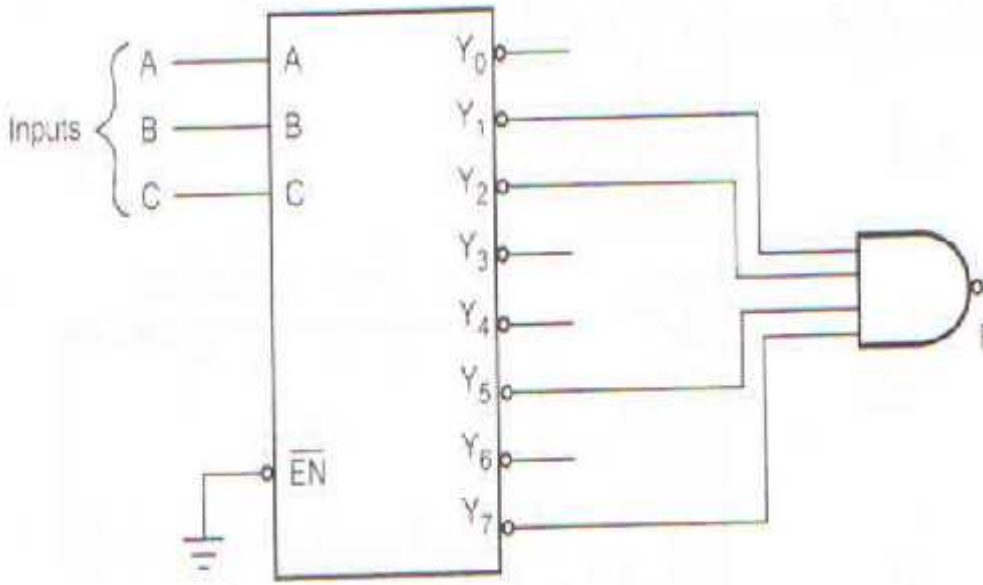
Realization of multiple output function using decoder

ctive Low Outputs
nction Implementation

$M(1,3,5,7)$



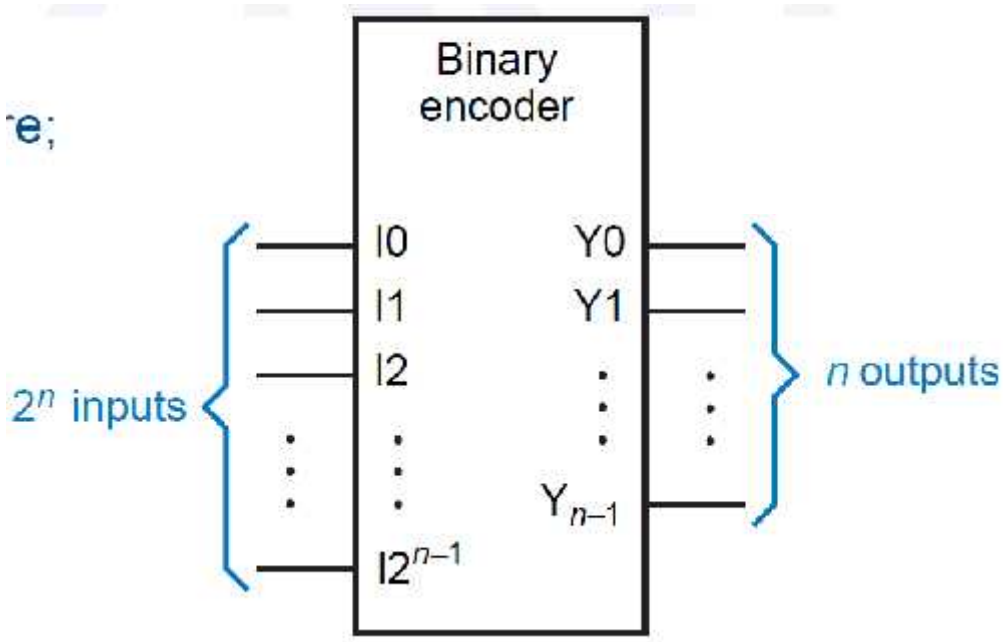
Active Low Outputs
SOP Function Implementatio



Encoders

f the device's output code has fewer bits than the input code, the dev
ually called an encoder.

obably the simplest encoder to build is a **2^n-to-n** or binary encoder.



Logic Symbol

8X3 Encoder (octal to binary encoder)

Inputs								Outputs		
I7	I6	I5	I4	I3	I2	I1	I0	Y2	Y1	Y0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

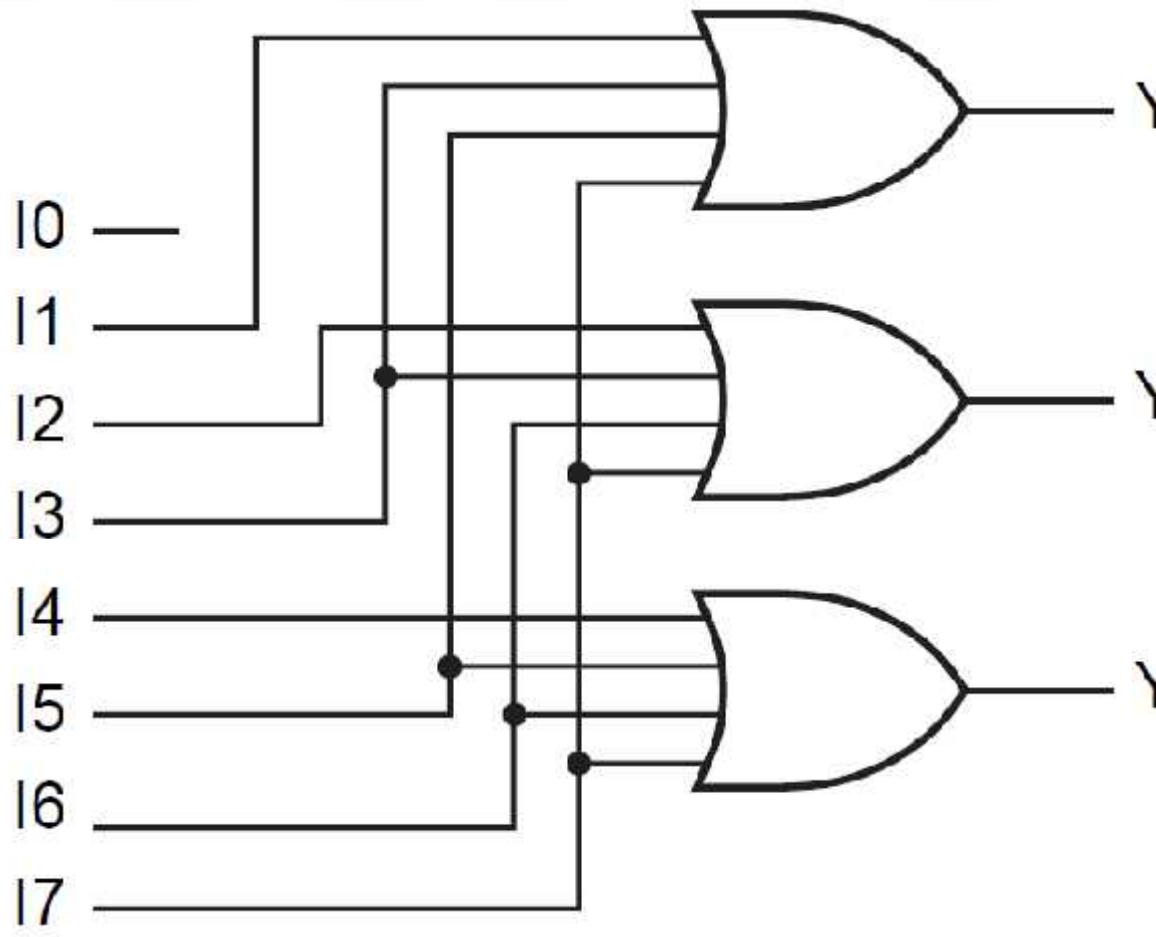
8X3 Encoder truth table

Encoders-8X3 Encoder

$$0 = I1 + I3 + I5 + I7$$

$$= I2 + I3 + I6 + I7$$

$$2 = I4 + I5 + I6 + I7$$



Logic diagram

Priority Encoders

riority encoder is an encoder circuit that includes the priority function.
his encoder, of two or more inputs are equal to '1' at the same time, the input hav
highest priority will take precedence.

iority Encoder

e the priority is increasing
m **D0 to D3**, I.e. **D3 is the**
nest priority.
output V (a valid output
cator) indicates one or
re of the inputs are equal to
.
l the inputs are zero, V is
al to zero, and other two
puts of the encoder are not
d

Inputs				Outputs	
D_0	D_1	D_2	D_3	Y_1	Y_0
0	0	0	0	X	X
1	0	0	0	0	0
X	1	0	0	0	1
X	X	1	0	1	0
X	X	X	1	1	1

4x2 priority encoder truth table

For Y_1

D_0D_1 \ D_2D_3	01	11	10
	1	1	1
	1	1	1
	1	1	1
	1	1	1

$Y_1 = D_2 + D_3$

For Y_0

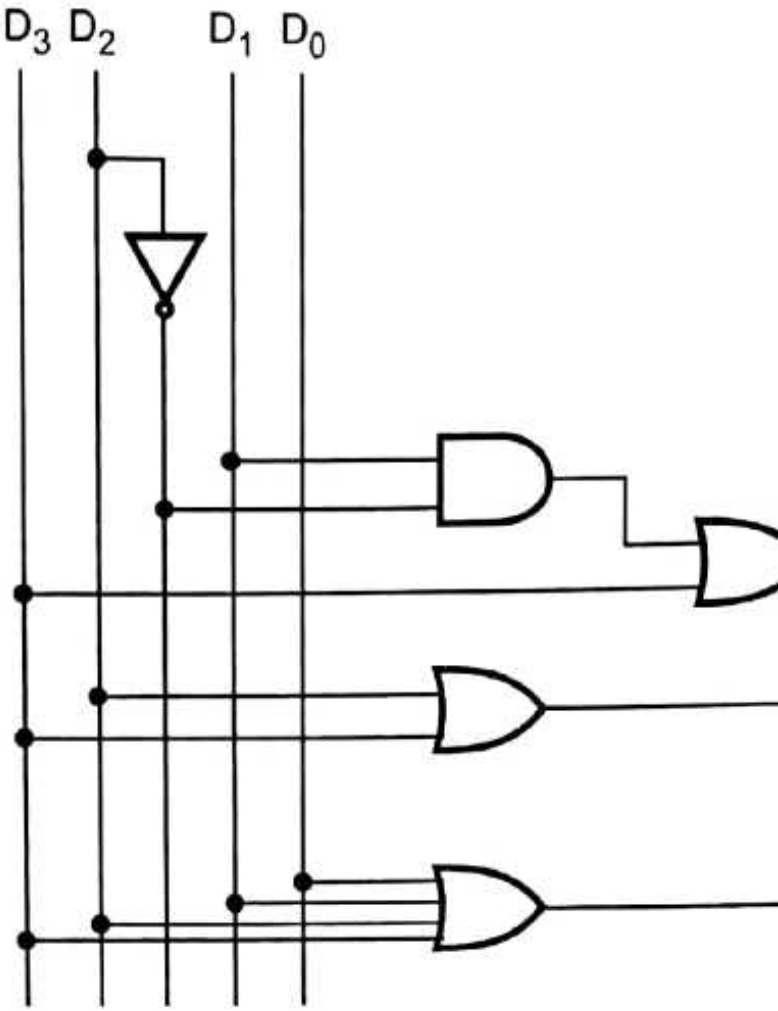
D_0D_1 \ D_2D_3	00	01	11	10
00	X	1	1	0
01	1	1	1	0
11	1	1	1	0
10	0	1	1	0

$Y_0 = D_3 + D_1\overline{D}_2$

For V

D_0D_1 \ D_2D_3	00	01	11	10
00	0	1	1	1
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

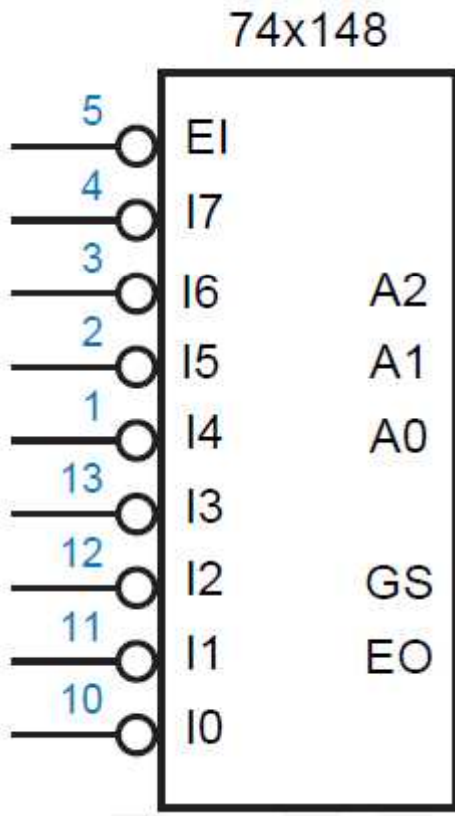
$V = D_0 + D_1 + D_2 + D_3$



Logic diagram

The 74x148 Priority Encoders

4x148 encoder accepts data from **eight active low inputs** and provides a binary
resentation on **the three active low outputs**.
ut **I0 has the least priority** and input **I7 has the highest priority**.
s the active low enable input.
is group signal, it is asserted when the device is enabled and
e or more of the inputs to the encoder are active.
is the enable output. It is an active low signal, can be used to
cade several priority encoder devices



8x3 priority encoder symbol

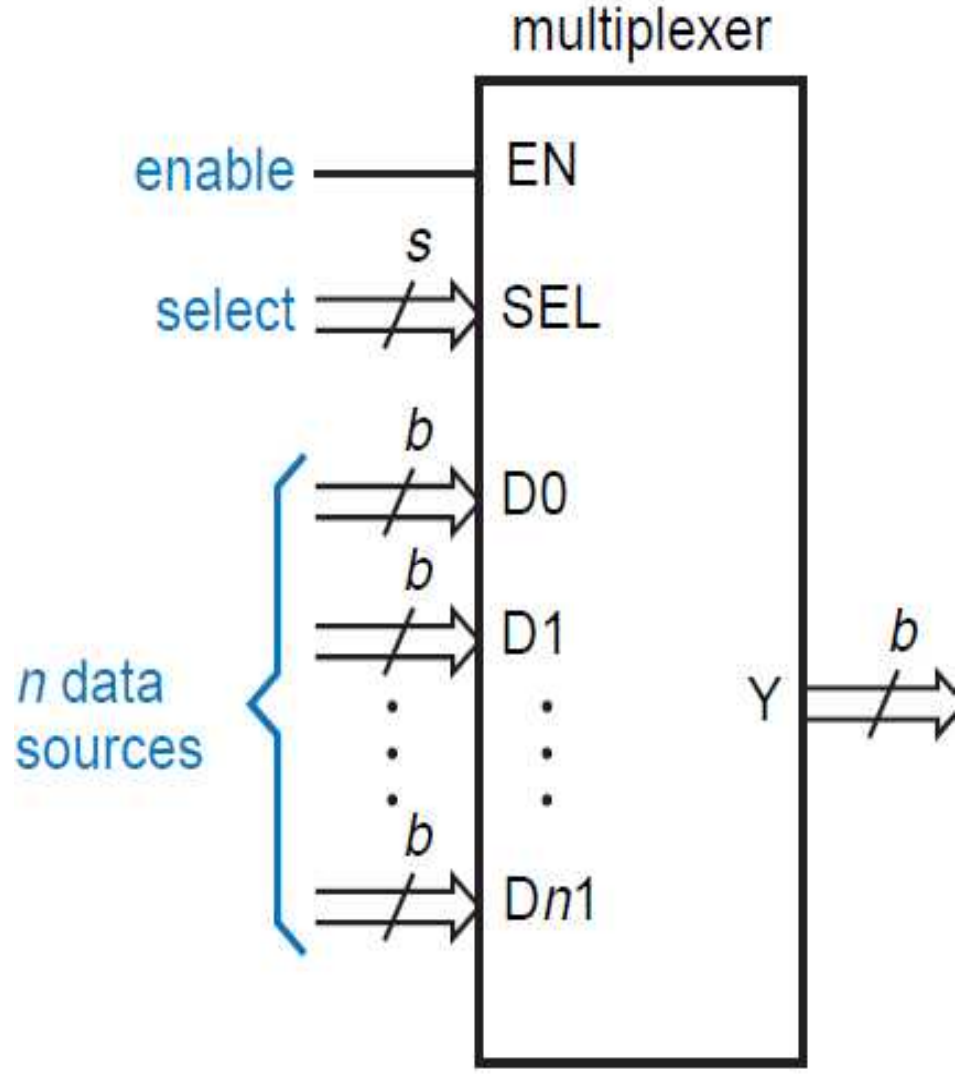
The 74x148 Priority En

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

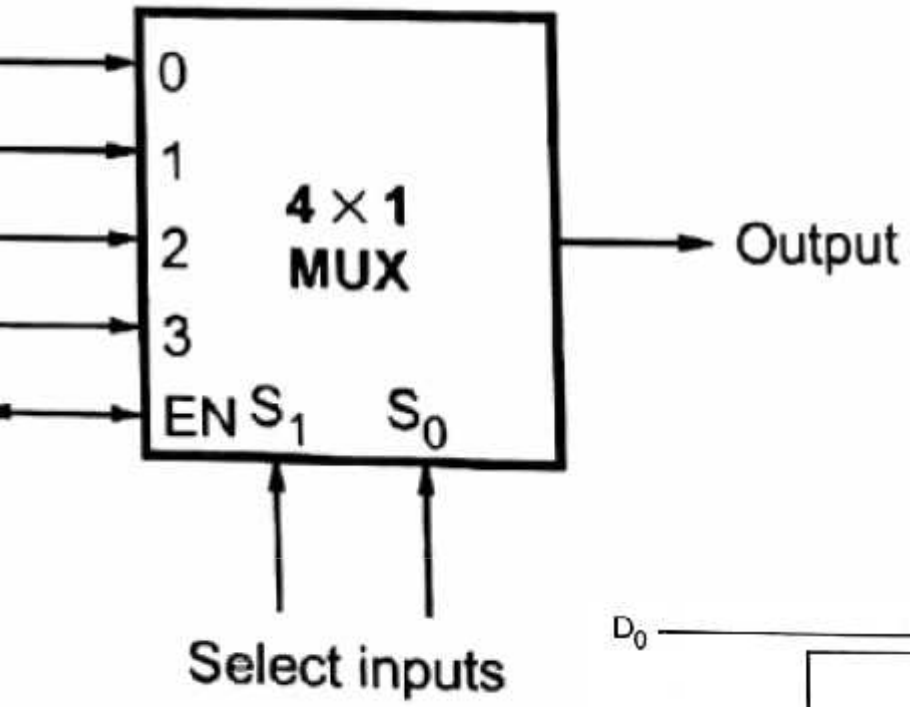
8x3 priority truth table

MULTIPLEXERS

multiplexer is a digital switch—it connects
ta from one of *n* sources to its output.
e basic multiplexer has several data input
es and a single output line.
e selection of a particular input line is
ntrolled by a set of selection lines.
nerally, there are 2^n input lines and n
ection lines whose bit combinations
termine which input is selected.
ultiplexer is **many into one**



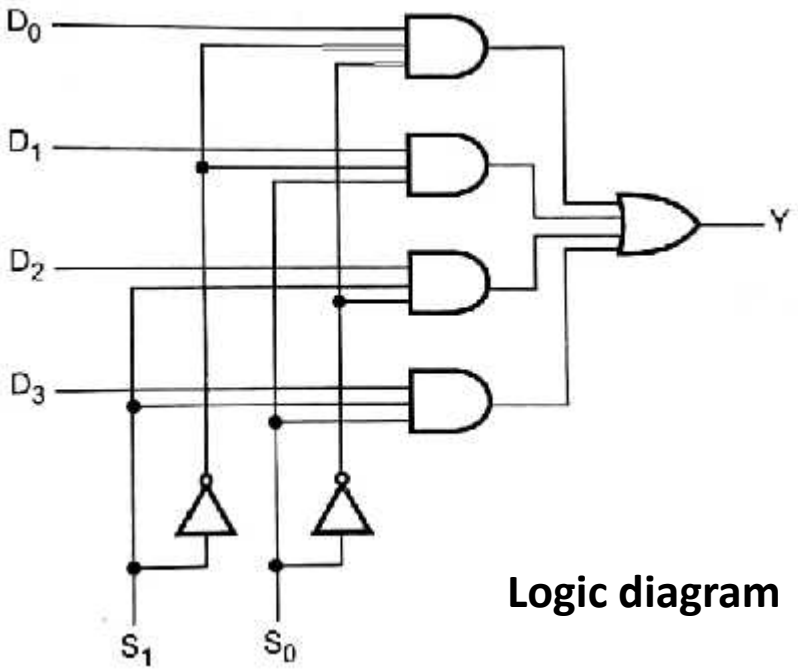
4x1 MULTIPLEXERS



ymbol

S_1	S_0	Y
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

Truth table



Logic diagram

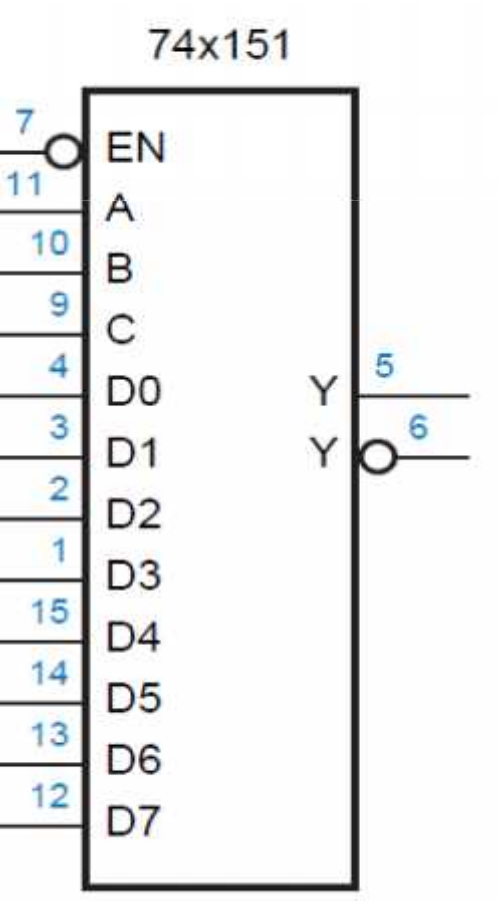
74x151 MULTIPLEXERS

he 74x151 is the **8x1 multiplexer**

provides **two outputs**, one output is **active low** and another output is **tive high**.

has the **three selection lines** (A,B,C) which select one of the eight inputs.

provided the **active low enable input**

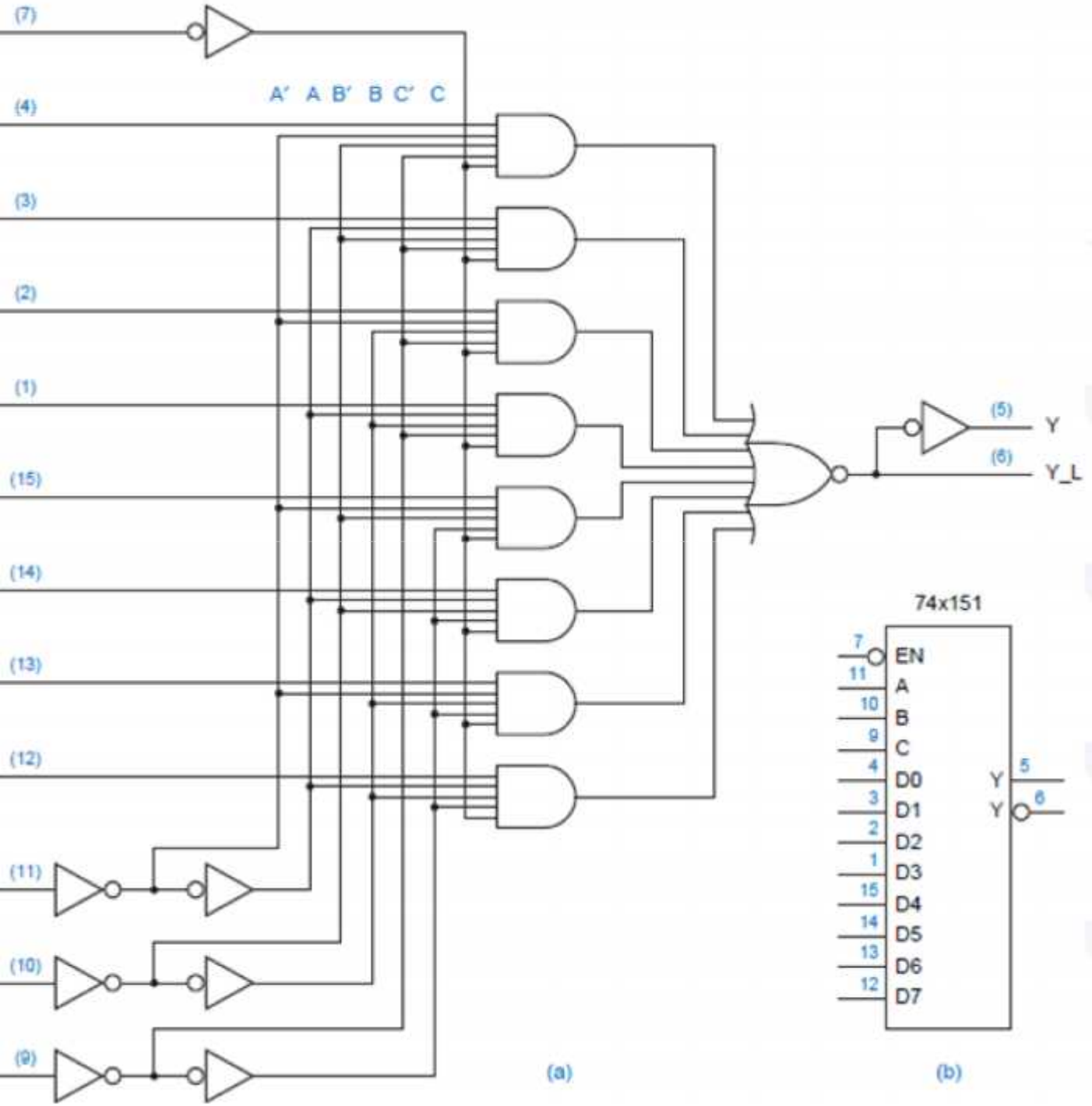


symbol

Inputs				Outputs	
EN_L	C	B	A	Y	Y_L
1	x	x	x	0	1
0	0	0	0	D0	D0′
0	0	0	1	D1	D1′
0	0	1	0	D2	D2′
0	0	1	1	D3	D3′
0	1	0	0	D4	D4′
0	1	0	1	D5	D5′
0	1	1	0	D6	D6′
0	1	1	1	D7	D7′

Trut

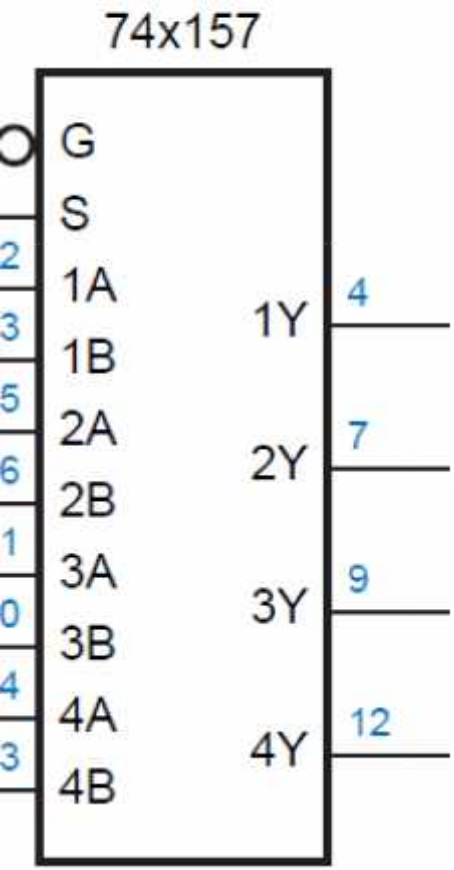
74x151 MULTIPLE



Logic diag

74x157 MULTIPLEXERS (Quad Two input Multiplexer)

he IC 74x157 is the **Quad two input multiplexer** which selects four bits of ata from two sources under the control of a common select input. provided the **active High enable input,** When enable is zero, all of the utputs are forced to low regardless of all other input conditions.

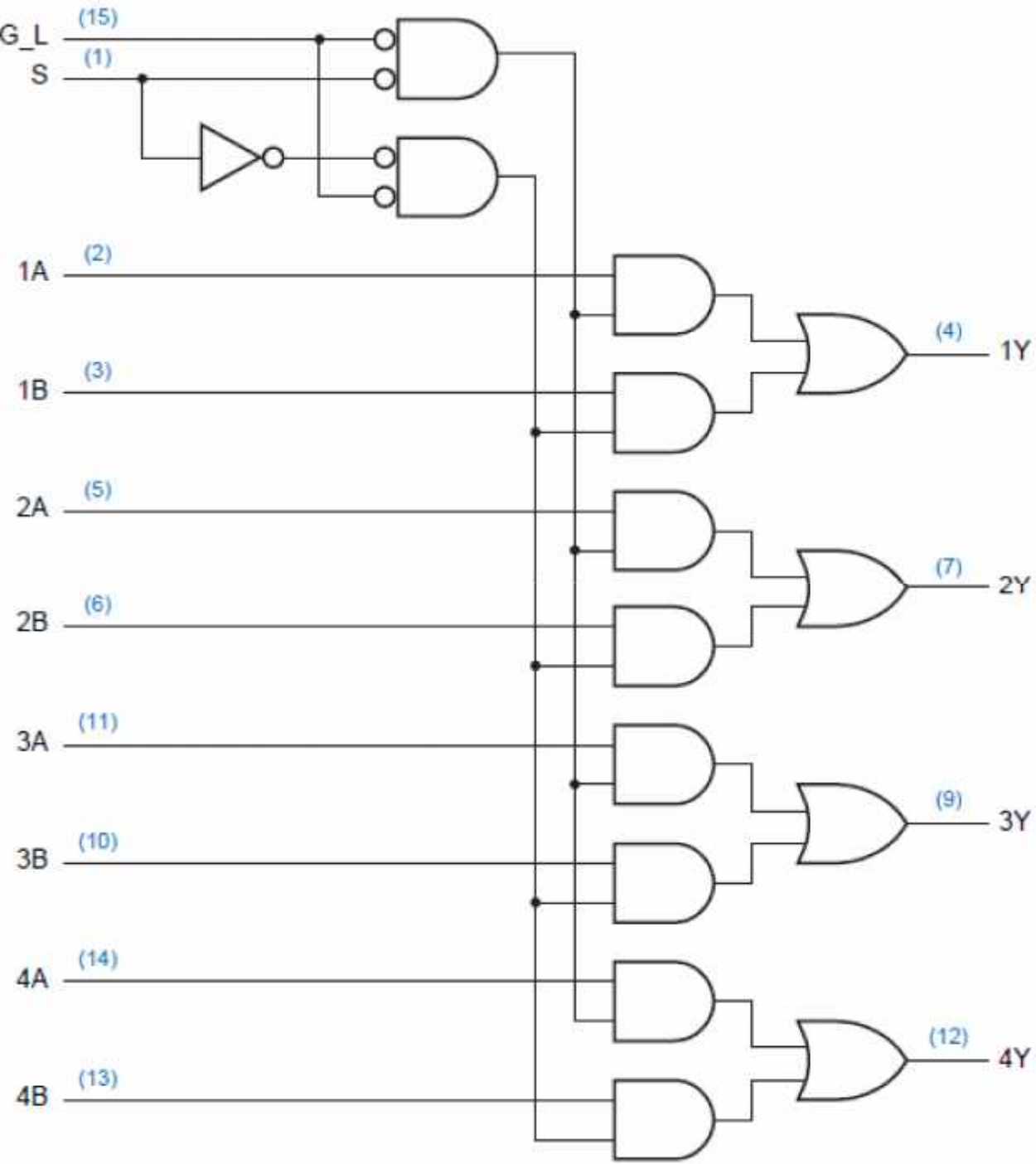


symbol

Inputs		Outputs			
G_L	S	1Y	2Y	3Y	4Y
1	x	0	0	0	0
0	0	1A	2A	3A	4A
0	1	1B	2B	3B	4B

Truth table

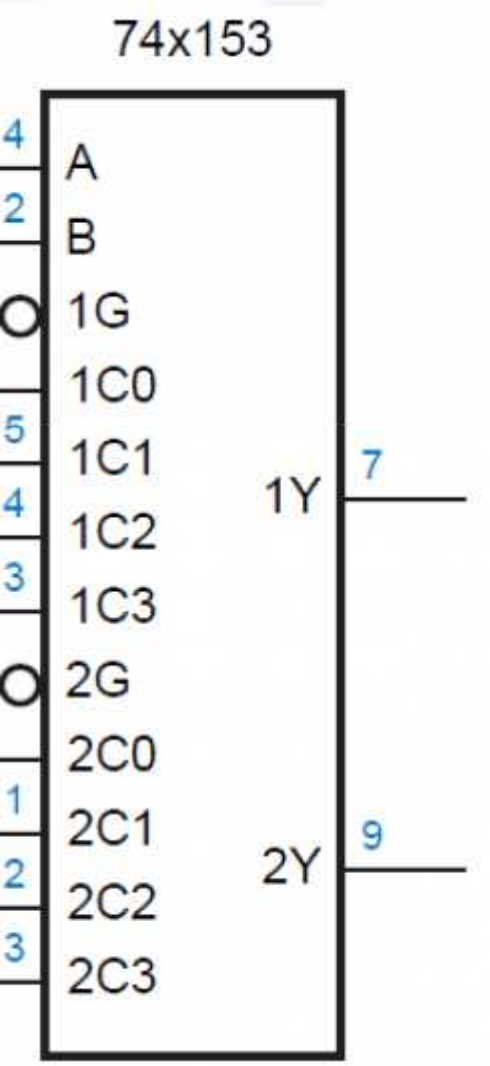
74x157 MULTIPLEX



Logic diag

74x153 MULTIPLEXERS (Dual 4x1 Multiplexer)

he IC 74x153 is the **dual 4x1 multiplexer.**



symbol

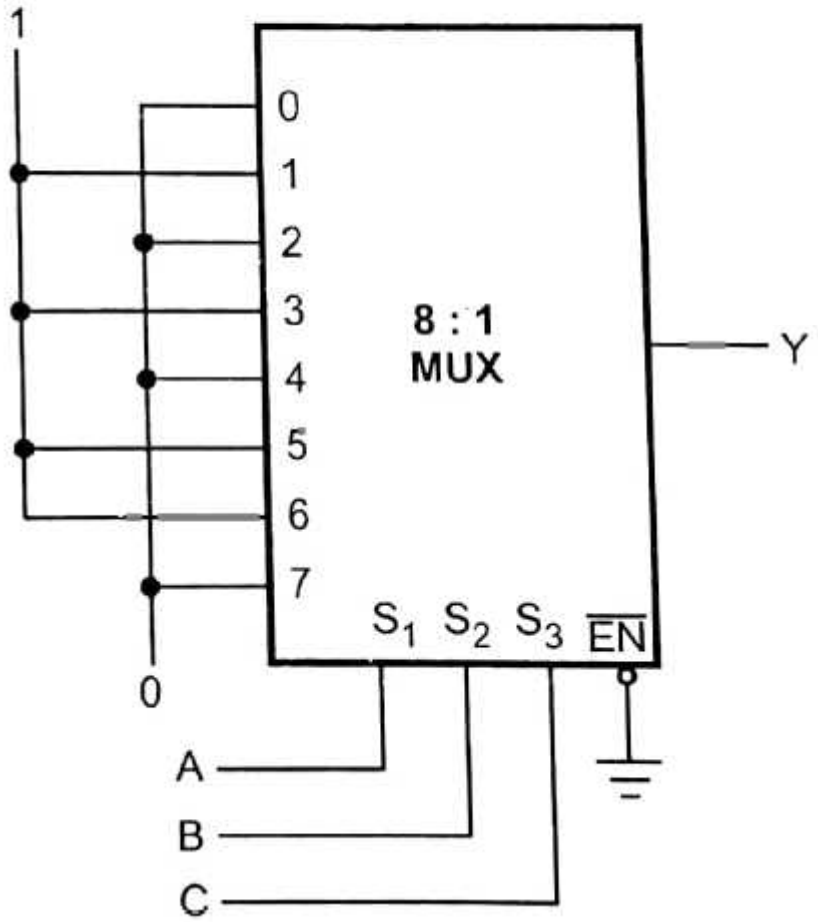
Inputs				Outpu	
1G_L	2G_L	B	A	1Y	
0	0	0	0	1C0	2
0	0	0	1	1C1	2
0	0	1	0	1C2	2
0	0	1	1	1C3	2
0	1	0	0	1C0	
0	1	0	1	1C1	
0	1	1	0	1C2	
0	1	1	1	1C3	
1	0	0	0	0	2
1	0	0	1	0	2
1	0	1	0	0	2
1	0	1	1	0	2
1	1	x	x	0	

Truth table

Implementation of combinational logic using MUXs

ement the following Boolean function using 8x1 multiplexer

$$F(A, B, C) = \sum m(1, 3, 5, 6)$$



Implementation of combinational logic using MUXs

ement the following Boolean function using 4x1 multiplexer

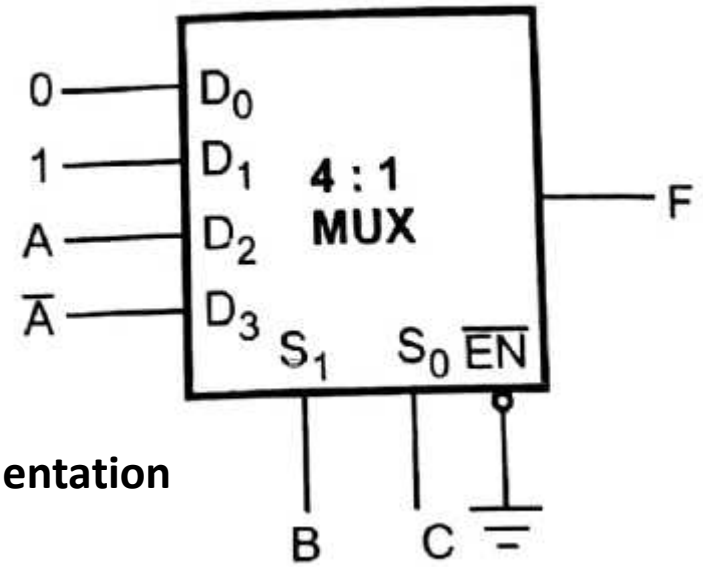
$$F(A, B, C) = \sum m(1, 3, 5, 6)$$

linterm	A	B	C	F
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	0

(a) Truth table

	D_0	D_1	D_2	D_3	
$\overline{A}$	0	①	2	③	row 1
A	4	⑤	⑥	7	row 2
	0	1	A	$\overline{A}$	

Implementation table



Multiplexer Implementation

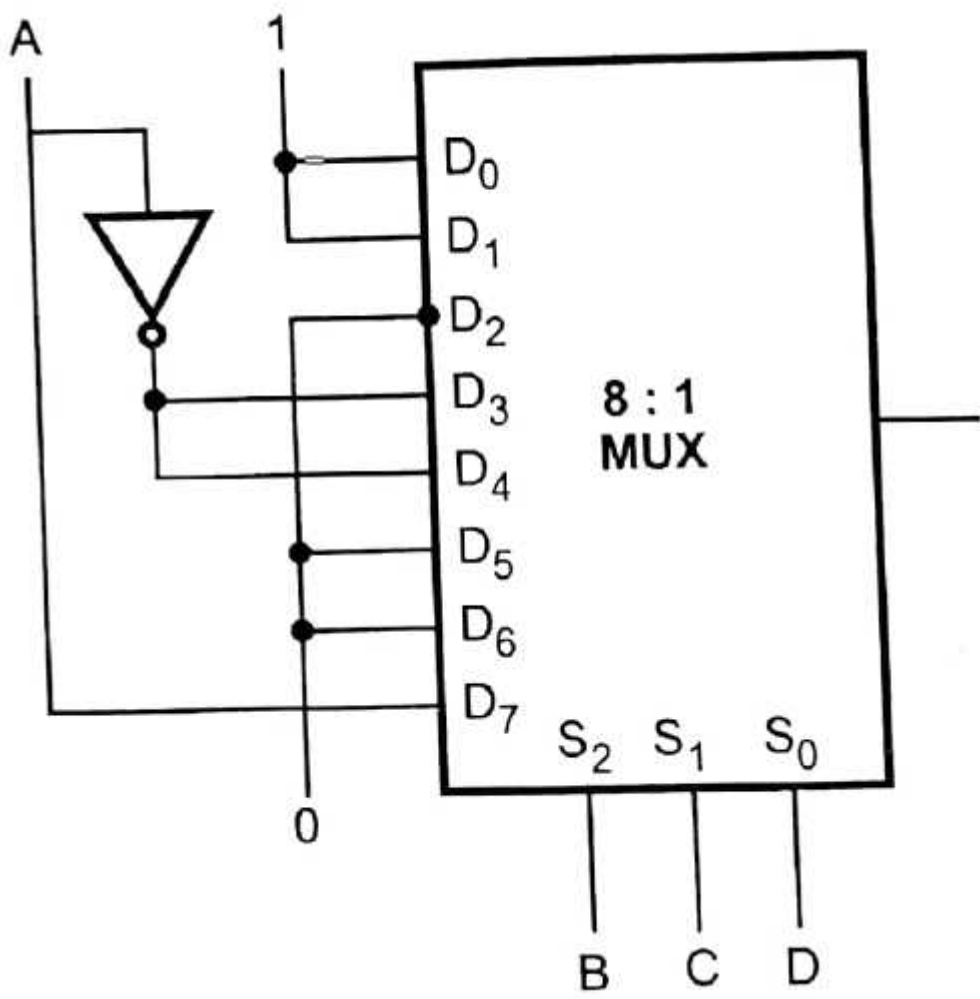
Implementation of combinational logic using MUXs

ement the following Boolean function using 8x1 multiplexer

$, Q, R, S) = \Sigma\, m\ (0, 1, 3, 4, 8, 9, 15)$

	D_1	D_2	D_3	D_4	D_5	D_6	D_7
	(1)	2	(3)	(4)	5	6	7
	(9)	10	11	12	13	14	(15)
	1	0	$\overline{A}$	$\overline{A}$	0	0	A

Implementation table



Multiplexer Implementat

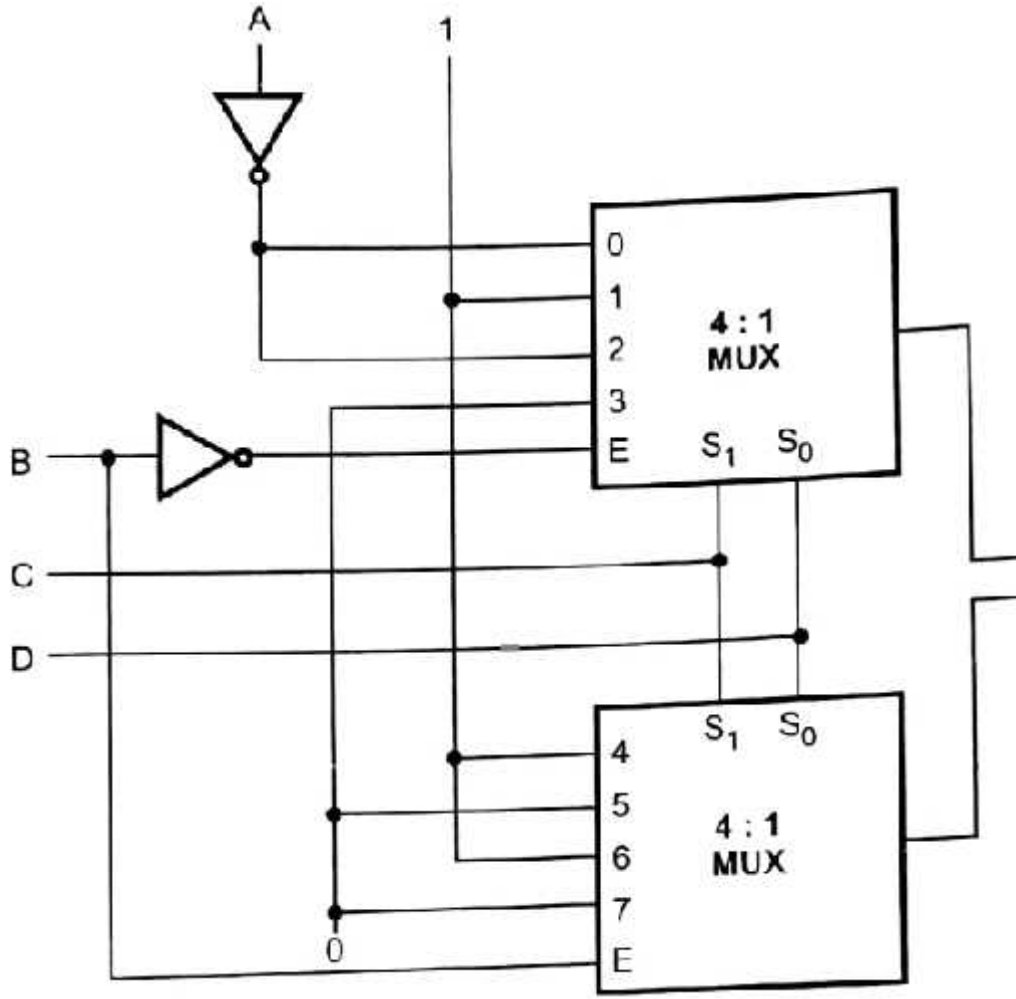
Implementation of combinational logic using MUXs

lement the following Boolean function using 4x1 multiplexer

$$B, C, D) = \sum m\,(0, 1, 2, 4, 6, 9, 12, 14)$$

D_1	D_2	D_3	D_4	D_5	D_6	D_7
(1)	(2)	3	(4)	5	(6)	7
(9)	10	11	(12)	13	(14)	15
1	$\bar{A}$	0	1	0	1	0

Implementation table



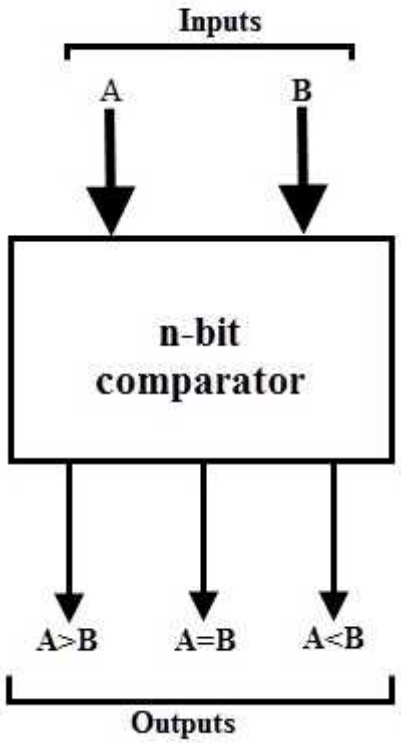
Multiplexer Implementation

COMPARATORS

omparator is a special combinational circuit designed primarily to compare the rel
gnitude of two binary numbers.

re are two main types of **Digital Comparator** available and these are

- **Identity Comparator** – It identifies the equality
  when A = B, either A = B = 1 (HIGH) or A = B = 0 (LOW)
- **Magnitude Comparator** – a Magnitude Comparator is a digital comparator which
  three output terminals.
  one each for **equality, A = B**
  greater than, A > B and
  less than, A < B



Block diagram of n bit comparator

2 BIT COMPARATORS

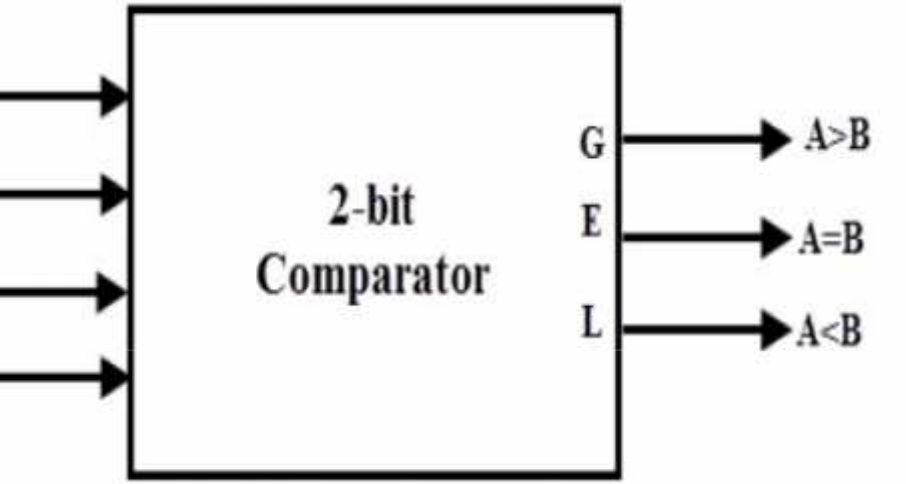


diagram of 2 bit comparator

Inputs				Outputs		
A_1	A_0	B_1	B_0	A>B	A=B	
0	0	0	0	0	1	
0	0	0	1	0	0	
0	0	1	0	0	0	
0	0	1	1	0	0	
0	1	0	0	1	0	
0	1	0	1	0	1	
0	1	1	0	0	0	
0	1	1	1	0	0	
1	0	0	0	1	0	
1	0	0	1	1	0	
1	0	1	0	0	1	
1	0	1	1	0	0	
1	1	0	0	1	0	
1	1	0	1	1	0	
1	1	1	0	1	0	
1	1	1	1	0	1	

Truth table for 2 bit comparator

2 BIT COMPARATORS

$$A>B:\ G = A0\,\overline{B1}\ \ \overline{B0} + A1\,\overline{B1} + A1\ A0\,\overline{B0}$$

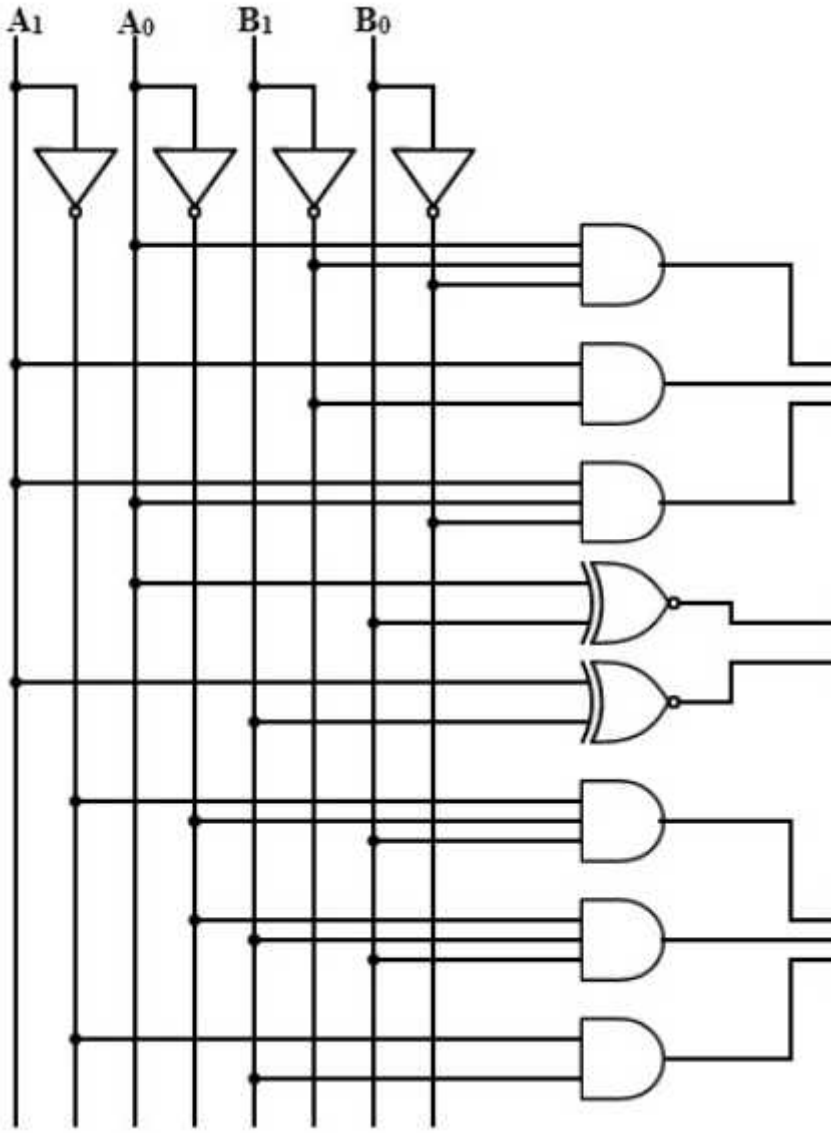
$$= \overline{A1}\ \ \overline{A0}\ \overline{B1}\ \ \overline{B0} + \overline{A1}\ A0\ \ \overline{B1}\ B0 + A1\ A0\ B1\ B0 + A1\,\overline{A0}\ B1\,\overline{B0}$$

$$= \overline{A1}\ \ \overline{B1}\ (\overline{A0}\ \ \overline{B0} + A0\,B0) + A1\,B1\ (A0\,B0 + \overline{A0}\ \ \overline{B0}\ )$$

$$= (A0\,B0 + \overline{A0}\ \ \overline{B0})\ (A1\ B1 + \ \overline{A1}\ \ \overline{B1})$$

$$= (A0 \text{ Ex-NOR } B0)\ (A1 \text{ Ex-NOR } B1)$$

$$A<B:\ L = \overline{A1}\ \ B1 + \overline{A0}\ B1\ B0 + \overline{A1}\ \ \overline{A0}\ B0$$

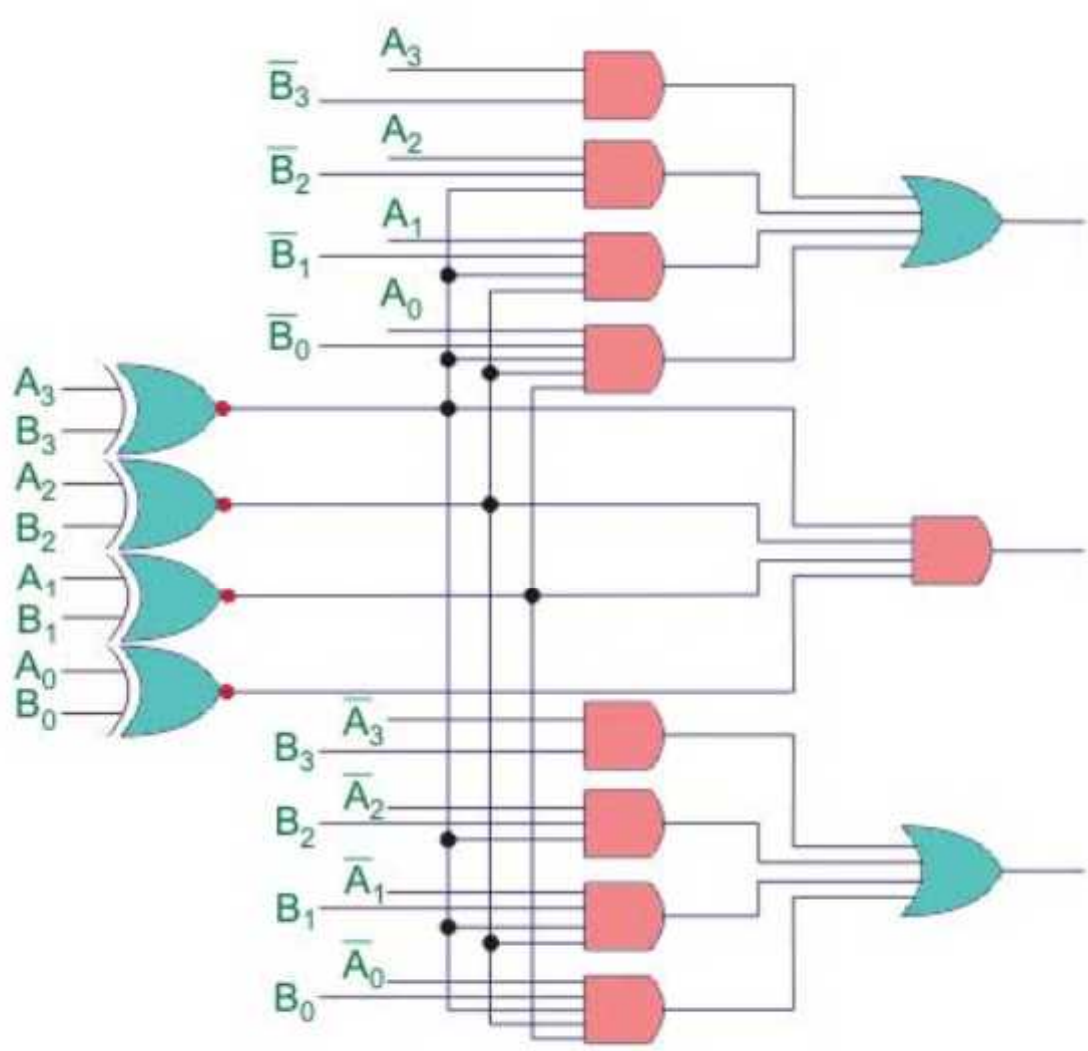


4 BIT COMPARATORS

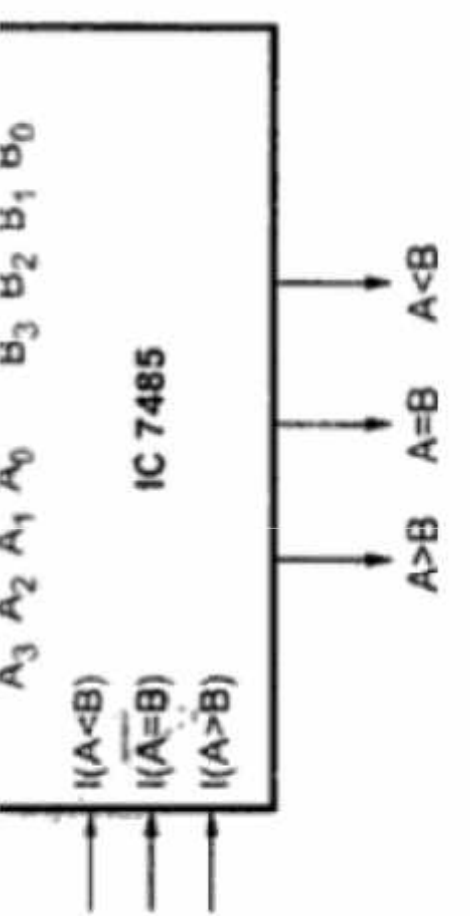
$$+(A_1 \odot B_1)(A_2\overline{B_2}) + (A_1 \odot B_1)(A_2 \odot B_2)A_3\overline{B_3}$$
$$_1 \odot B_1)(A_2 \odot B_2)(A_3 \odot B_3)A_4\overline{B_4} \ldots\ldots (i)$$

$$(A_1 \odot B_1)(\overline{A_2}B_2) + (A_1 \odot B_1)(A_2 \odot B_2)A_3B_3 + (A_1 \odot B_1)(A_2 \odot B_2)$$
$$(A_3 \odot B_3)\overline{A_4}B_4 \ldots\ldots (ii)$$

$$\odot B_1)(A_2 \odot B_2)(A_3 \odot B_3)(A_4 \odot B_4) \ldots\ldots (iii)$$



The IC 7485 (4 BIT COMPARATOR)



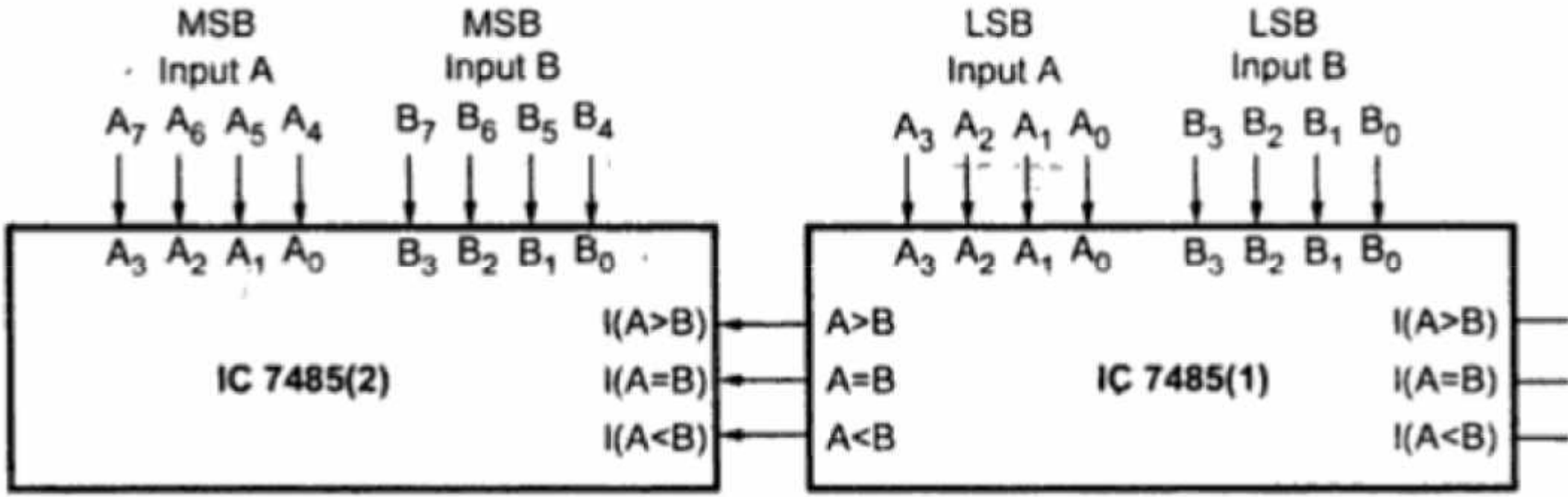
Symbol

Comparing inputs								Cascading inputs			Outpu	
A_3,	B_3	A_2,	B_2	A_1,	B_1	A_0,	B_0	$A>B$	$A<B$	$A=B$	$A>B$	$A<B$
$A_3 > B_3$		×		×		×		×	×	×	1	0
$A_3 < B_3$		×		×		×		×	×	×	0	1
$A_3 = B_3$		$A_2 > B_2$		×		×		×	×	×	1	0
$A_3 = B_3$		$A_2 < B_2$		×		×		×	×	×	0	1
$A_3 = B_3$		$A_2 = B_2$		$A_1 > B_1$		×		×	×	×	1	0
$A_3 = B_3$		$A_2 = B_2$		$A_1 < B_1$		×		×	×	×	0	1
$A_3 = B_3$		$A_2 = B_2$		$A_1 = B_1$		$A_0 > B_0$		×	×	×	1	0
$A_3 = B_3$		$A_2 = B_2$		$A_1 = B_1$		$A_0 < B_0$		×	×	×	0	1
$A_3 = B_3$		$A_2 = B_2$		$A_1 = B_1$		$A_0 = B_0$		1	0	0	1	0
$A_3 = B_3$		$A_2 = B_2$		$A_1 = B_1$		$A_0 = B_0$		0	1	0	0	1
$A_3 = B_3$		$A_2 = B_2$		$A_1 = B_1$		$A_0 = B_0$		0	0	1	0	0
$A_3 = B_3$		$A_2 = B_2$		$A_1 = B_1$		$A_0 = B_0$		×	×	1	0	0
$A_3 = B_3$		$A_2 = B_2$		$A_1 = B_1$		$A_0 = B_0$		1	1	0	0	0
$A_3 = B_3$		$A_2 = B_2$		$A_1 = B_1$		$A_0 = B_0$		0	0	0	1	1

Truth Table

The IC 7485 (4 BIT COMPARATOR)

n of 8 bit comparator using two 7485



8 bit comparator using two 7485

8 BIT COMPARATORS

ehavioral model for mparator

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

entity vcompare is
    port (
        A, B: in STD_LOGIC_VECTOR (7 downto 0);
        EQ, NE, GT, GE, LT, LE: out STD_LOGIC
    );
end vcompare;

architecture vcompare_arch of vcompare is
begin
process (A, B)
  begin
    EQ <= '0'; NE <= '0'; GT <= '0'; GE <= '0'; LT <= '0'; LE <=
    if A = B then EQ <= '1'; end if;
    if A /= B then NE <= '1'; end if;
    if A > B then GT <= '1'; end if;
    if A >= B then GE <= '1'; end if;
    if A < B then LT <= '1'; end if;
    if A <= B then LE <= '1'; end if;
  end process;
end vcompare_arch
```

ADDERS & SUBTRACTORS

DERS

Half Adder

ull Adder

N-Bit Parallel Adder or Ripple Adder

Carry Lookahead Adders

BTRACTORS

Half Subtractor

ull Subtractor

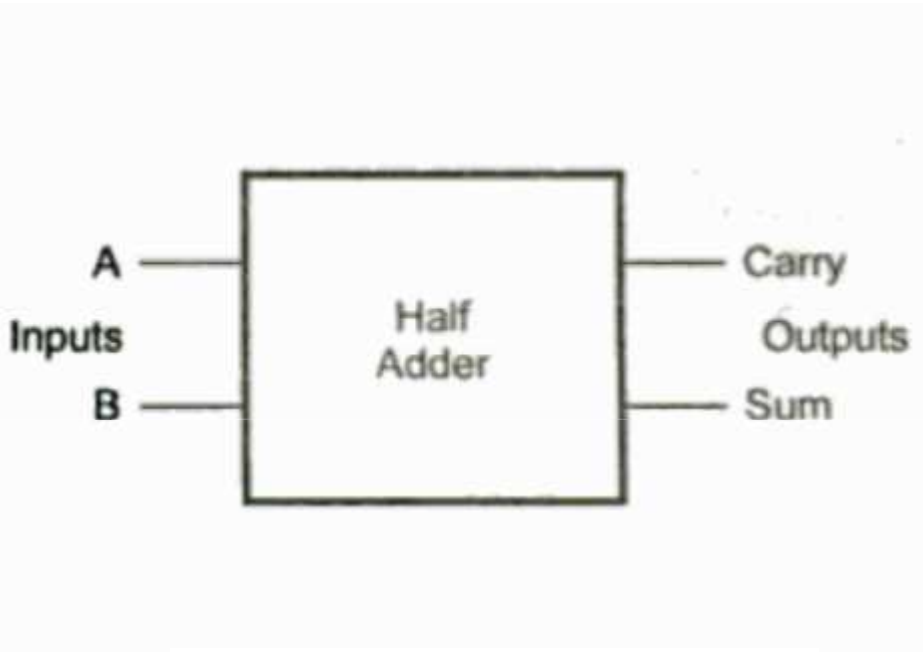
N-bit subtractor

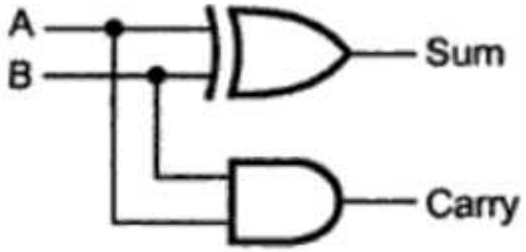
ADDERS

F ADDER

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

Truth Table





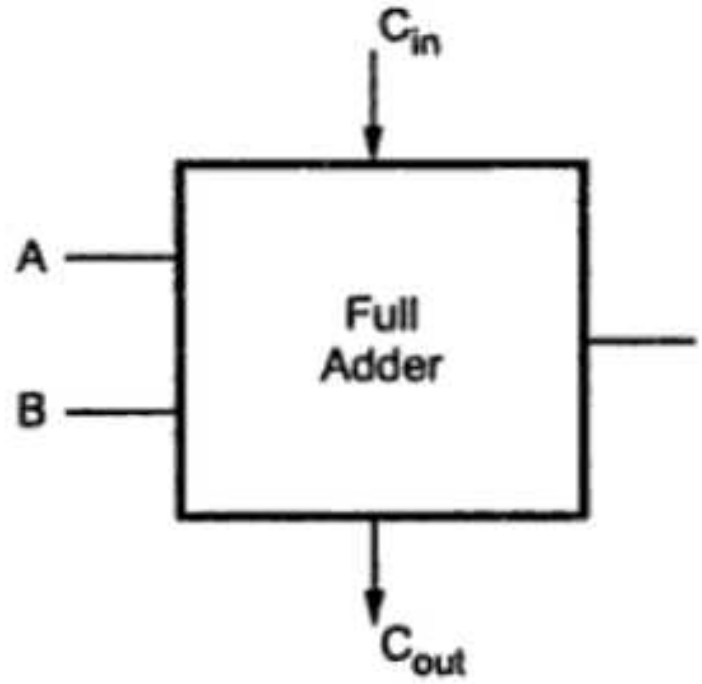
Logic Diagram

ADDERS

ADDER

Inputs			Outputs	
	B	C_{in}	Carry	Sum
	0	0	0	0
	0	1	0	1
	1	0	0	1
	1	1	1	0
	0	0	0	1
	0	1	1	0
	1	0	1	0
	1	1	1	1

th Table



$$\text{Sum} = \overline{A}\,\overline{B}\,C_{in} + \overline{A}\,B\,\overline{C}_{in} + A\,\overline{B}\,\overline{C}_{in} + A\,B\,C_{in}$$

$$= C_{in}\,(\overline{A}\,\overline{B} + AB) + \overline{C}_{in}\,(\overline{A}\,B + A\,\overline{B})$$

$$= C_{in}\,(A \odot B) + \overline{C}_{in}\,(A \oplus B) = C_{in}\,\overline{(A \oplus B)} + \overline{C}_{in}\,(A$$

$$= C_{in} \oplus (A \oplus B)$$

Logic Diagram

ADDERS

ADDER

For Carry (C_{out})

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

$C_{out} = AB + A\,C_{in} + B\,C_{in}$

For Sum

A \ BC_{in}	00	01	11	10
0	0	1	0	1
1	1	0	1	0

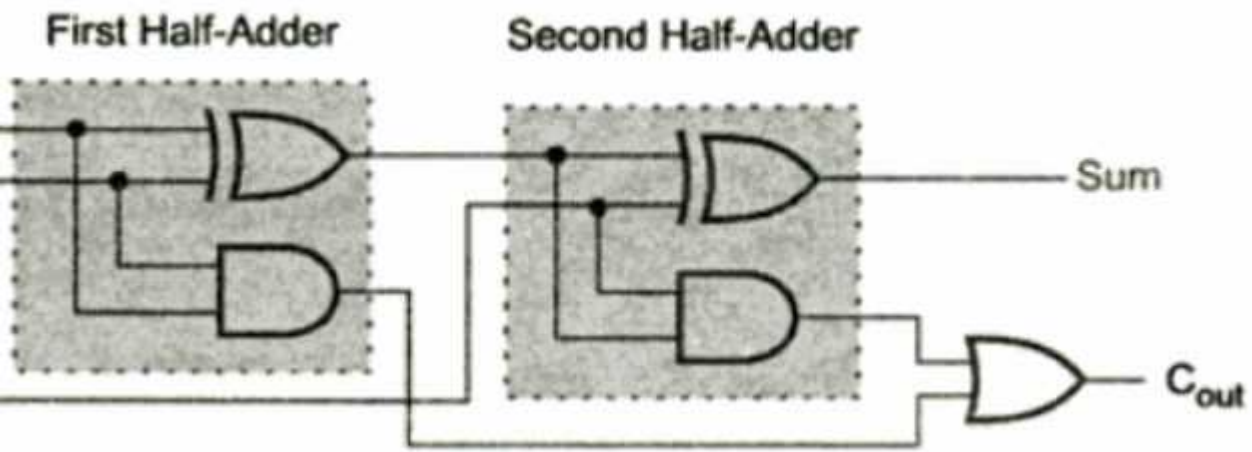
$Sum = \overline{A}\,\overline{B}C_{in} + \overline{A}B\overline{C}_{in} + A\overline{B}\,\overline{C}_{in} + ABC_{in}$

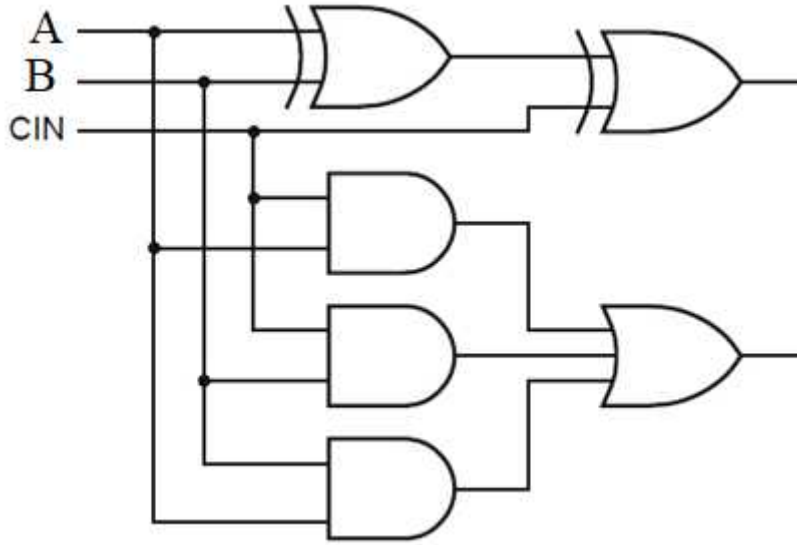
$$Sum = \overline{A}\,\overline{B}\,C_{in} + \overline{A}\,B\,\overline{C}_{in} + A\,\overline{B}\,\overline{C}_{in} + A\,B\,C_{in}$$

$$= C_{in}\,(\overline{A}\,\overline{B} + AB) + \overline{C}_{in}\,(\overline{A}\,B + A\,\overline{B})$$

$$= C_{in}\,(A \odot B) + \overline{C}_{in}\,(A \oplus B) = C_{in}\,\overline{(A \oplus B)} +$$

$$= C_{in} \oplus (A \oplus B)$$



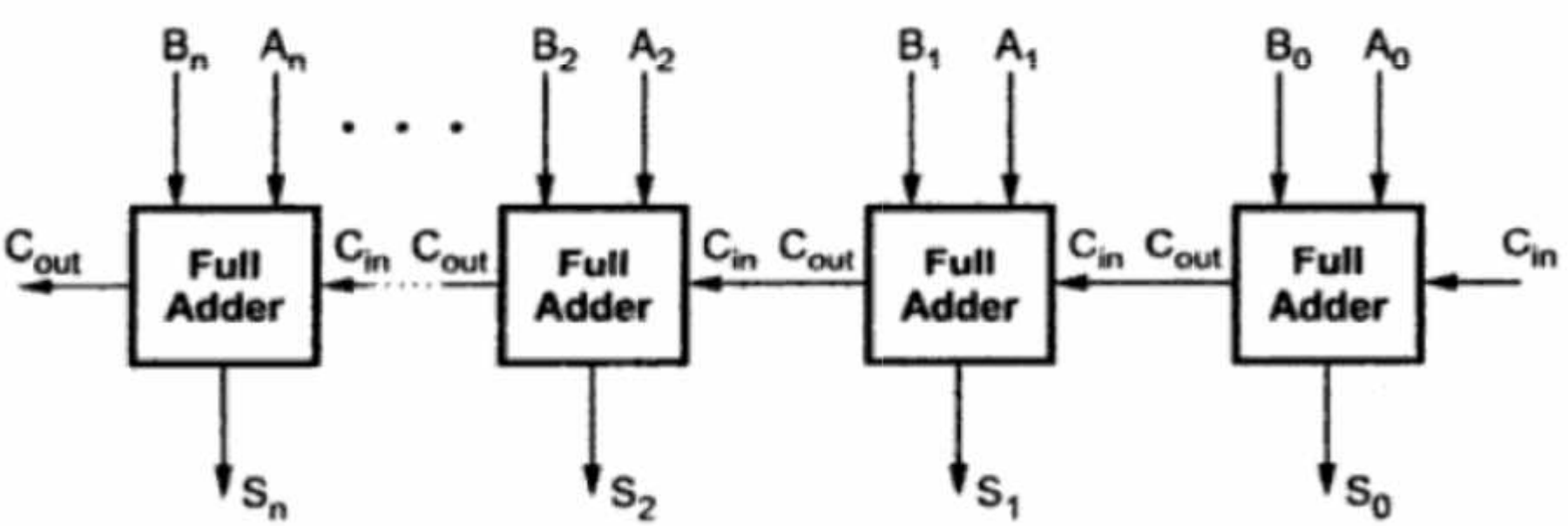


Logic diagra

ADDERS

Parallel ADDER

e ripple carry adder is constructed by cascading full adders (FA) blocks in series. Or
der is responsible for the addition of two binary digits at any stage of the ripple ca
e of the most serious drawbacks of this adder is that the delay increases linearly w
e bit length.

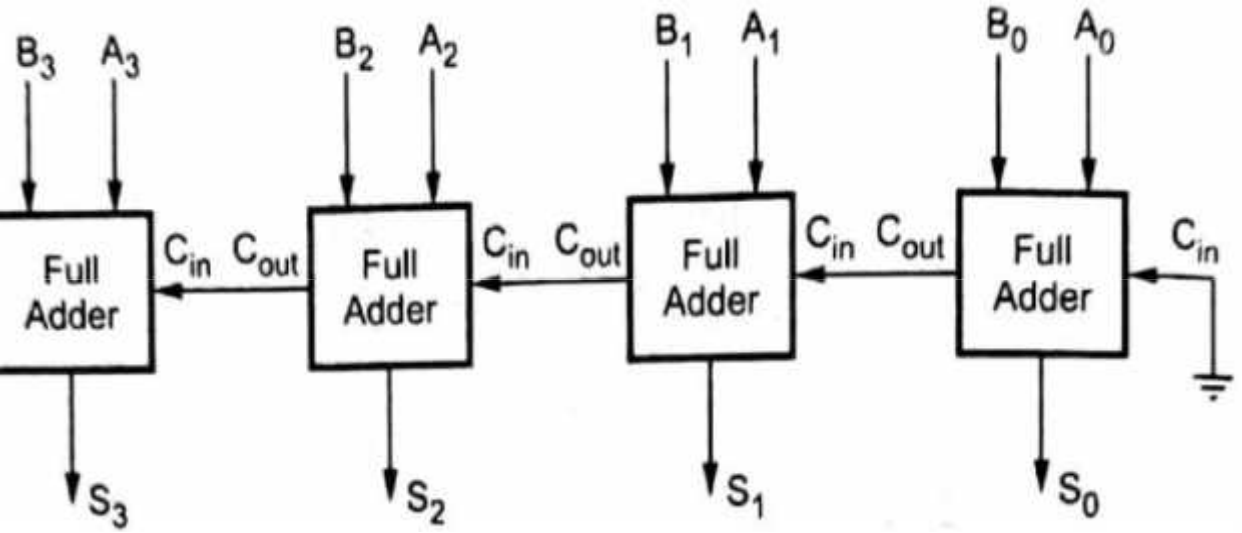


Block diagram of n-bit parallel adder

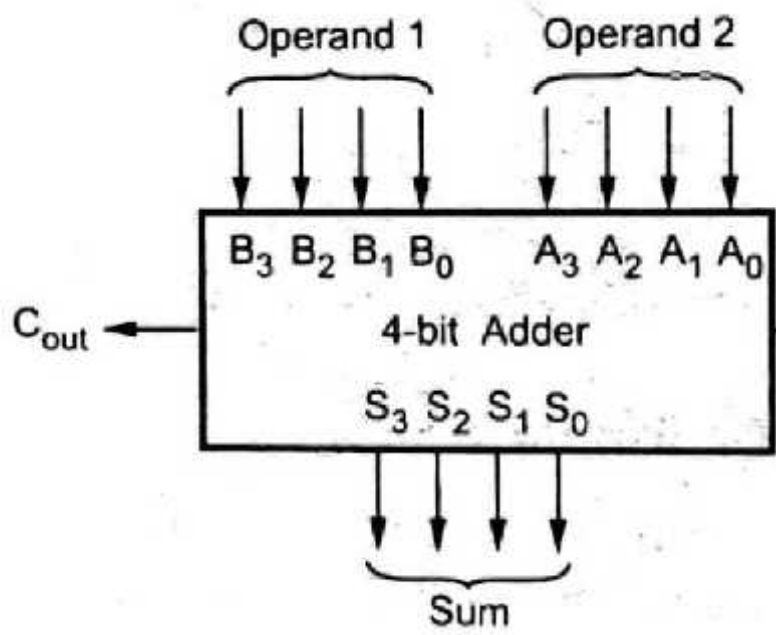
ADDERS

Parallel ADDER

e 4 bit ripple carry adder is constructed by cascading four full adders (FA) blocks in ies.



liagram of 4-bit parallel adder



4-bit parallel adder

Structural Program for 4-Bit Parallel ADDER


```
ARY IEEE ;
IEEE.std_logic_1164.all;
TY adder4 IS
PORT      (CIN                  : IN STD_LOGIC;
           A3, A2, A1, A0       : IN STD_LOGIC;
           B3, B2, B1, B0       : IN STD_LOGIC;
           S3, S2, S1, S0       : OUT STD_LOGIC;
           Cout                 : OUT STD_LOGIC);
adder4;
ITECTURE Structure OF adder4 IS
SIGNAL c1, c2, c3 : STD_LOGIC;
COMPONENT fulladd
     PORT (   CIN, A, B         : IN STD_LOGIC;
              S, cOUT           : OUT STD_LOGIC);
END COMPONENT;
N
e0 : fulladd PORT MAP (CIN, A0, B0, S0, C1);
e1 : fulladd PORT MAP (C1, A1, B1, S1, C2);
e2 : fulladd PORT MAP (C2, A2, B2, S2, C3);
e3 : fulladd PORT MAP
     CIN=>C3, A=>A3, B=>B3, S=>S3, COUT=> COUT);
Structure;
```

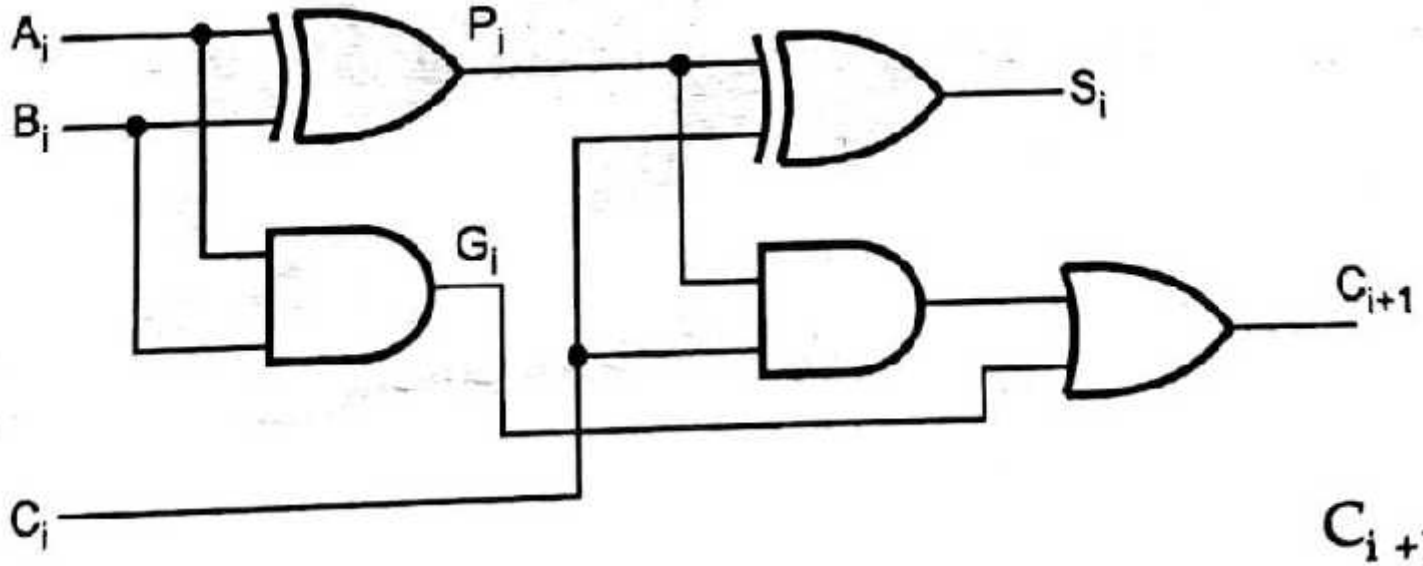
ADDERS

y Lookahead Adder

e ripple-carry adder, its limiting factor is the time it takes to propagate the carry.
e carry look-ahead adder solves this problem by calculating the carry signals in adv
sed on the input signals. The result is a reduced carry propagation time.
arry-Lookahead adder is a fast parallel adder as it reduces the propagation delay.
nsider the circuit of the full adder, Here we define two functions **Carry generate (G**
rry propagate (Pi)

$= A_i \oplus B_i$

$= A_i B_i$



$S_i = P$

$C_{i+1} = G_i +$

ADDERS

is called a carry generate and it produces on carry when both inputs Ai and Bi are
gardless of the input carry.

s called a carry propagate because it is term associated with the propagation of th
ry from Ci to Ci+1.

e Boolean expression for the carry output of the each stage can be written as

$$C_{i+1} = G_i + P$$

$$C_2 = G_1 + P_1C_1$$

$$C_3 = G_2 + P_2C_2 = G_2 + P_2\,(G_1 + P_1C_1)$$

$$= G_2 + P_2G_1 + P_2\,P_1C_1$$

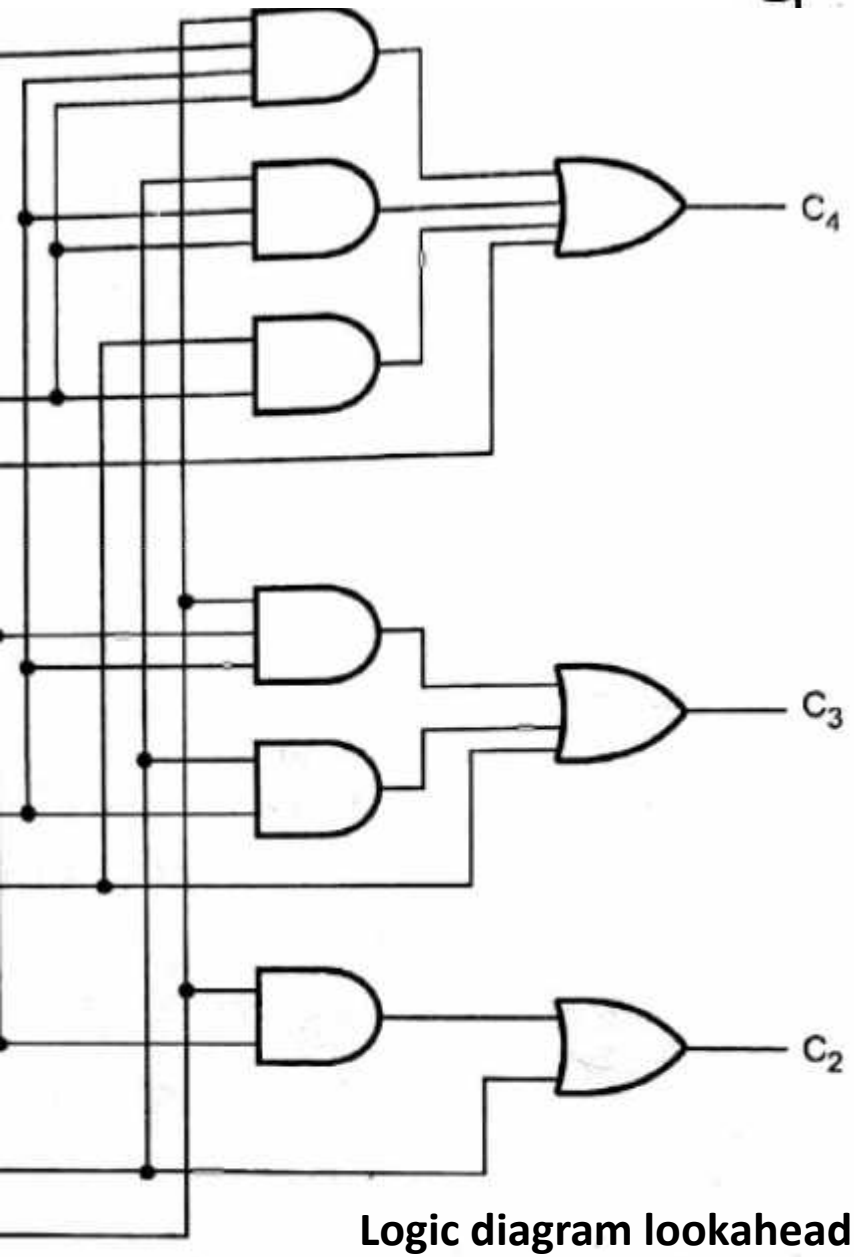
$$C_4 = G_3 + P_3\,C_3 = G_3 + P_3\,(G_2 + P_2\,G_1 + P_2\,P_1C_1)$$

$$= G_3 + P_3G_2 + P_3\,P_2\,G_1 + P_3\,P_2\,P_1\,C_1$$

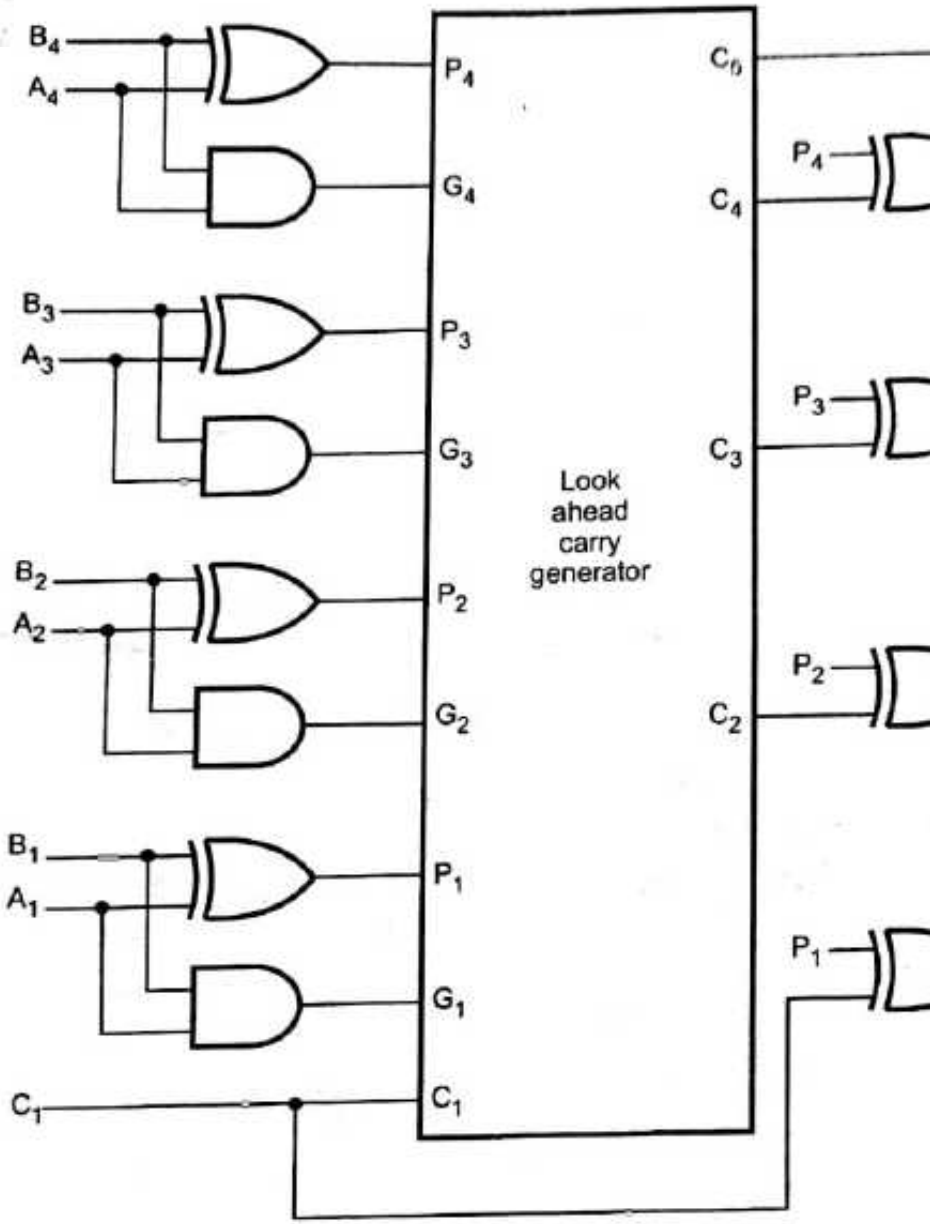
$$P_i = A_i \oplus B_i$$

$$G_i = A_i B_i$$

$$S_i =$$

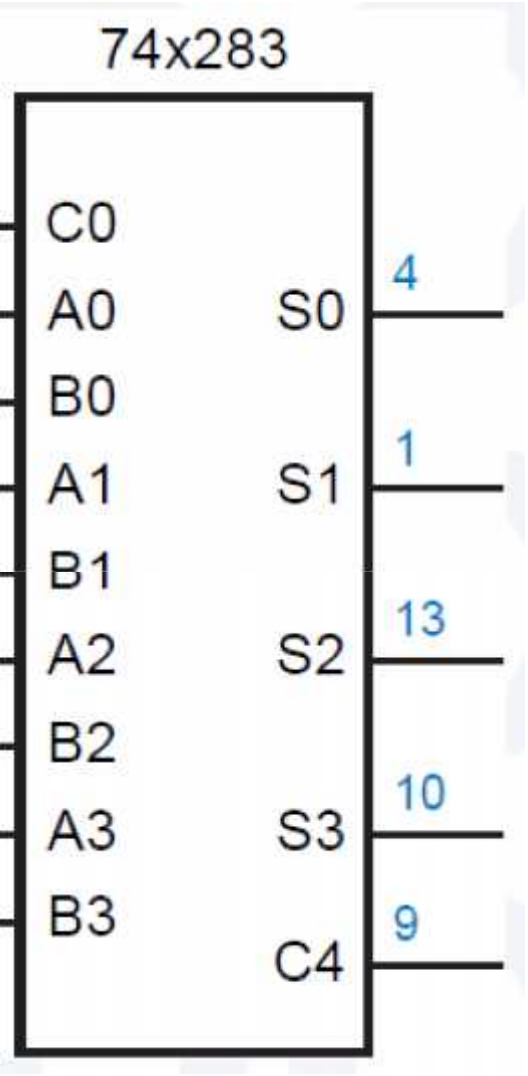


Logic diagram lookahead carry generator



4 bit parallel adder with lookahead carry g

IC 74LS283 Binary Parallel Adder



Symbol for the 4 bit adder

- The IC 74x283 contains four inter connecte adders and carry look ahead circuit for high operation.
- The inputs to the IC are A0A1A2A3 and B0B1B2B3, the outputs are S0S1S2S3 and C output carry

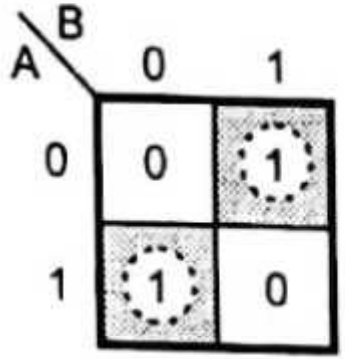
SUBTRACTORS

F SUBTRACTOR

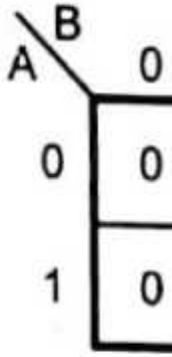
Inputs		Outputs	
A	B	Difference	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

uth Table

For Difference



For B

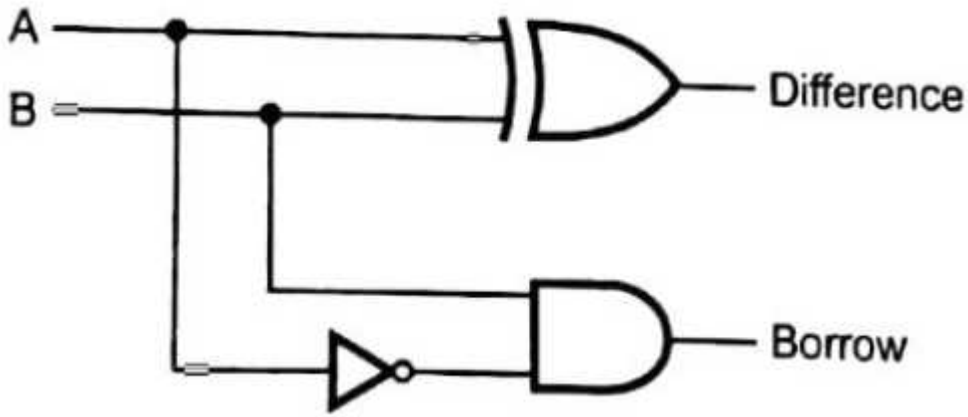


Bor

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

= $A \oplus B$

K-Map Simplification



Logic Diagram

SUBTRACTORS

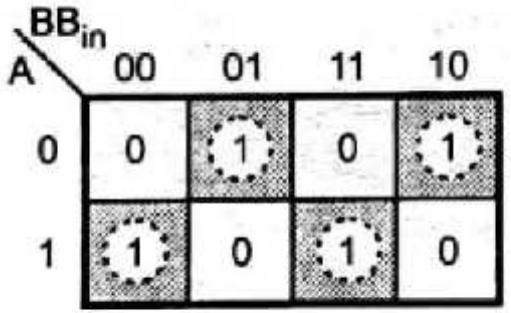
SUBTRACTOR

Inputs			Outputs	
A	B	B_{in}	D	B_{out}
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

th Table

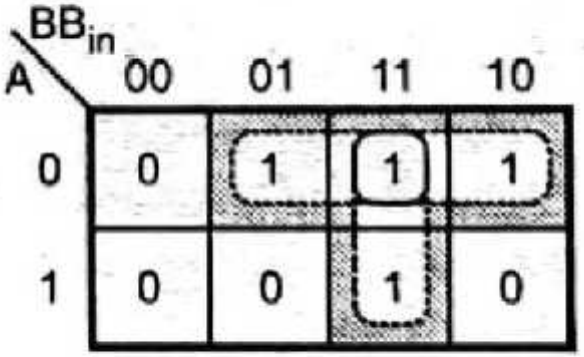
Logic Diagram

For D



$$D = \bar{A}\bar{B}B_{in}+\bar{A}B\bar{B}_{in}+A\bar{B}\bar{B}_{in}+ABB_{in}$$

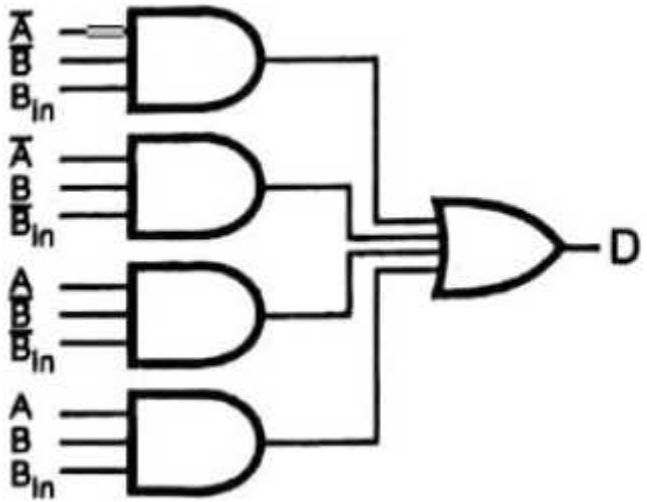
For B_{out}

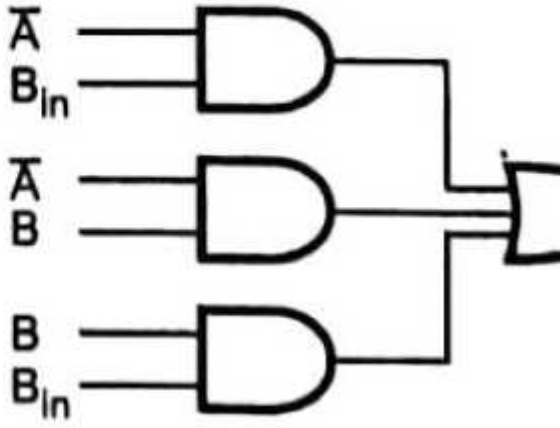


$$B_{out} = \bar{A}B_{in}+\bar{A}B+BB_{in}$$

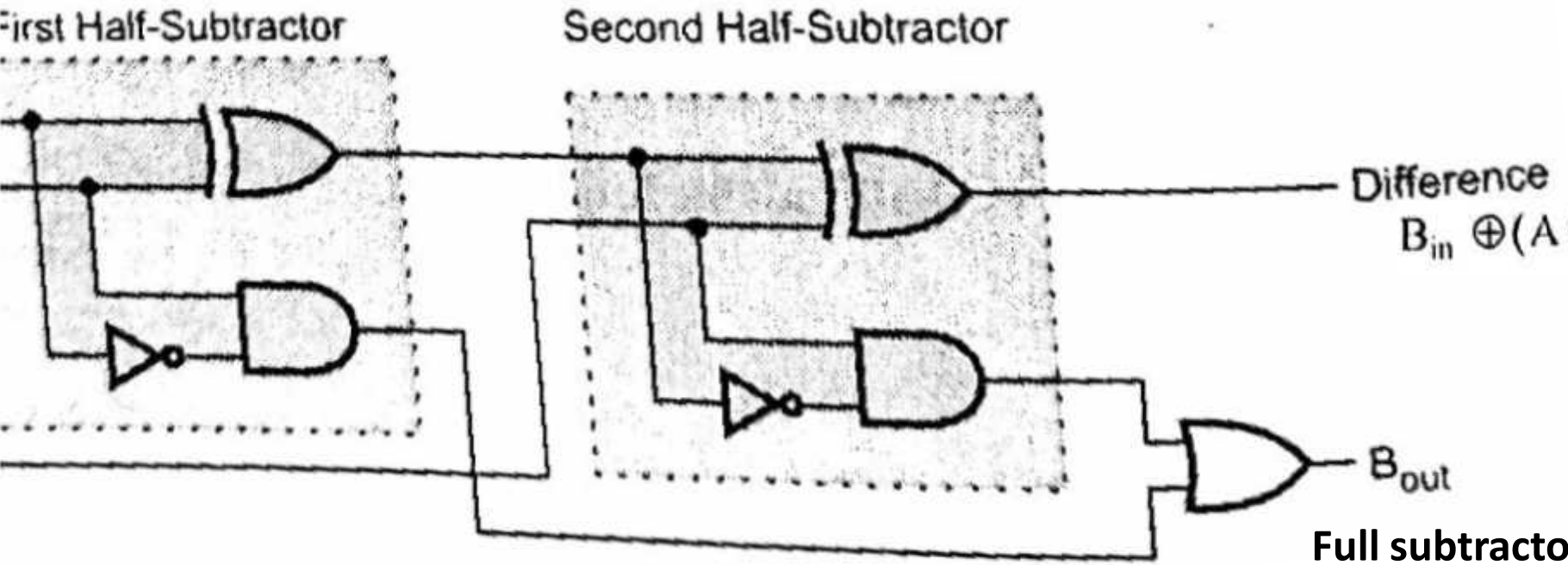
SUBTRACTORS

SUBTRACTOR



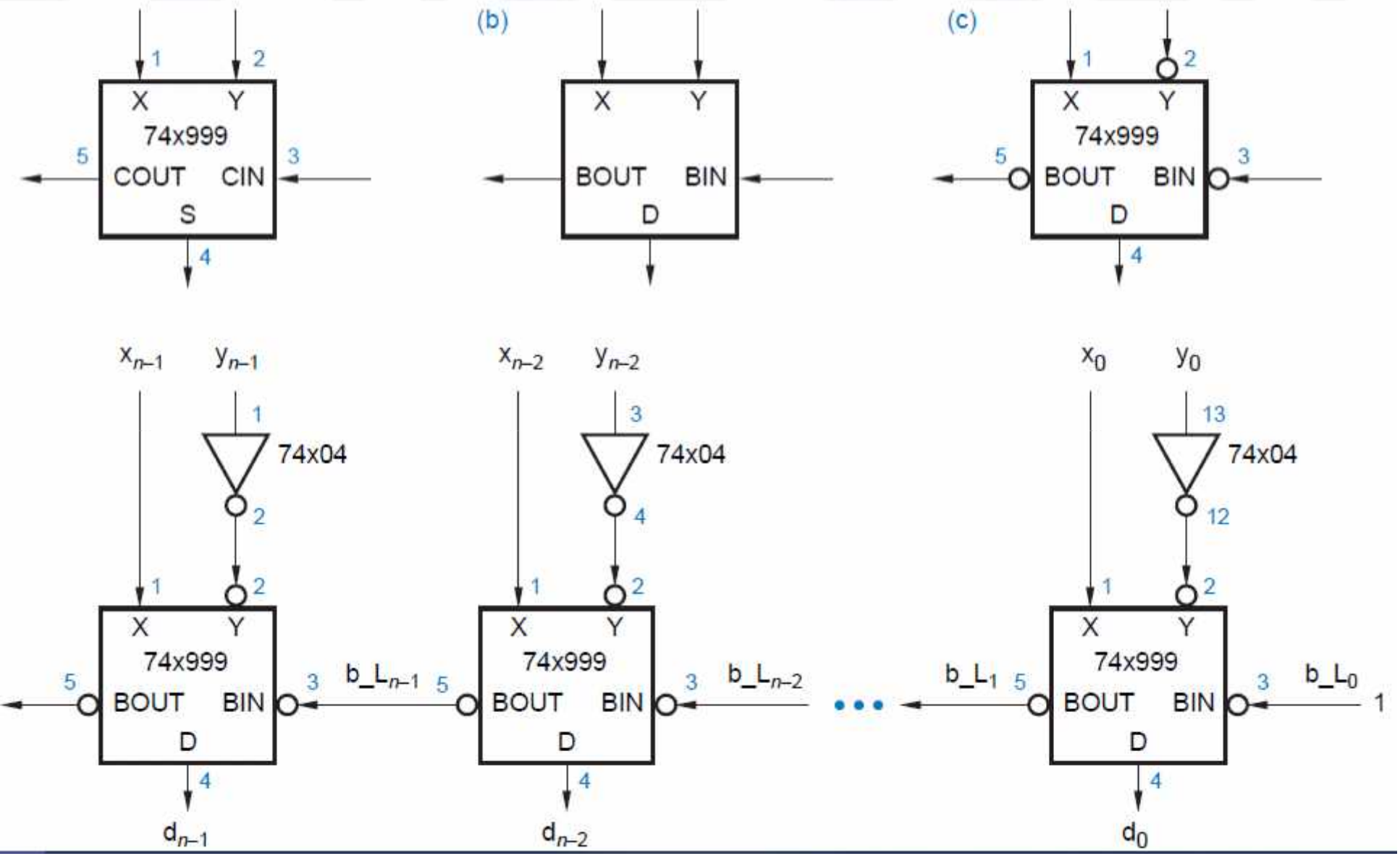


Logic di



Full subtractor using half su

SUBTRACTORS

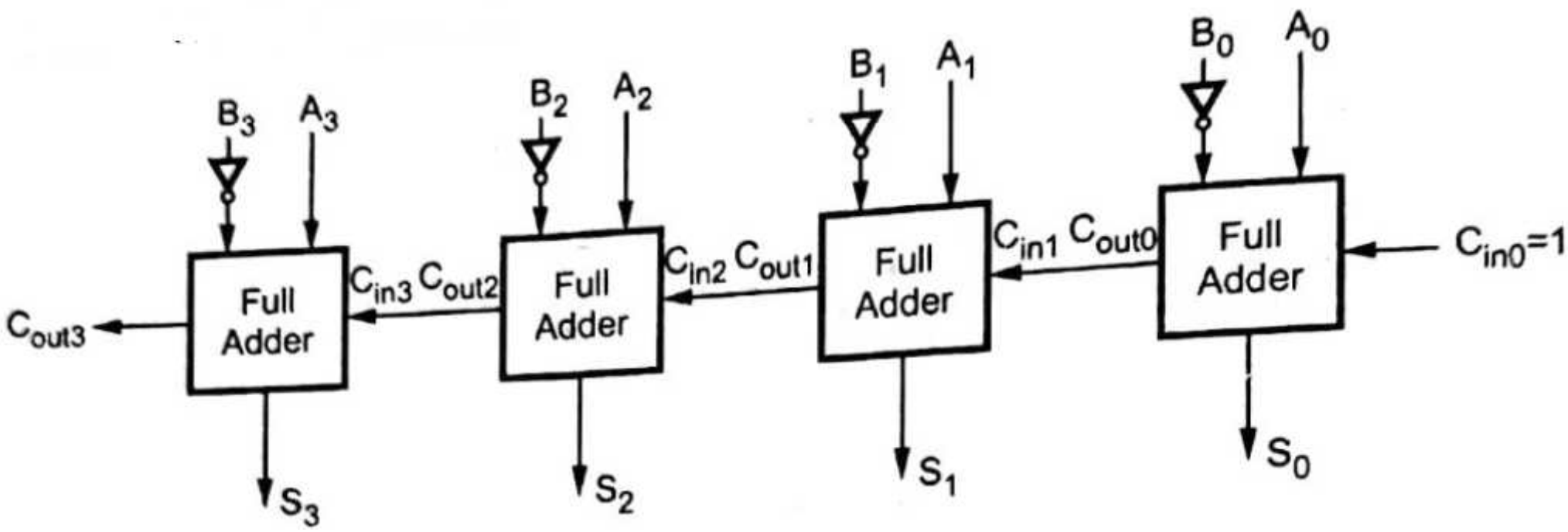


Designing subtractors using adders: (a) full adder; (b) full subtractor; (c) interpreting the device in (a) as a full subtractor; (d) ripple subtractor.

SUBTRACTORS

Parallel Subtractor

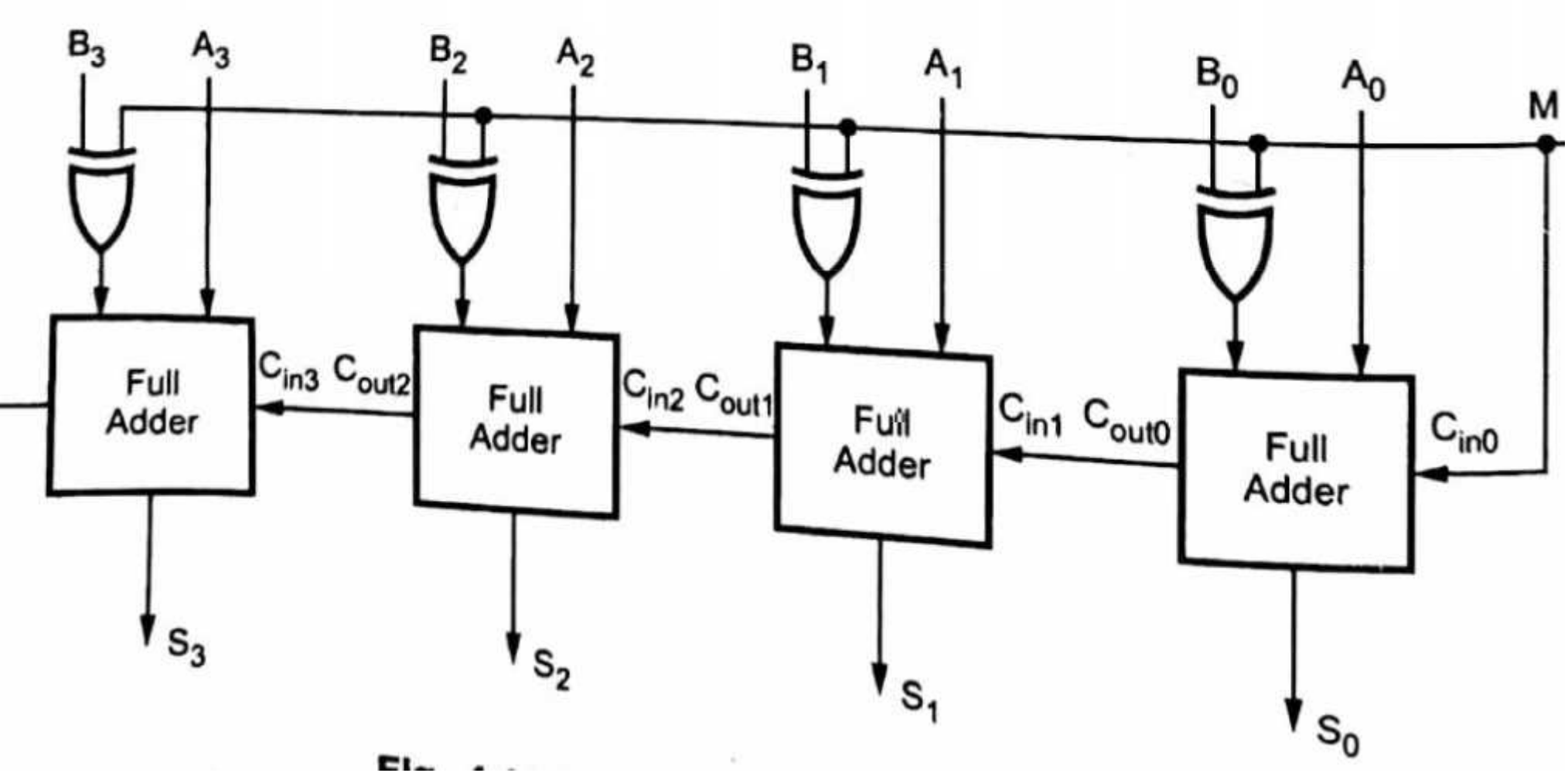
e ripple carry adder is constructed by cascading full adders (FA) blocks in series. Or
der is responsible for the addition of two binary digits at any stage of the ripple ca
e of the most serious drawbacks of this adder is that the delay increases linearly w
e bit length.



Block diagram of n-bit parallel subtractor

t Adder and Subtractor

he Mode input M controls the operation of the circuit
Vhen M=0, The circuit is an **Adder**
Vhen M=1, The circuit is an **Subtractor**



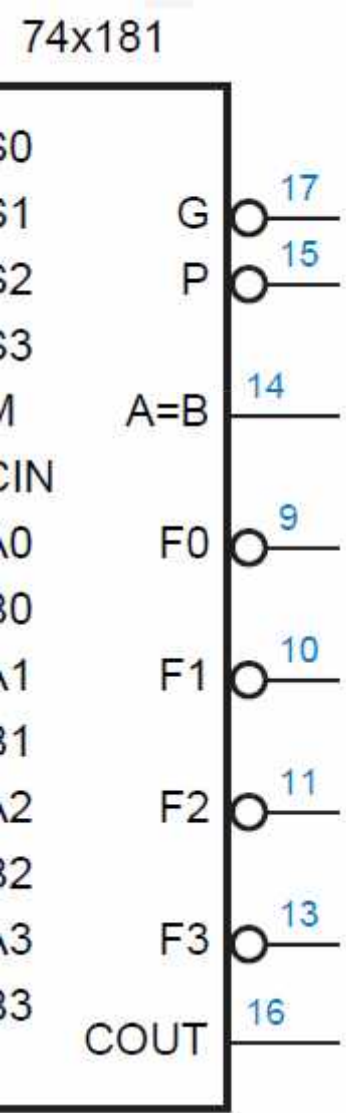
4 bit adder and subtractor

Arithmetic Logic Unit (ALU)

erforms the arithmetic and logical operations.

asically a multifunction combinational logic circuit.

C 74x181 is a 4 bit ALU



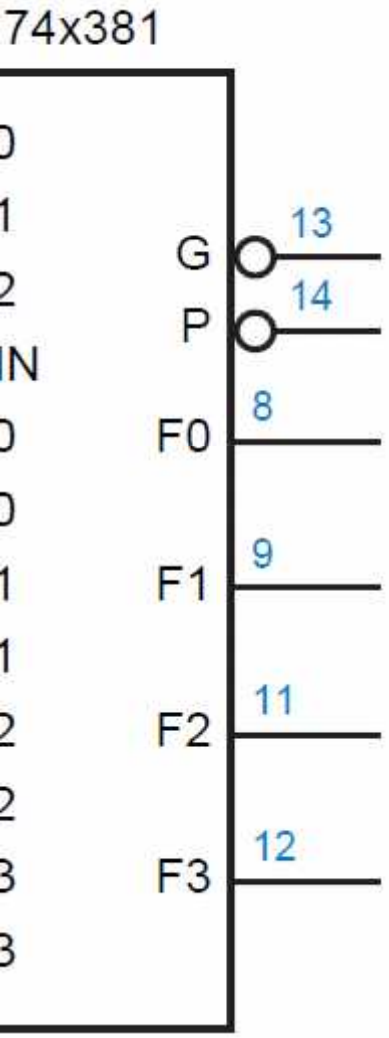
Symbol

Pin names	Description
A_0 - A_3	Operand Inputs
B_0 - B_3	Operand Inputs
S_0 - S_3	Function Select Inputs
M	Mode Control Input
C_{IN}	Carry Input
F_0 - F_3	Function Outputs
A = B	Comparator Output
G	Carry Generate Output
P	Carry Propagate Output
C_{OUT}	Carry Output

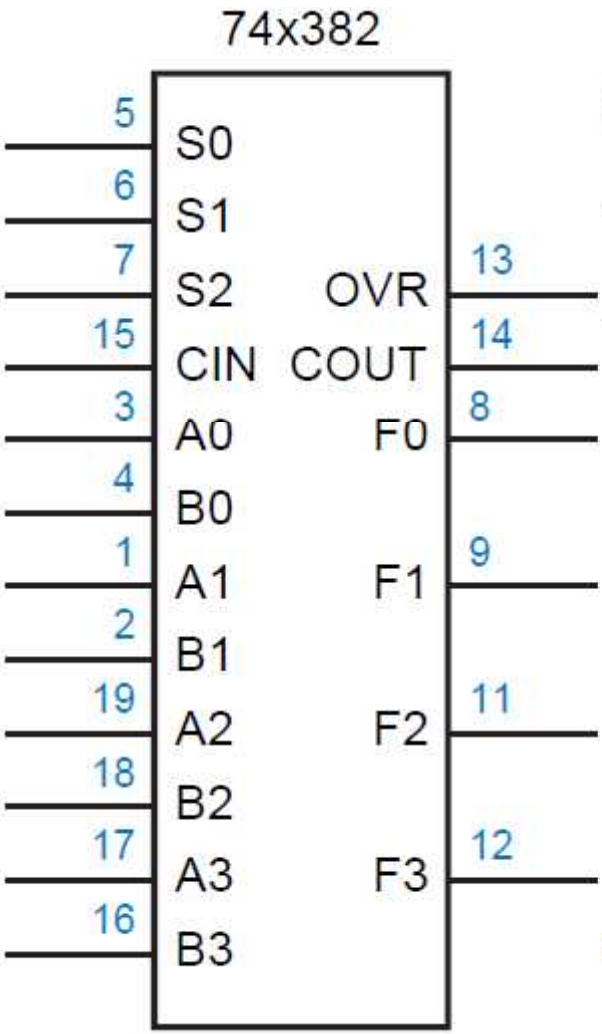
Inputs				Function	
S3	S2	S1	S0	*M = 0 (arithmetic)*	*M = 1 (logic)*
0	0	0	0	F = A minus 1 plus CIN	F = A′
0	0	0	1	F = A · B minus 1 plus CIN	F = A′ + B′
0	0	1	0	F = A · B′ minus 1 plus CIN	F = A′ + B
0	0	1	1	F = 1111 plus CIN	F = 1111
0	1	0	0	F = A plus (A + B′) plus CIN	F = A′ · B′
0	1	0	1	F = A ·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

more 4 bit ALUs are **74x381 and 74x382**

difference between the two ALUs are one provide group carry lookahead carry out
the other provides ripple carry and overflow outputs.



mbol



7x382 Symbol

Inputs			Function
S2	S1	S0	
0	0	0	F = 0000
0	0	1	F = B minus A minus 1
0	1	0	F = A minus B minus 1
0	1	1	F = A plus B plus CIN
1	0	0	F = A ⊕ B
1	0	1	F = A + B
1	1	0	F = A · B
1	1	1	F = 1111

Functional Table

ogram for 74x381 ALU

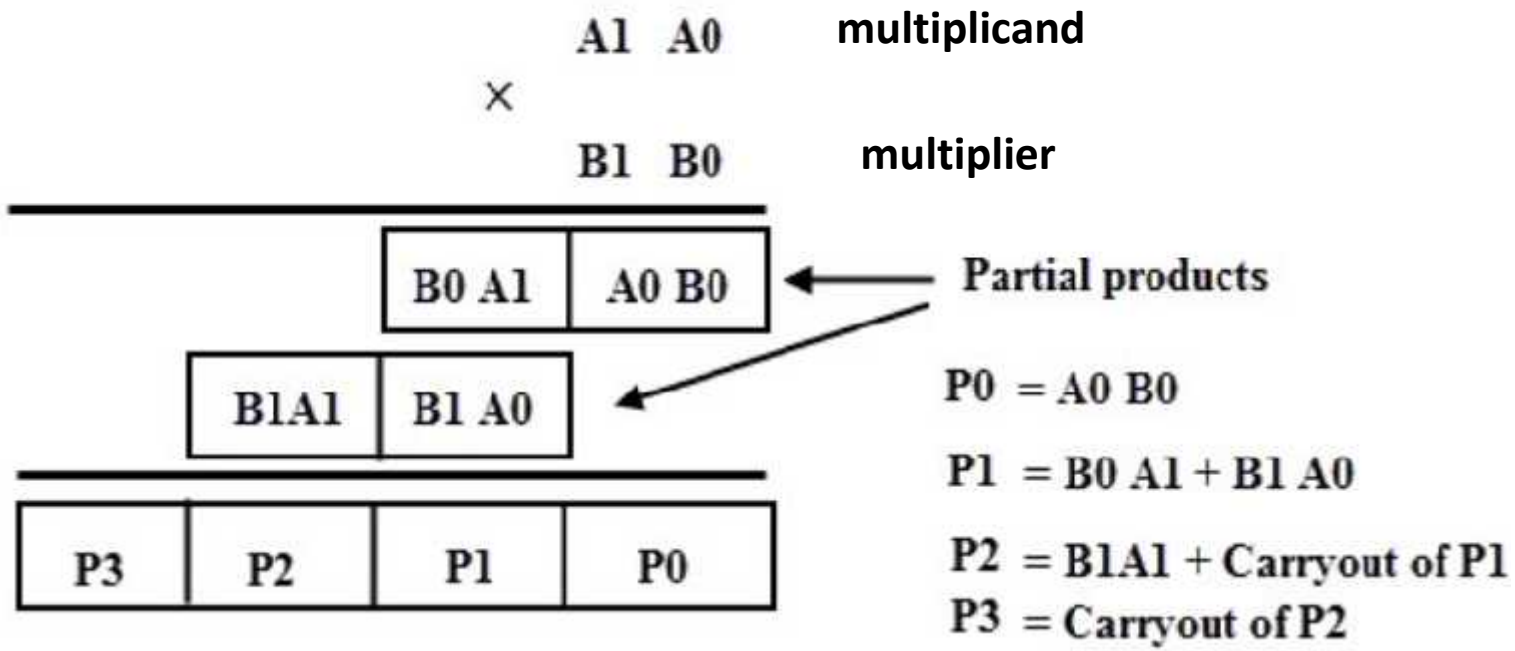
```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_unsigned.all;
ENTITY ALU IS
     PORT ( S : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
       A,B    : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
       F      : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
       (IN    : OUT STD_LOGIC);
END ALU;
ARCHITECTURE Behavior OF ALU IS
BEGIN
  PROCESS (S, A, B)
    BEGIN
        CASE S IS
          WHEN "000"=> F<="0000";
          WHEN "001"=> F<=B-A-1+CIN;
          WHEN "010"=> F<=A-B-1+CIN;
          WHEN "011"=> F<=A+B+CIN;
          WHEN "100"=> F<=A XOR B;
          WHEN "101"=> F<=A OR B;
          WHEN "110"=> F<=A AND B;
          WHEN OTHERS => F<=1111;
          END CASE;
       END PROCESS;
    END Behavior;
```

COMBINATIONAL MULTIPLIERS

nary multiplier is a combinational logic circuit used in digital systems to perfor
iplication of two binary numbers.

us consider two unsigned 2 bit binary numbers A and B to generalize the multipli
ess.

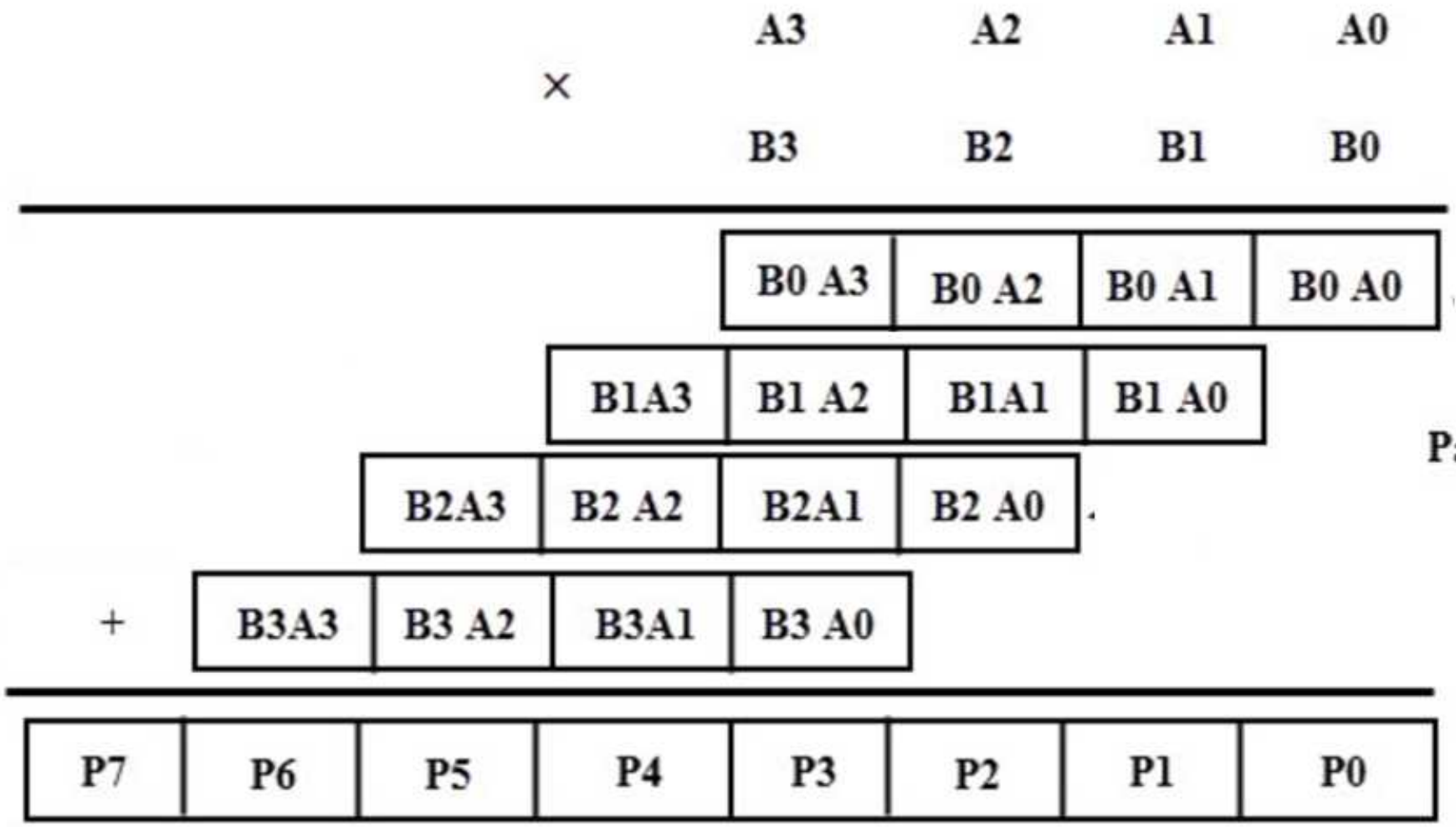
multiplicand **A is equal to A1A0** and the multiplier **B is equal to B1B0**.



2x2 multiplier

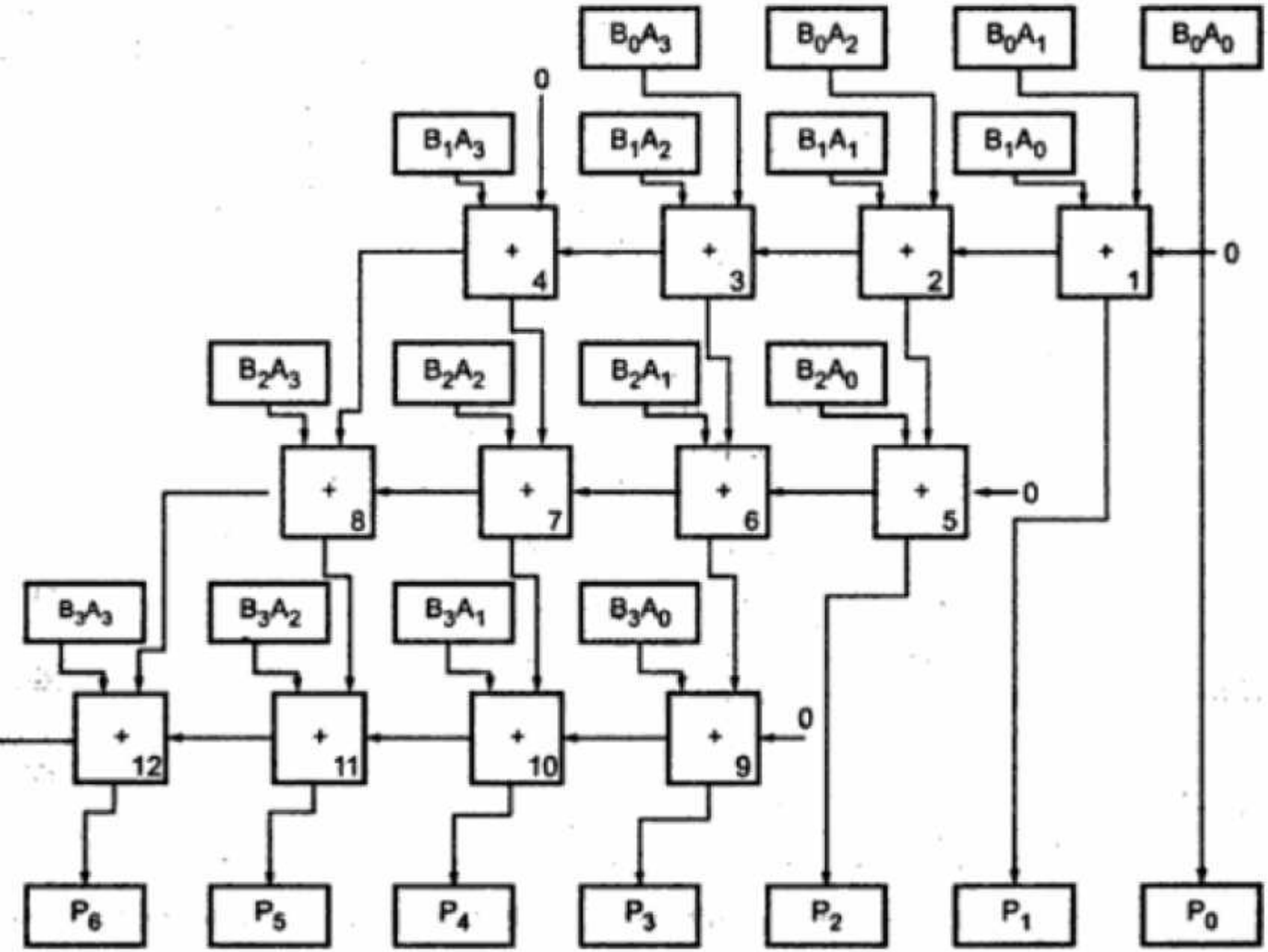
COMBINATIONAL MULT

Multiplier



Multiplier

COMBINATIONAL MULT



P0=B0A0
P1=B0A1+B1A0
P2=B0A2+B1A1+B
P3=B0A3+B1A2+B
P4=B1A3+B2A2+B
P5=B2A3+B3A2
P6=B3A3
P7=Carry out

Combinational m

PROGRAMMABLE LOGIC DEVICES (PLDs)

Programmable Logic Device (PLD) is an electronic component used to bui
onfigurable digital circuits

ey contain an array of AND gates & another array of OR gates.

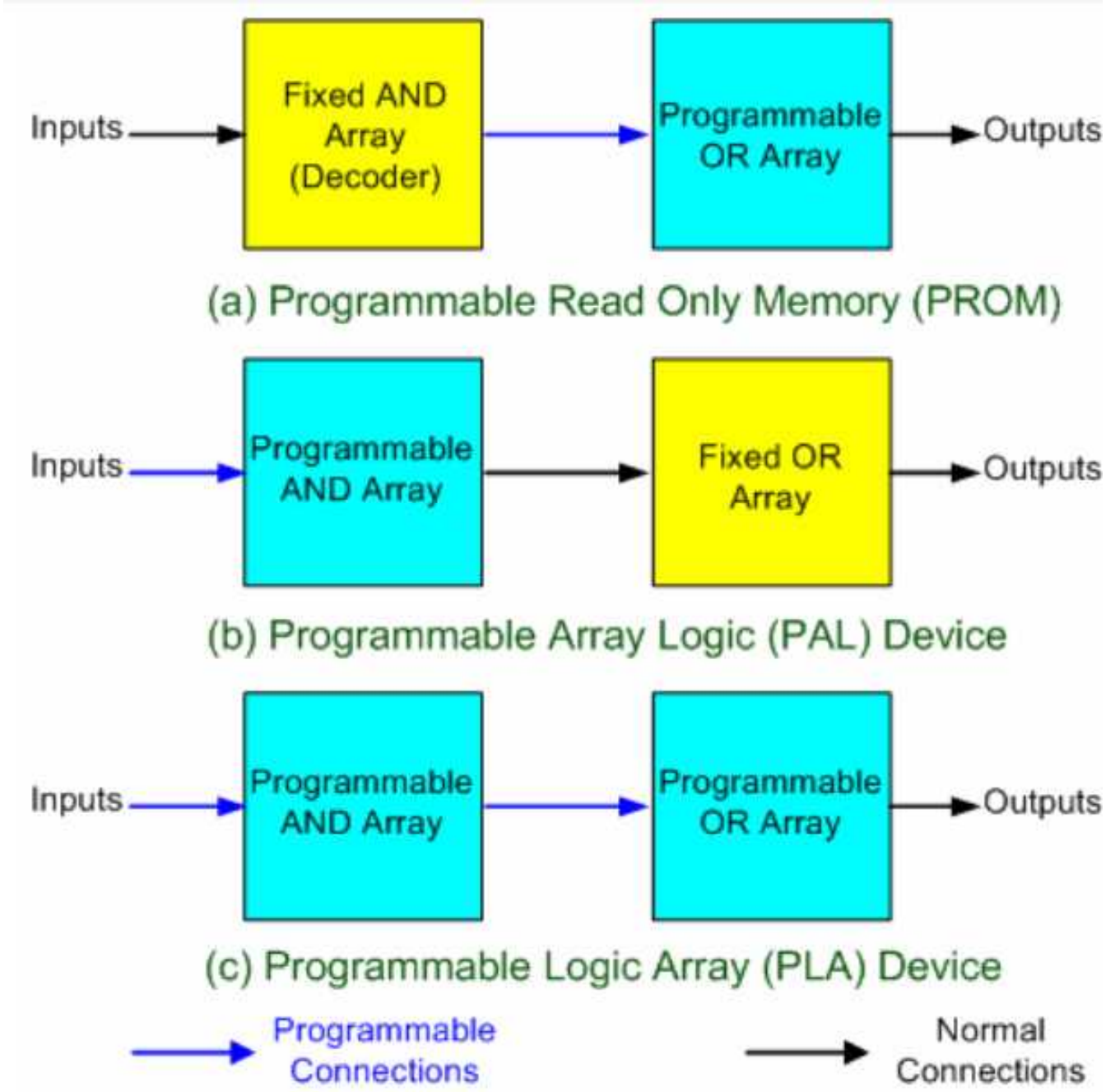
e process of entering the information into these devices is known as **programmin**

ically, users can program these devices or ICs electrically in order to implement th
olean functions based on the requirement.

ere are three kinds of PLDs based on the type of array(s), which has programmab
ture.

1) **Programmable Read Only Memory (PROM)**
2) **Programmable Logic Array (PLA)**
3) **Programmable Array Logic (PAL)**

PROGRAMMABLE LOGIC DEVICES (PLDs)

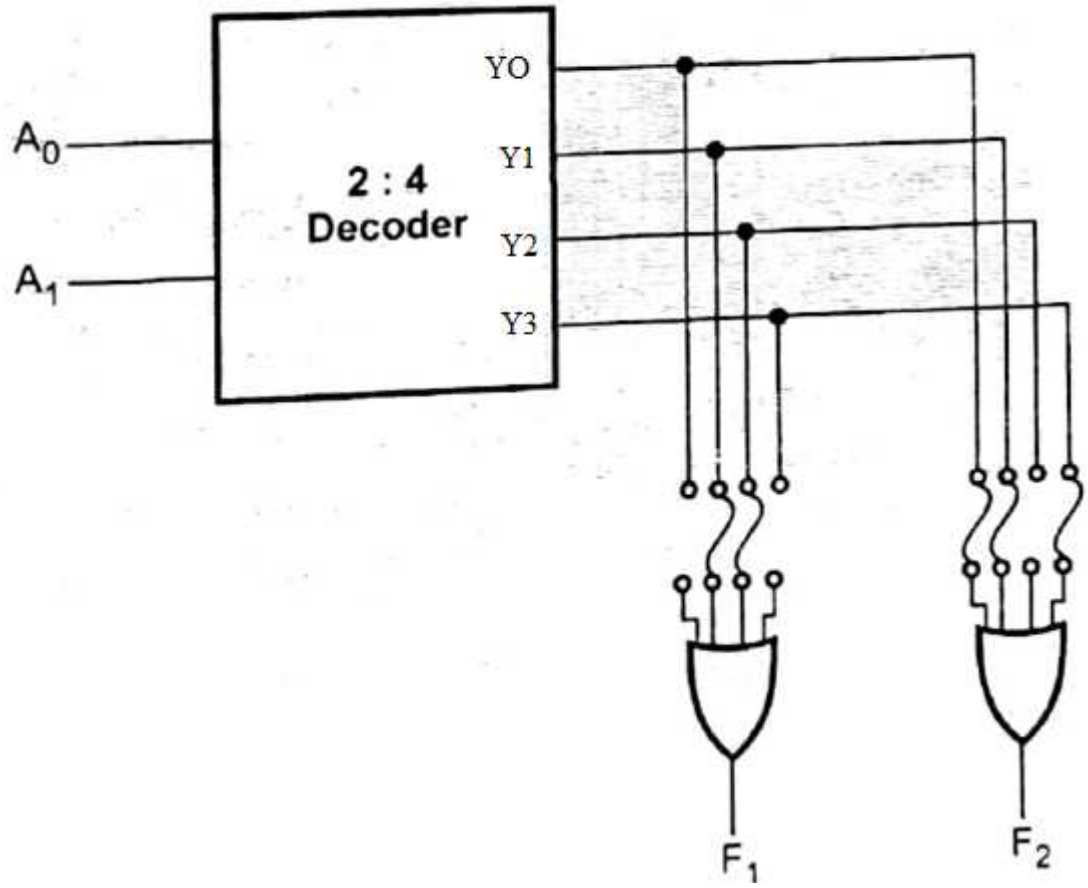


ammable Read Only Memory (PROM)

grammable Read Only Memory is the programmable logic devices in which **AND a**
d and **OR array is programmable**.

lement the given functions using PROM

$$F_1(A_1, A_0) = \sum m\,(1, 2)$$
$$F_2(A_1, A_0) = \sum m\,(0, 1, 3)$$



Programmable Logic Array (PLA)

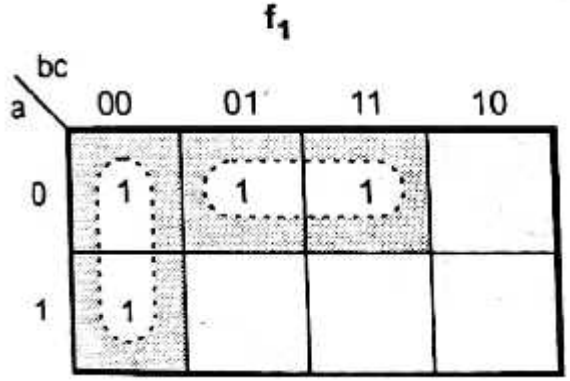
grammable Logic Array(PLA) is a fixed architecture logic device with programm
D gates followed by programmable OR gates

lement the given functions
ng 3x4x2 PLA architecture

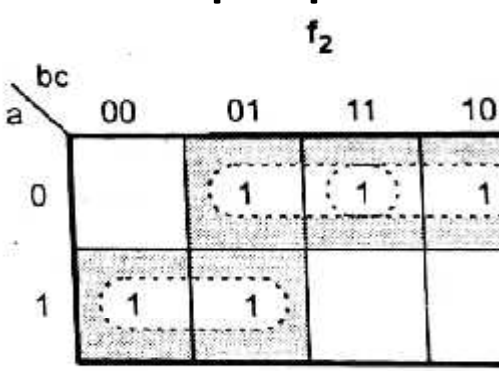
$$,c) = \sum m(0,1,3,4)$$

$$,c) = \sum m(1,2,3,4,5)$$

K-map implementa



$f_1 = \bar{b}\,\bar{c} + \bar{a}\,c$



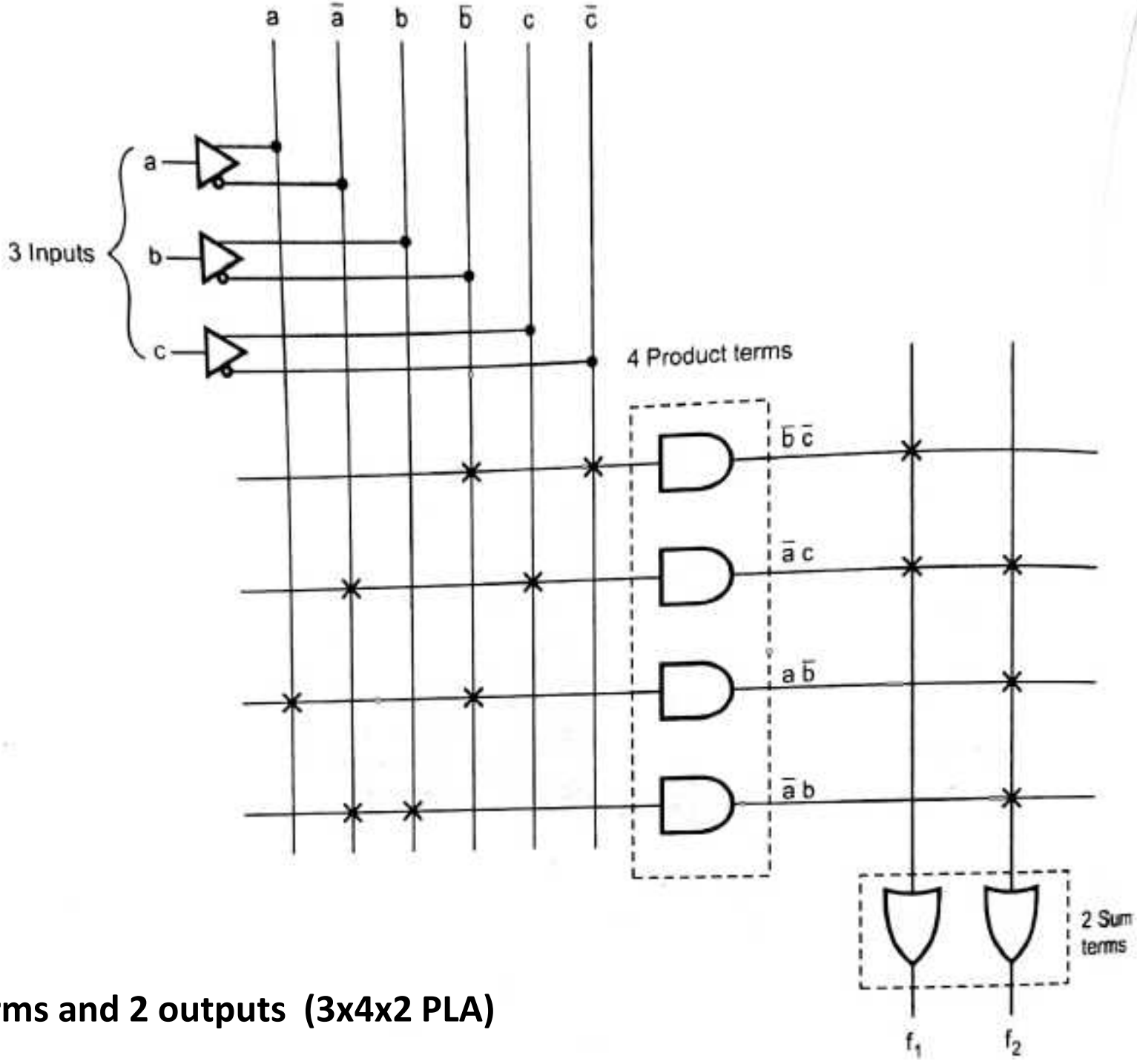
$f_2 = a\,\bar{b} + \bar{a}\,c + \bar{a}\,b$

Product terms	Inputs			Outputs	
	a	b	c	f_1	
$\bar{b}\,\bar{c}$	–	0	0	1	
$\bar{a}\,c$	0	–	1	1	
$a\,\bar{b}$	1	0	–	–	
$\bar{a}\,b$	0	1	–	–	
				T	

PLA Programming table

$\bar{b}\,\bar{c} + \bar{a}\,c$

$a\,\bar{b} + \bar{a}\,c + \bar{a}\,b$



with 3 inputs, 4 product terms and 2 outputs (3x4x2 PLA)

Programmable Logic Array (PLA)

ement the given functions using 3x4x2 PLA architecture

$$f_1(a_2, a_1, a_0) = \sum m(0,1,3,5)$$

$$f_2(a_2, a_1, a_0) = \sum m(3,5,7)$$

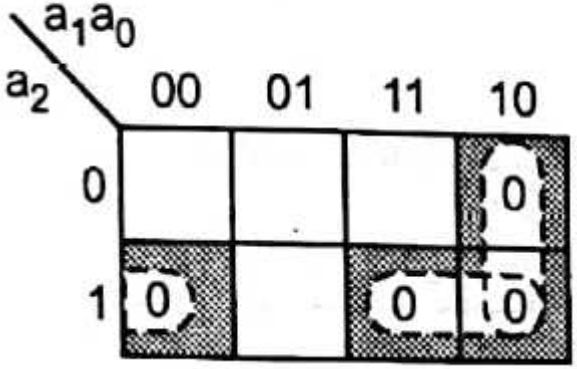
map implementation we got this Boolean equations

$$\bar{a}_2\,\bar{a}_1 + \bar{a}_2\,a_0 + \bar{a}_1\,a_0$$

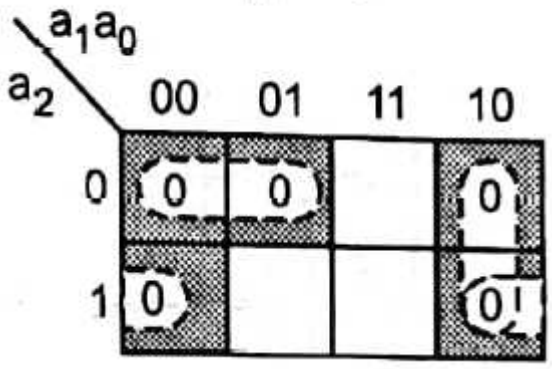
$$a_2\,a_0 + a_1\,a_0$$

To implement to this functions it require **3x5x2 PLA** but to implement using **3x4x2 PLA**, so consider the POS form of SOP form

$\bar{f}_1$-map



$\bar{f}_2$-map



$$\bar{f}_1 = a_2\,\bar{a}_0 + a_1\,\bar{a}_0 + a_2\,a_1$$

$$\bar{f}_2 = \bar{a}_2\,\bar{a}_1 + a_1\,\bar{a}_0$$

PLA Programming table

Programmable Logic Array (PLA)

$= a_2 \bar{a}_0 + a_1 \bar{a}_0 + a_2 a_1$

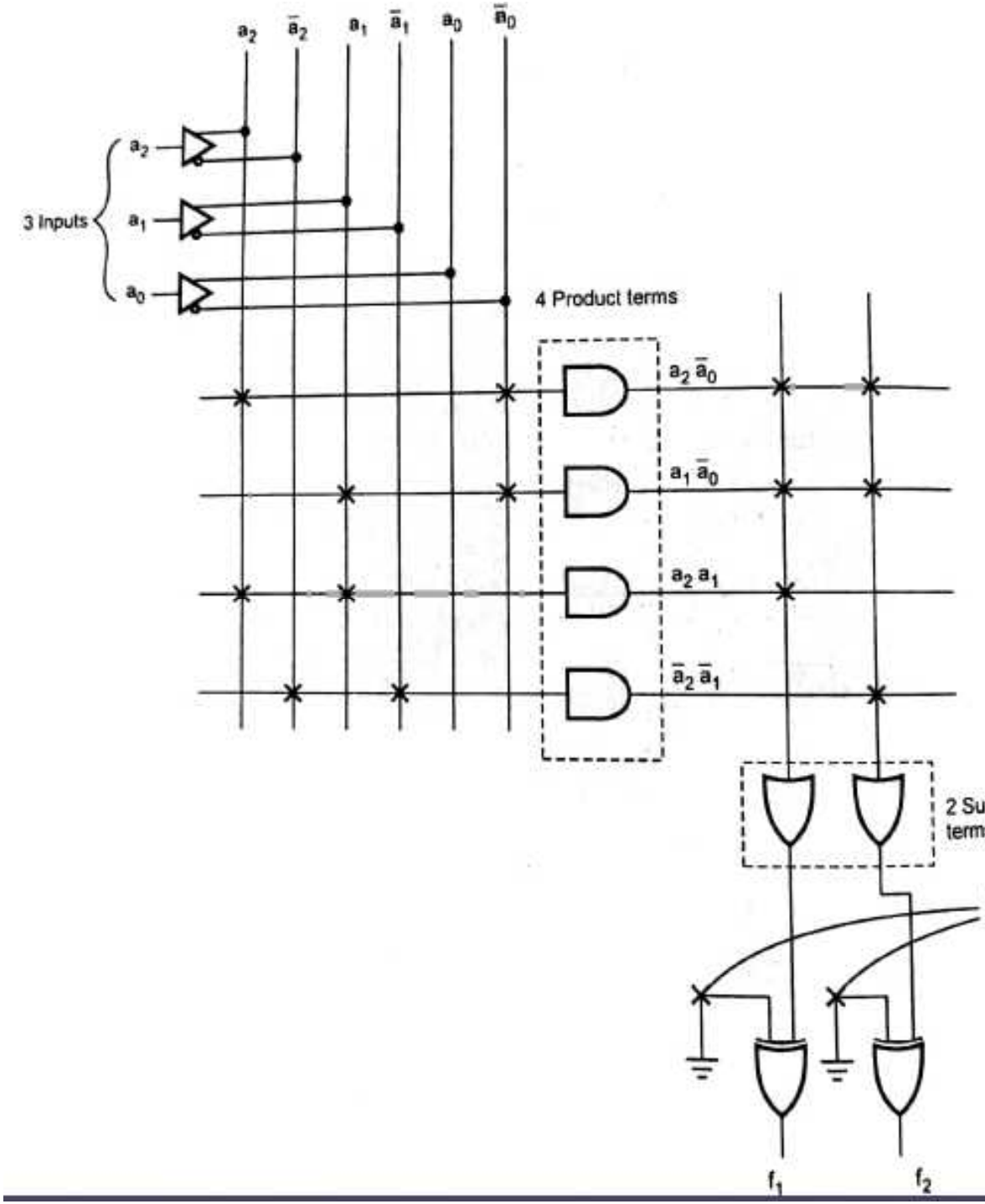
$= \bar{a}_2 \bar{a}_1 + a_1 \bar{a}_0 + a_2 \bar{a}_0$

Product terms	Inputs			Outputs	
	a_2	a_1	a_0	f_1	
$a_2 \bar{a}_0$	1	–	0	1	
$a_1 \bar{a}_0$	–	1	0	1	
$a_2 a_1$	1	1	–	1	
$\bar{a}_2 \bar{a}_1$	0	0	–	–	
				C	

PLA Programming table

$$a_2\,\bar{a}_0 + a_1\,\bar{a}_0 + a_2\,a_1$$

$$\bar{a}_2\,\bar{a}_1 + a_1\,\bar{a}_0 + a_2\,\bar{a}_0$$



3 inputs, 4 product terms and 2 outputs

Programmable Array Logic (PAL)

grammable Array Logic (PAL) is a Logic device with programmable AND owed by fixed OR gates

lement the given functions using PAL architecture

$w(A, B, C, D) = \sum m(0, 2, 6, 7, 8, 9,$
$x(A, B, C, D) = \sum m(0, 2, 6, 7, 8, 9,$
$y(A, B, C, D) = \sum m(2, 3, 8, 9, 10, 12$
$z(A, B, C, D) = \sum m(1, 3, 4, 6, 9, 12,$

For w

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

$y = \bar{A}B\bar{D} + \bar{A}BC + A\bar{C}$

For x

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

$x = \bar{A}B\bar{D} + \bar{A}BC + A\bar{C} + BC\bar{D}$

For y

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

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

For z

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

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

Programmable Logic Array (PAL)

$\overline{D} + \overline{A}BC + A\overline{C}$

$\overline{} + \overline{A}BC + A\overline{C} + BC\overline{D}$

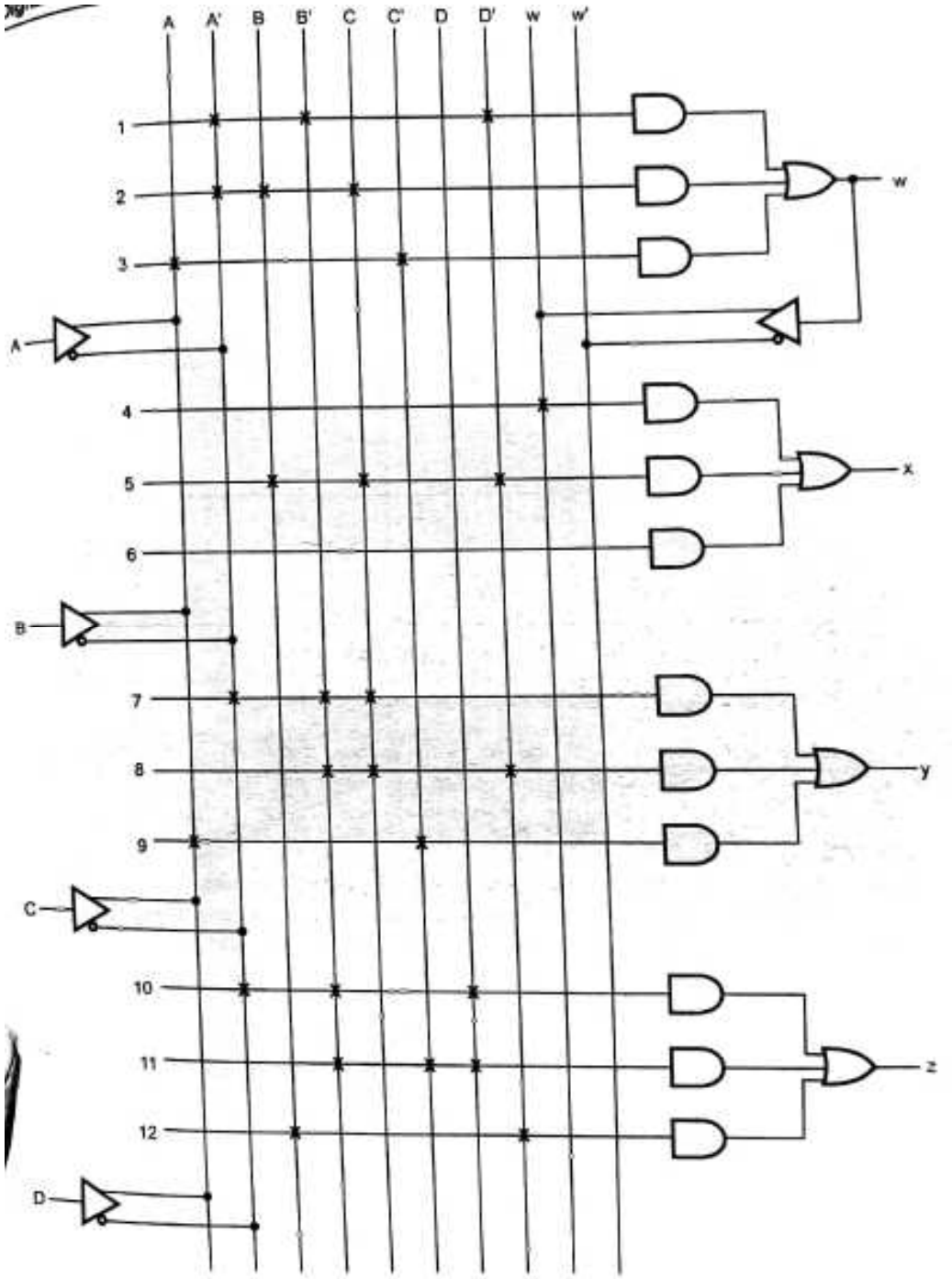
$\overline{C} + \overline{B}C\overline{D} + A\overline{C}$

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

Product term	AND Inputs					Outputs
	A	B	C	D	w	
1	0	0	–	0	–	$w = \overline{A}\,\overline{B}\overline{D} + \overline{A}\,BC + A\overline{C}$
2	0	1	1	–	–	
3	1	–	0	–	–	
4	–	–	–	–	1	$x = w + BC\overline{D}$
5	–	1	1	0	–	
6	–	–	–	–	–	
7	0	0	1	–	–	$y = \overline{A}\,\overline{B}C + \overline{B}C\overline{D} + A\overline{C}$
8	–	0	1	0	–	
9	1	–	0	–	–	
10	0	0	–	1	–	$z = \overline{A}\,\overline{B}D + \overline{B}\overline{C}D + B\overline{D}$
11	–	0	0	1	–	
12	–	1	–	0	–	

gramming table

nmable Logic Array (PAL)



3-wide AND-OR structure and 4 outputs

UNIT-IV

SEQUENTIAL MACHINE DESIGN PRACTICES

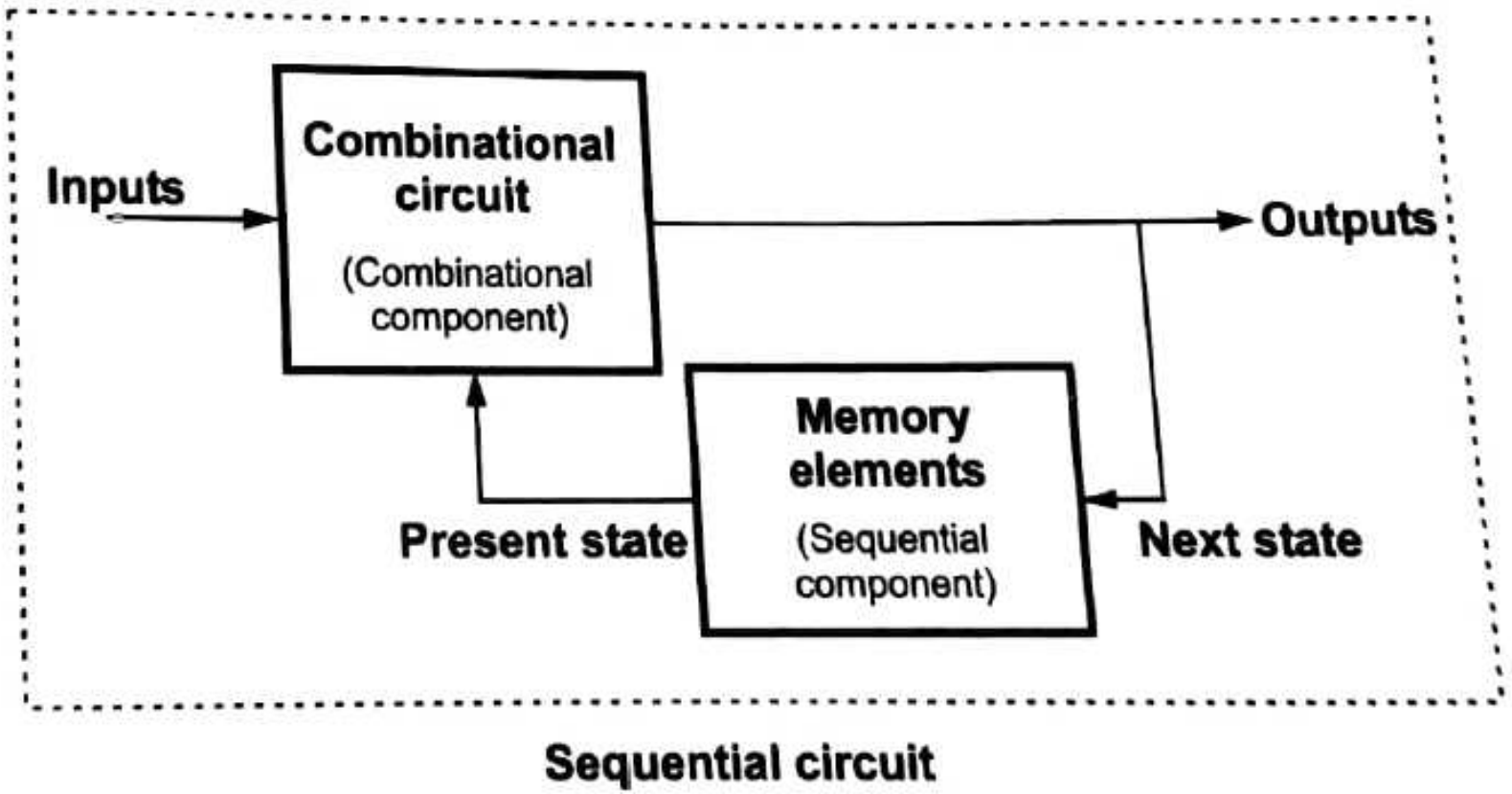
w of design of State machines; Standard building block ICs for Shift registers, par

conversion , shift register counters, Ring counters; Johnson counters, LFSR cou

L models for the above standard building block ICs. Synchronous Design example

ard ICs.

SEQUENTIAL CIRCUITS



Block diagram of Sequential Circuits

SEQUENTIAL CIRCU

Sr. No.	Combinational circuits	Sequential circuits
1.	In combinational circuits, the output variables are at all times dependent on the combination of input variables.	In sequential circuits, the output variables depend not only on the present input variables but they also depend upon the past history of these input variables.
2.	Memory unit is not required in combinational circuits.	Memory unit is required to store the past history of input variables in the sequential circuit.
3.	Combinational circuits are faster in speed because the delay between input and output is due to propagation delay of gates.	Sequential circuits are slower than the combinational circuits.
4.	Combinational circuits are easy to design.	Sequential circuits are comparatively hard to design.
5.	Parallel adder is a combinational circuit.	Serial adder is a sequential circuit.

Difference between combinational and Sequential Circuits

SEQUENTIAL CIRCU

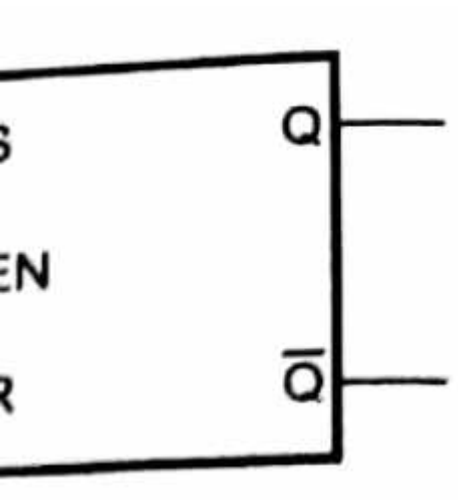
Synchronous sequential circuits	Asynchronous sequential circui
In synchronous circuits, memory elements are clocked flip-flops.	In asynchronous circuits, memory elen are either unclocked flip-flops or time elements.
In synchronous circuits, the change in input signals can affect memory element upon activation of clock signal.	In asynchronous circuits change in inp signals can affect memory element at instant of time.
The maximum operating speed of clock depends on time delays involved.	Because of absence of clock, asynchr circuits can operate faster than synchr circuits.
Easier to design.	More difficult to design.

Difference between Synchronous and l and Asynchronous Sequential Circuits

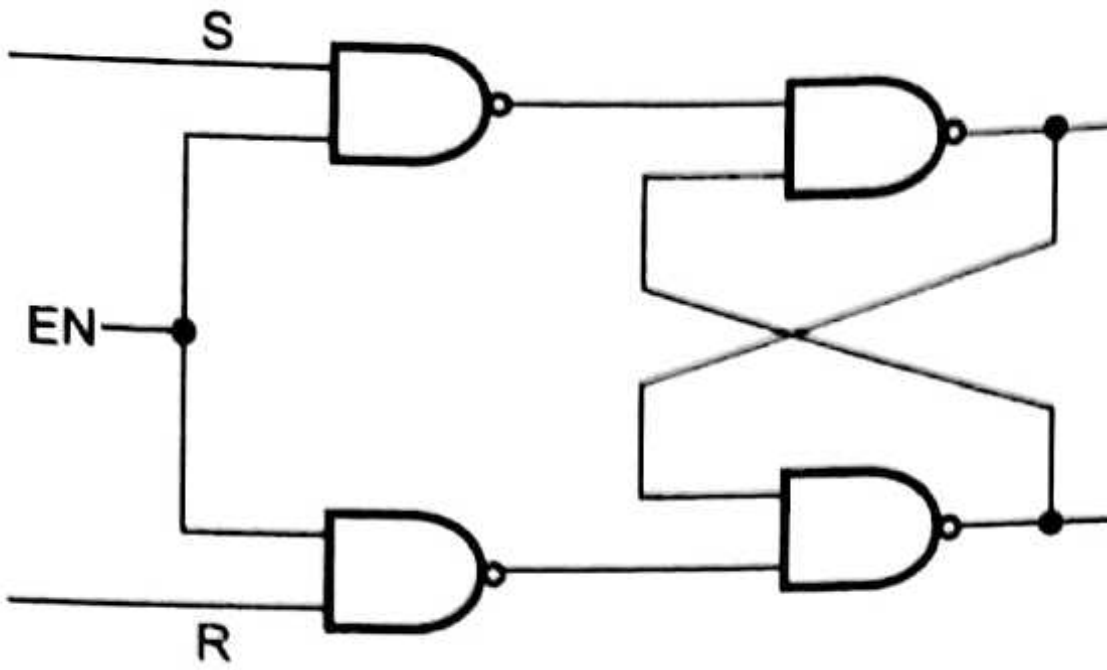
h

ch is an electronic logic circuit with two stable states i.e. it is a bistable multivibrato
ch has a feedback path to retain the information.
ce a latch can be a memory device.
ch can store one bit of information as long as the device is powered on.

atch



mbol



SR Latch Using NAND gates

tch

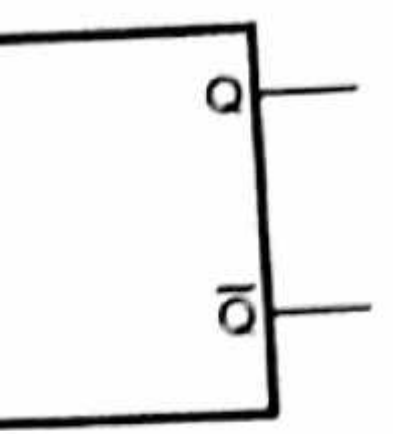
EN	S	R	Q_n	Q_{n+1}	State
1	0	0	0	0	No change (NC)
1	0	0	1	1	
1	0	1	0	0	Reset
1	0	1	1	0	
1	1	0	0	1	Set
1	1	0	1	1	
1	1	1	0	X	Indeterminate
1	1	1	1	X	
0	X	X	0	0	No change (NC)
0	X	X	1	1	

SR Latch truth table

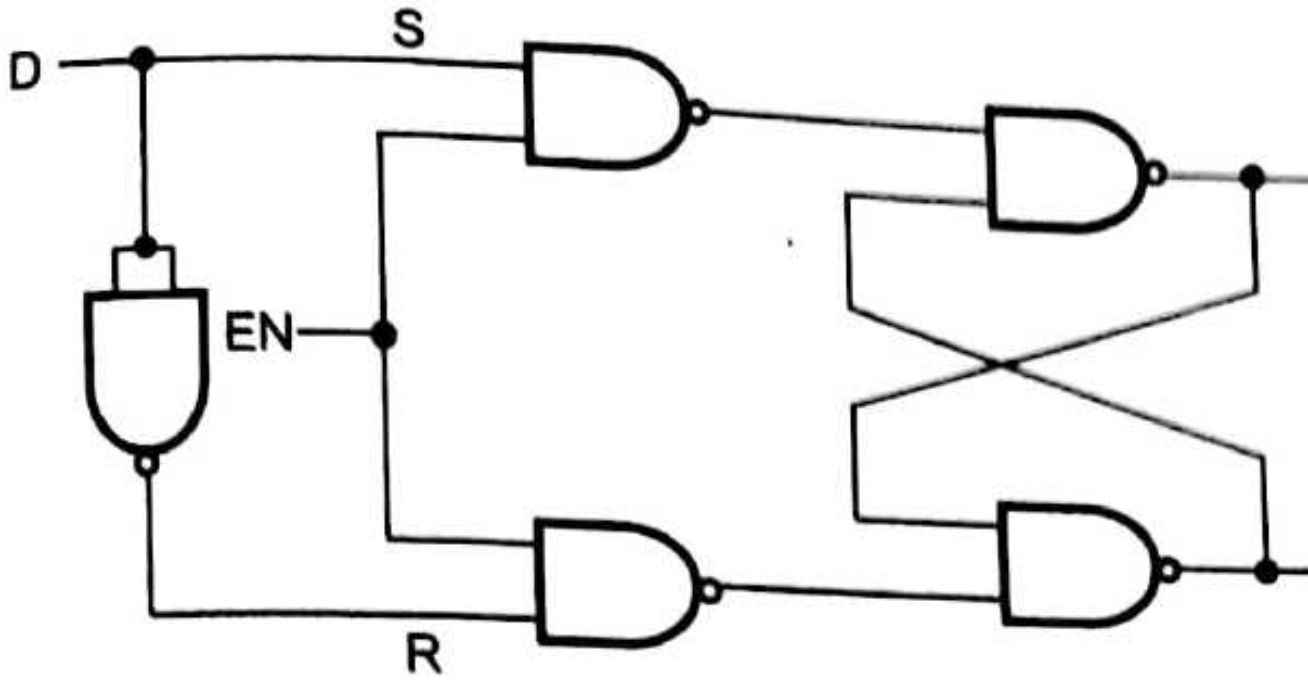
cteristic Equation

$= S + \overline{R} Q_n$

tch



mbol



D L

ble

	D	Q_n	Q_{n+1}	State
	0	X	0	Reset
	1	X	1	Set
	X	X	Q_n	No change (NC)

Characteristic Equat

$$Q_{n+1} = EN \cdot D + \overline{EN}$$

l and Edge Triggering

Triggering

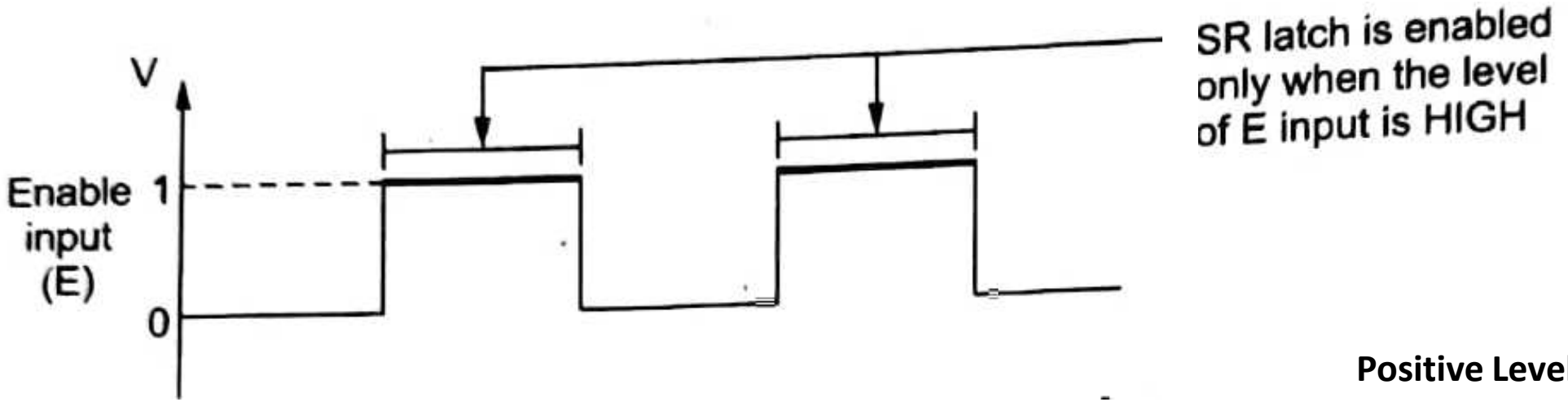
he level triggering the output state is allowed to change according to the input
ve level (**either positive or negative**) is maintained at the **enable** input.

ch is the level triggering

re are two types of Level triggering: (1) Positive Level Triggering
(2) Negative Level Triggering

ve Level Triggering:

tput of latch responds to the input changes only when its enable is **HIGH (1)**

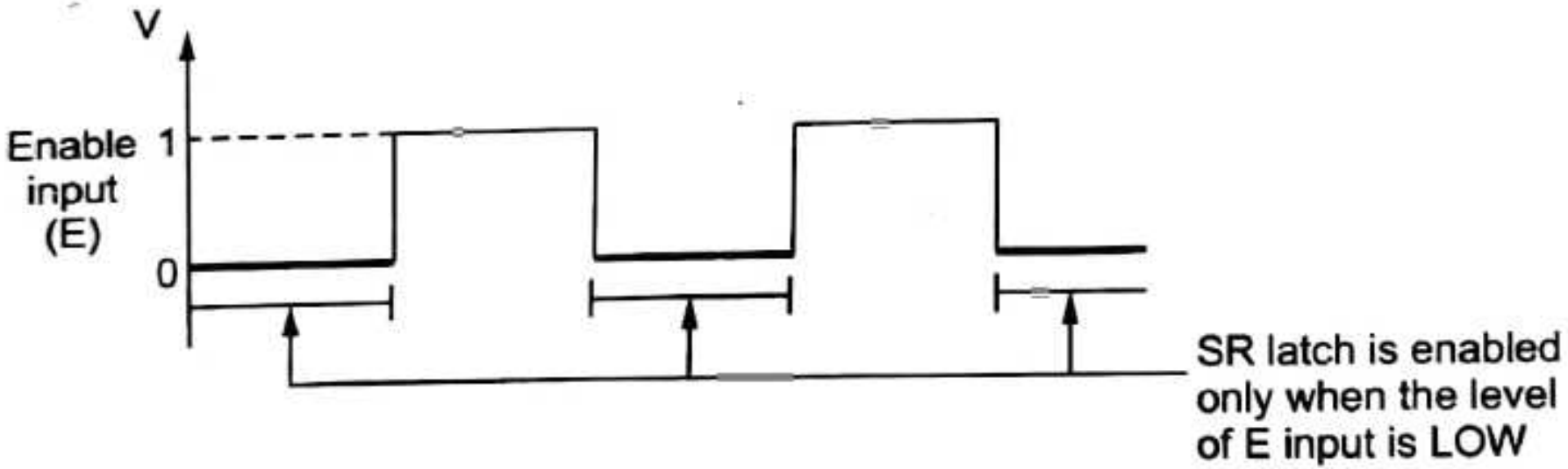


Positive Level Trig

l and Edge Triggering

ve Level Triggering:

tput of latch responds to the input changes only when its enable is **LOW (0)**



Negative Level Trig

Triggering

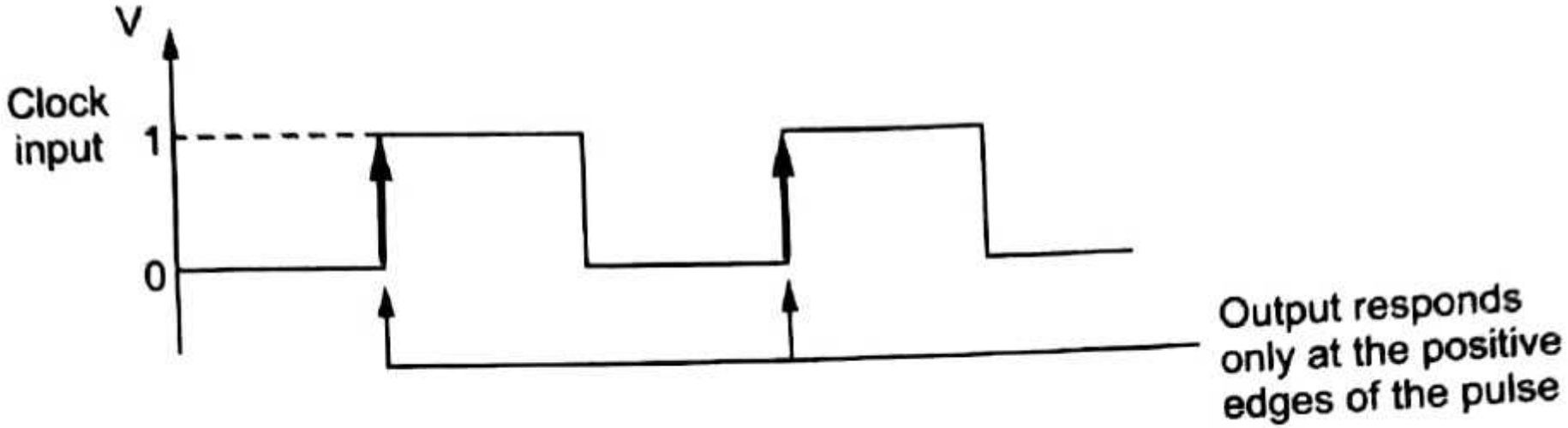
ne edge triggering the output responds to the changes in the input only at the p
egative edge of the clock pulse at the clock input.

-Flop is the edge triggering

re are two types of edge triggering: (1) Positive edge Triggering
(2) Negative edge Triggering

ve edge Triggering:

utput responds to the input changes only at the positive edge of the clock pulse
nput.

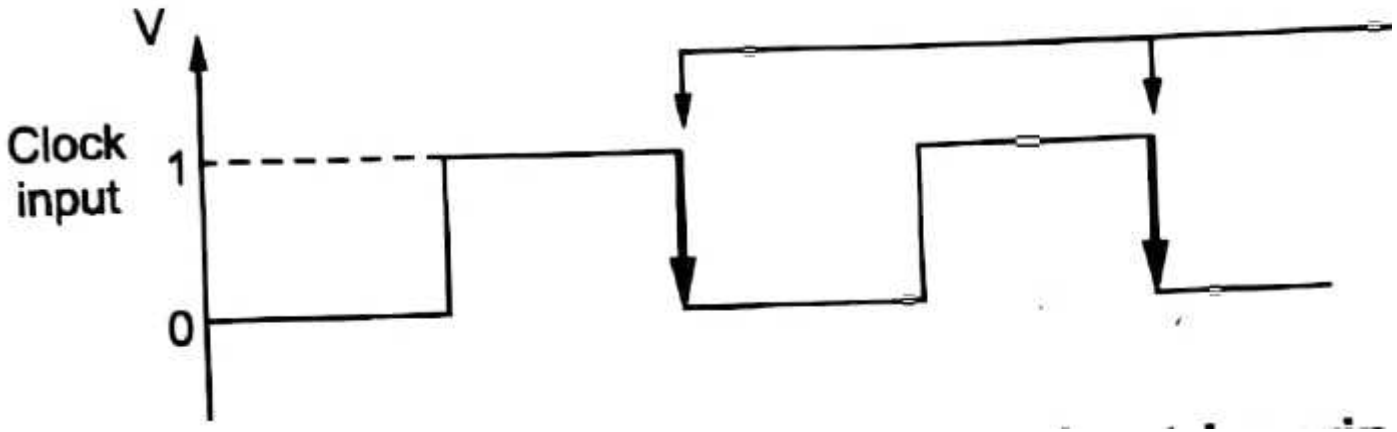


Positive edge Trigg

Triggering

ve edge Triggering:

utput responds to the input changes only at the negative edge of the clock pulse nput.

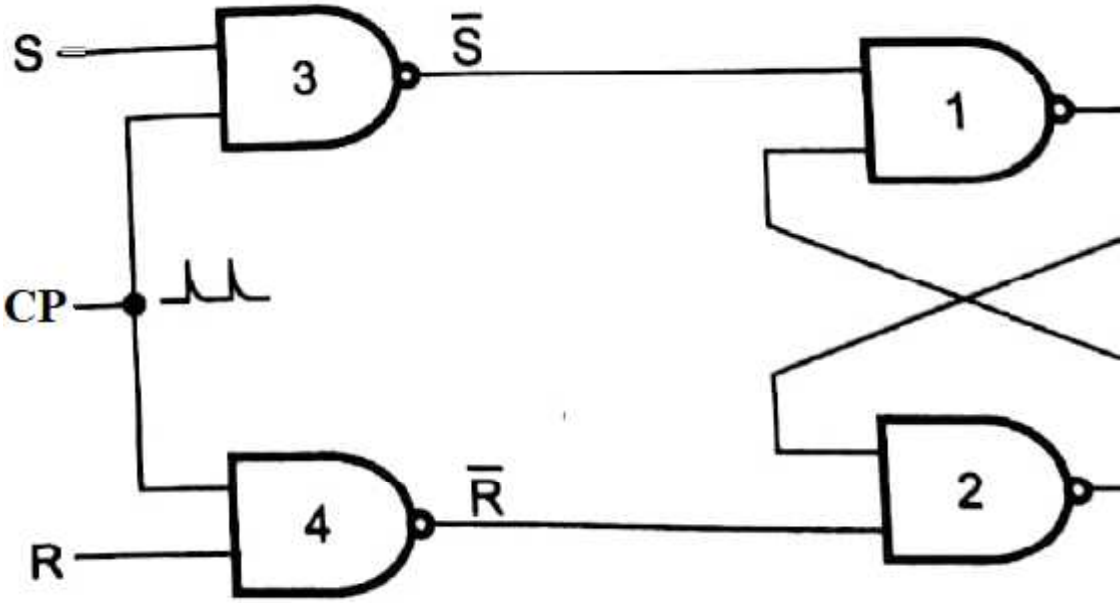


negative edge Trig

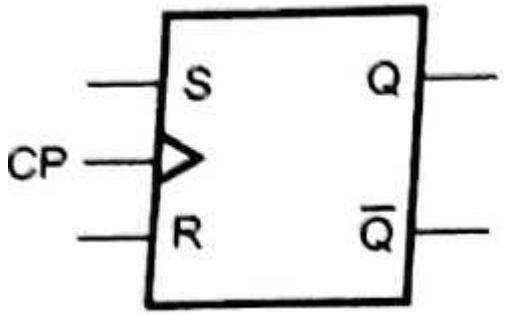
ked SR Flip-Flop

S	R	Q_n	Q_{n+1}	State
0	0	0	0	No change(NC)
0	0	1	1	
0	1	0	0	Reset
0	1	1	0	
1	0	0	1	Set
1	0	1	1	
1	1	0	X	Indeterminate
1	1	1	X	
X	X	0	0	No change(NC)
X	X	1	1	

h table for clocked SR flip-flop

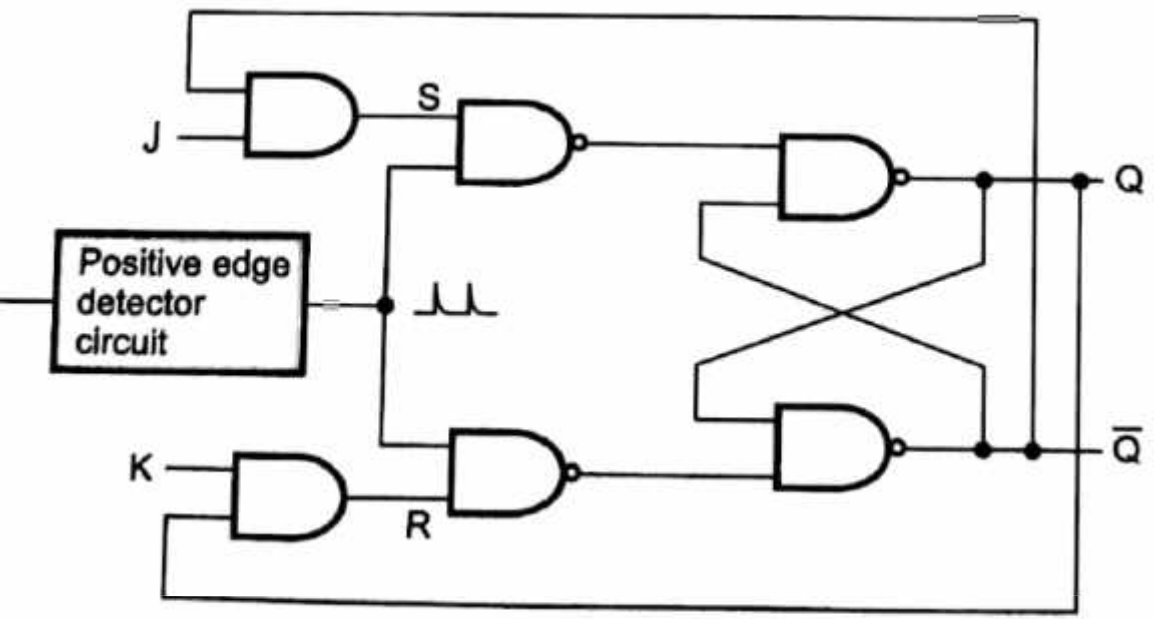


Positive edge clocked SR flip-flop

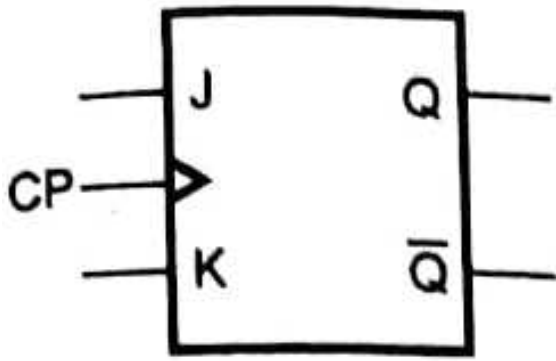


symbol

ed JK Flip-Flop



Positive edge clocked JK flip-flop



symb

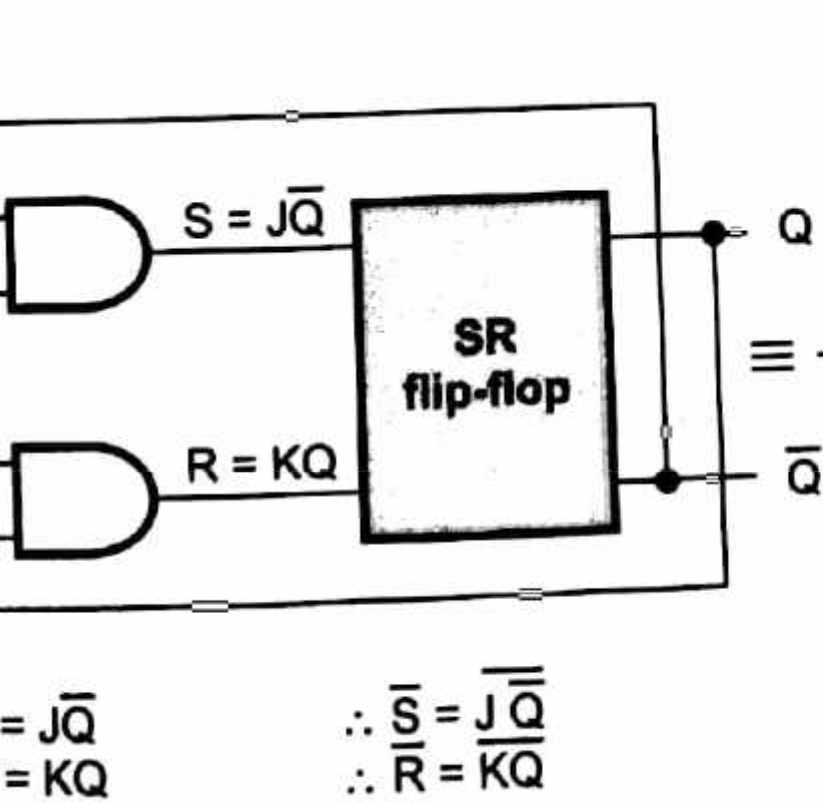
J	K	Q_{n+1}
0	0	0
0	1	0
1	0	1
1	1	1
0	0	1
0	1	0
1	0	1
1	1	0

≡

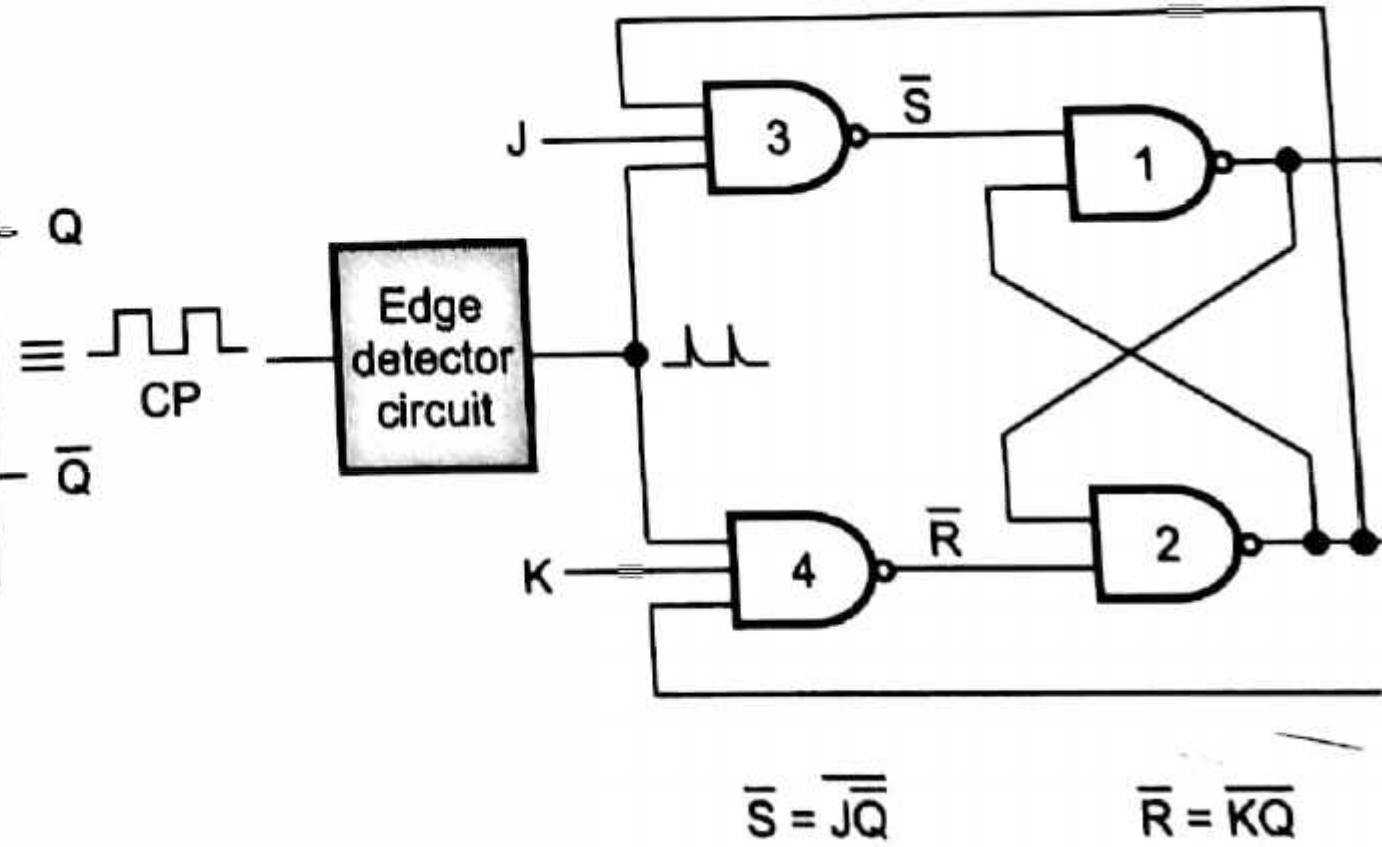
J	K	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	$\overline{Q_n}$

Truth table for clocked JK flip-flop

p-Flop Using NAND Gates

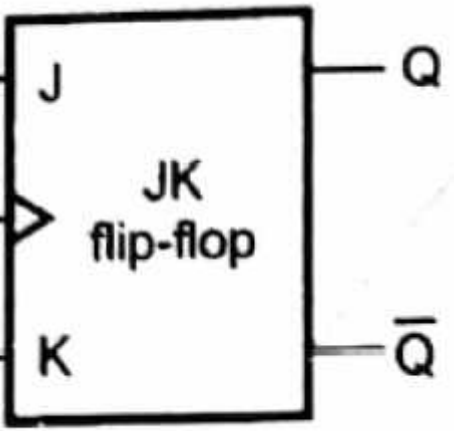


K Flip flop using SR Flip flop

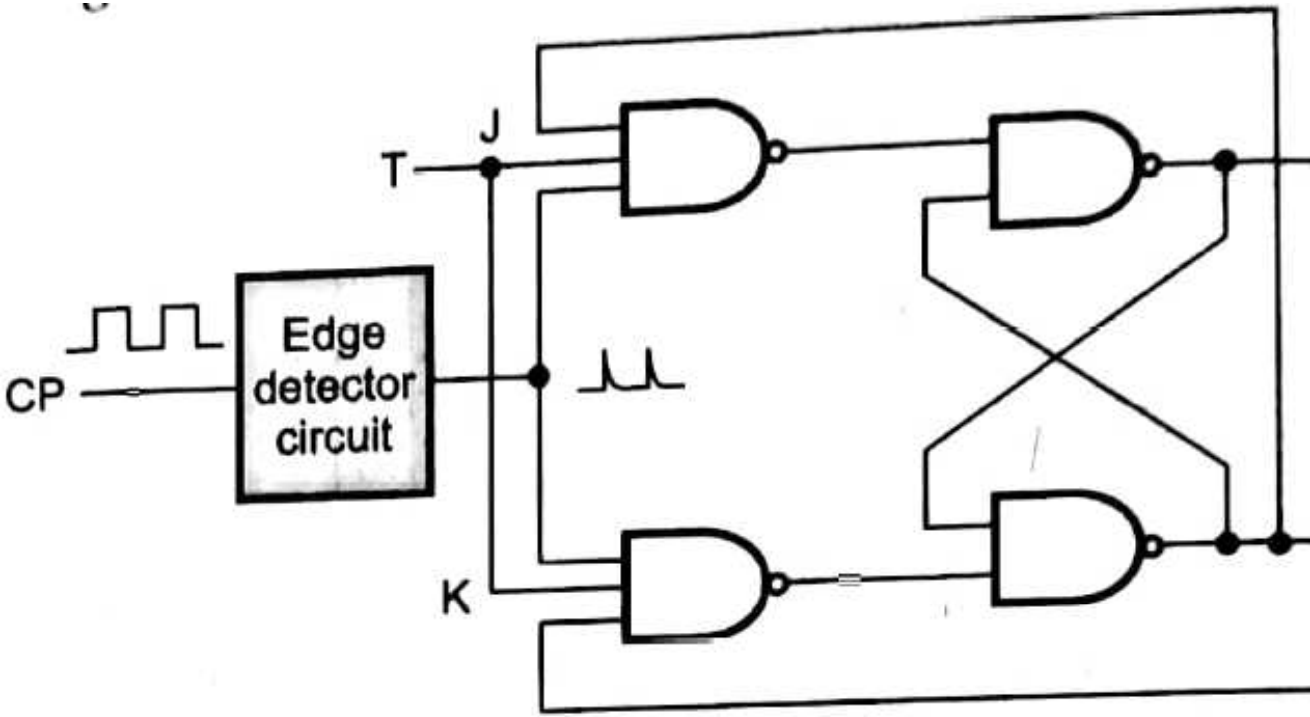


JK Flip flop using nand gates

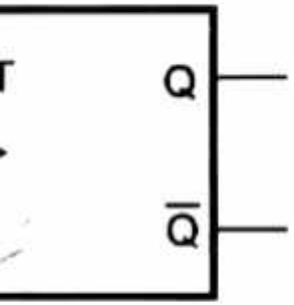
ed T Flip-Flop



bol



Positive edge clocked T flip-



ogic symbol

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

b) Truth table

≡

T	Q_{n+1}
0	Q_n
1	$\overline{Q_n}$

Q_n \ T	0	1
0	0	(1)
1	(1)	0

$\therefore Q_{n+1} = T\overline{Q}_n + \overline{T}Q_n$

(c) Characteristic equation

acteristic equation of all flip flops

Flip-flop	Characteristic equation
SR	$Q_{n+1} = S + \overline{R}\, Q_n$
D	$Q_{n+1} = D$
JK	$Q_{n+1} = J\, \overline{Q}_n + \overline{K} Q_n$
T	$Q_{n+1} = T\, \overline{Q}_n + \overline{T} Q_n$

tion table of all flip flops

FLOP

R	S	Q_{n+1}
0	0	Q_n
0	1	1
1	0	0
1	1	*

Truth table

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

Excitation table

tion table of all flip flops

P FLOP

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

Truth table

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

Excitation table

tion table of all flip flops

FLOP

D	Q_{n+1}
0	0
1	1

uth table

FLOP

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

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

Excitation table

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

p Flop to JK Flip Flop Conversion

Inputs		Present state	Next state	Flip-flop inputs	
J	K	Q_n	Q_{n+1}	S	R
0	0	0	0	0	X
0	0	1	1	X	0
0	1	0	0	0	X
0	1	1	0	0	1
1	0	0	1	1	0
1	0	1	1	X	0
1	1	0	1	1	0
1	1	1	0	0	1

Excitation table

p Flop to JK Flip Flop Conversion

implification

For S

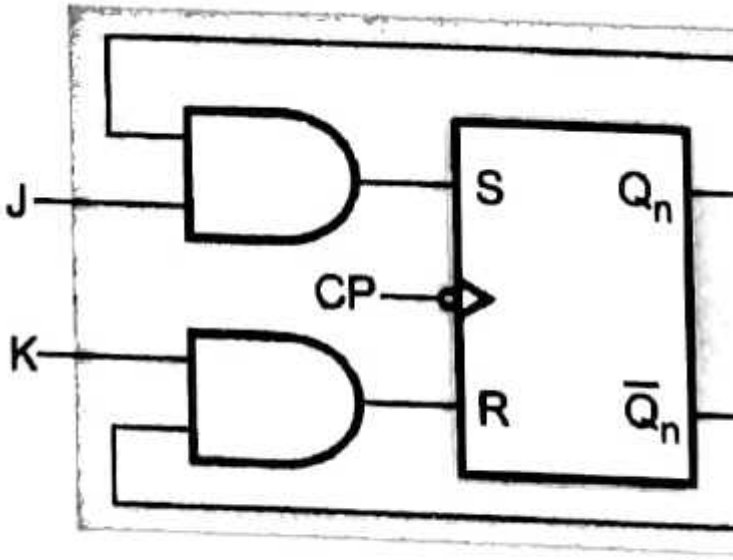
J \ K Q_n	00	01	11	10
0	0	X	0	0
1	1	X	0	1

$S = J\overline{Q}_n$

For R

J \ K Q_n	00	01	11	10
0	X	0	1	X
1	0	0	1	0

$R = K Q_n$



JK f

Logic Diagram

CODE FOR SR FLIP FLOP

```
rary ieee;
e ieee. std_logic_1164.all;
tity SR_FF is
ORT( S,R,CLK: in std_logic;
      Q: out std_logic);
d SR_FF;
rchitecture behavioral of SR_FF is
gin
ocess (CLK)
nal temp: std_logic;
gin
CLK‘ event and CLK='1') then
S='0' and R='0')then
emp <= temp;
```

```
elsif (S='0' and R='1')t
temp <='0';
elsif(S='1' and R=‘0')tl
temp <='1';
else
temp <='Z';
end if;
end if;
Q <= temp;
end PROCESS;
end behavioral;
```

CODE FOR D FLIP FLOP

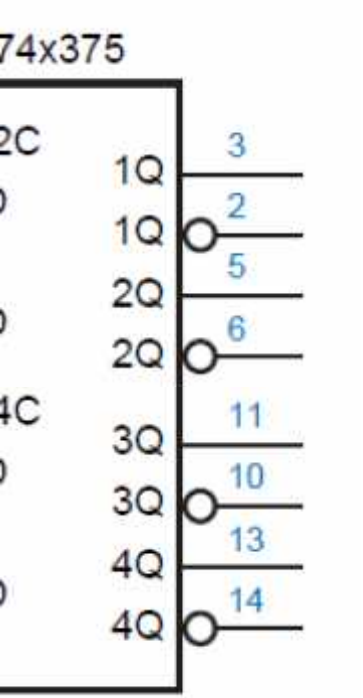

```
LIBRARY IEEE;
USE IEEE. std_logic_1164.all;
ENTITY DFF IS
PORT ( D, Clock : IN STD_LOGIC;
                Q : OUT STD_LOGIC);
END DFF;

ARCHITECTURE Behavior OF DFF IS
BEGIN
PROCESS (Clock)
BEGIN
  IF Clock'EVENT AND Clock = '1' THEN
    Q <= D;
  END IF;
END PROCESS;
END Behavior;
```

CODE FOR T FLIP FLOP

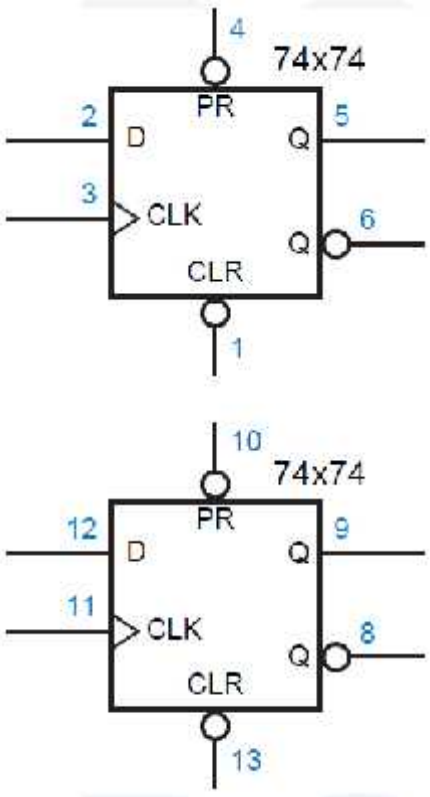
```
rary ieee;
e ieee. std_logic_1164.all;
tity T_FF is
rt( T,CLK: in std_logic;
      Q: out std_logic);
d T_FF;
rchitecture behavioral of T_FF is
gin
ocess (CLK)
nal temp: std_logic;
gin
CLK‘ event and CLK='1') then
T=‘0') then
mp <= temp;
else
temp <= not temp;
end if;
end if;
Q <= temp;
end process;
end behavioral;
```

TCHES AND FLIP FLOPS

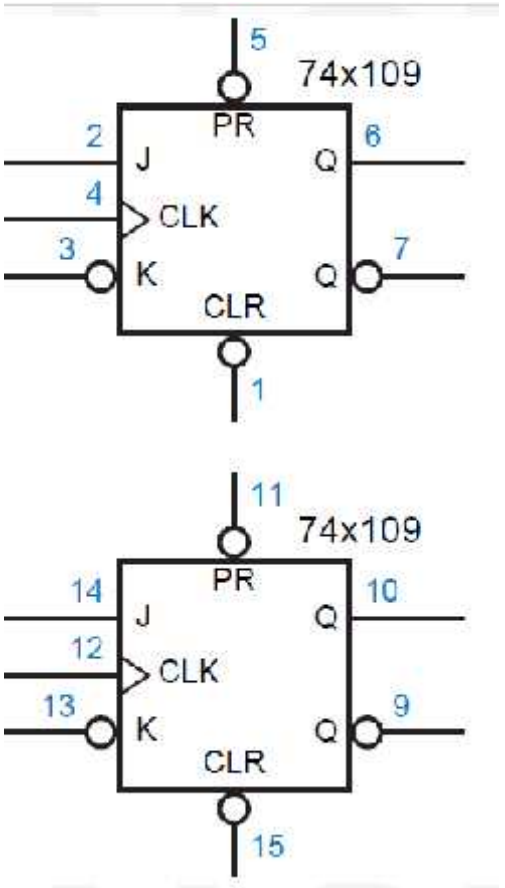
5 D latch



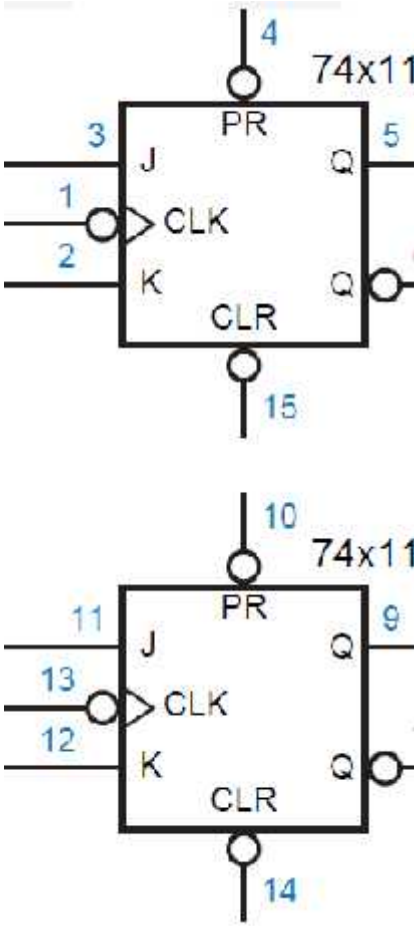
74X74 Edge triggered D Flip Flop



74X109 JK Flip Flop



74X112 JK Flip

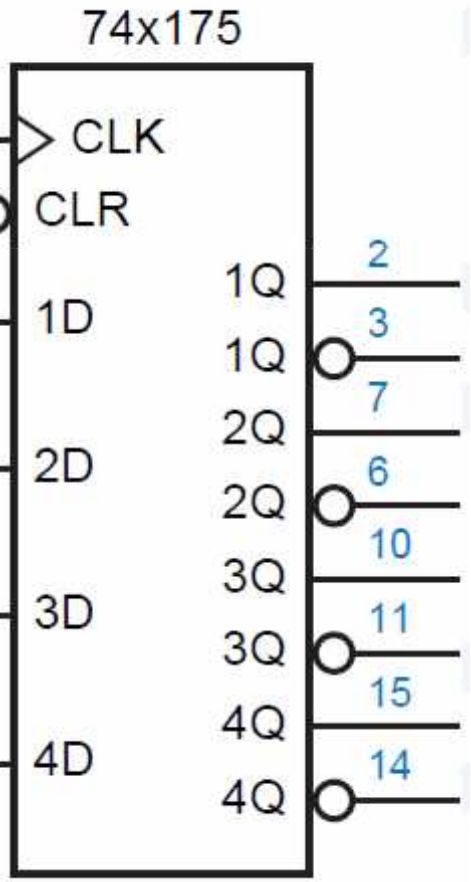


ters

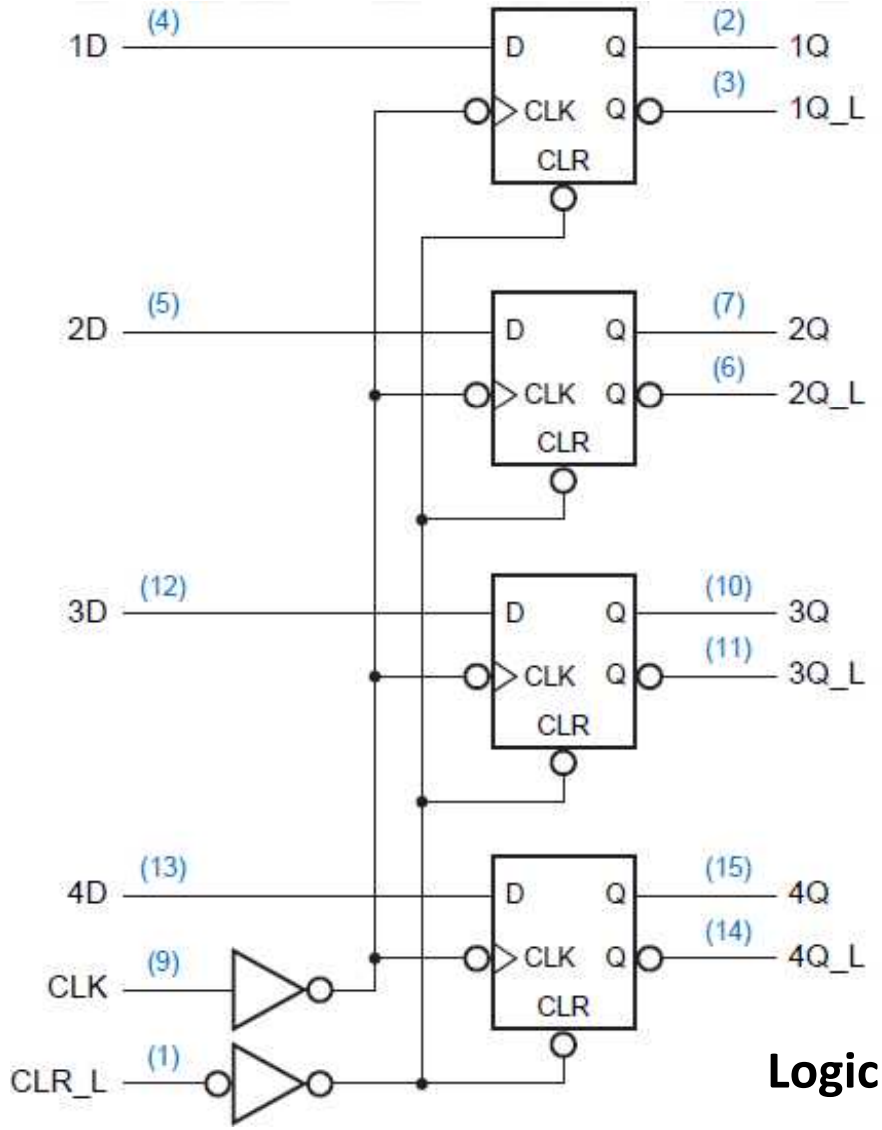
collection of two or more with a common clock input is called a register.

gisters are often used to store a collection of related bits such as a byte of data in mputer.

175 4-bit register



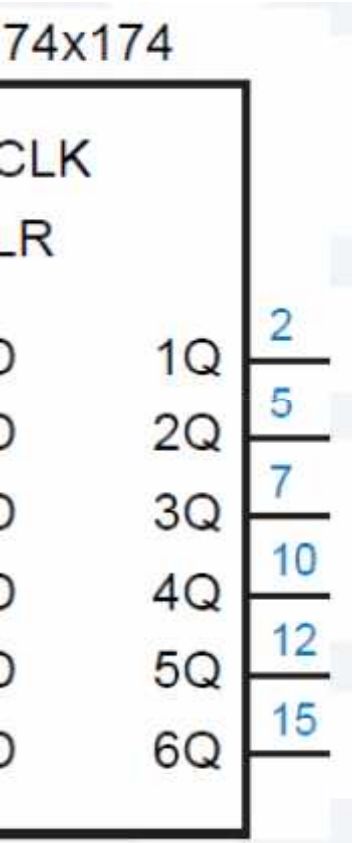
mbol



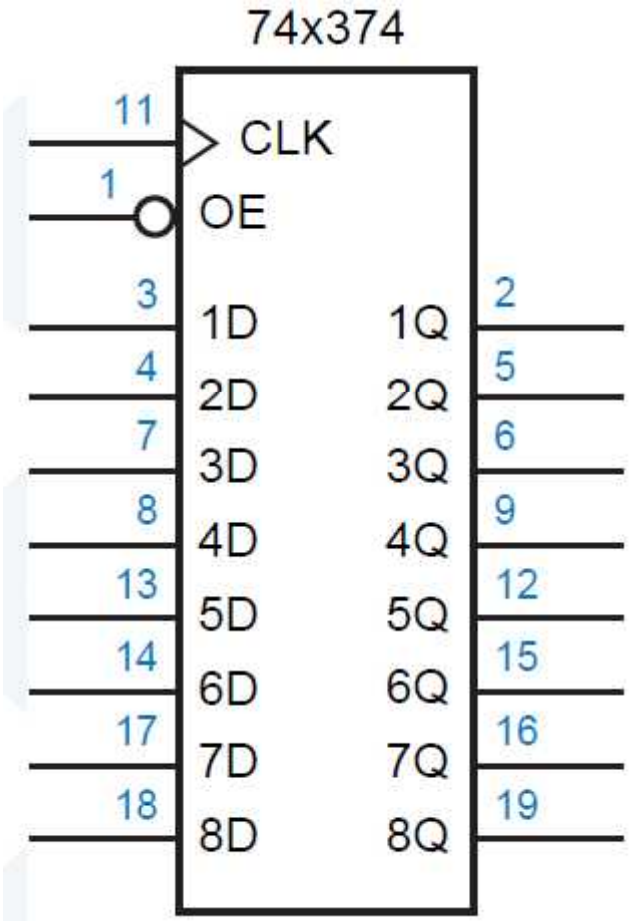
Logic Diagram

ters

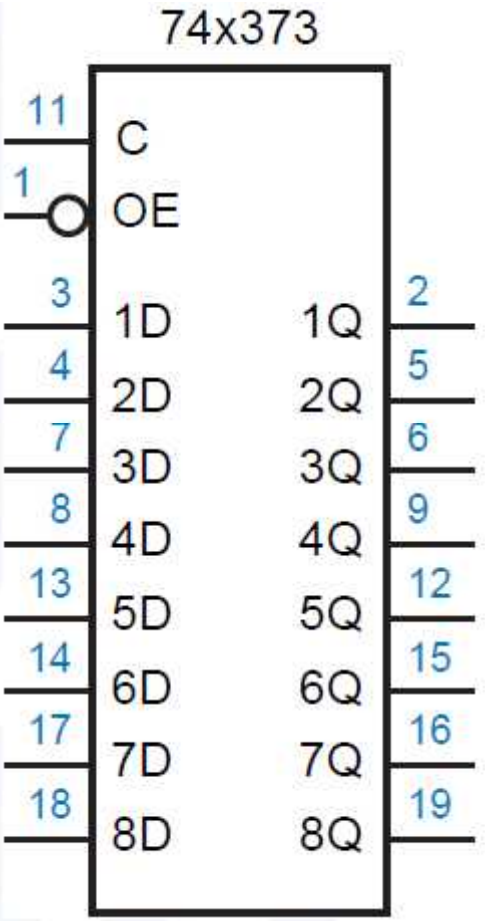
74 6 bit register



74x374 8 bit register



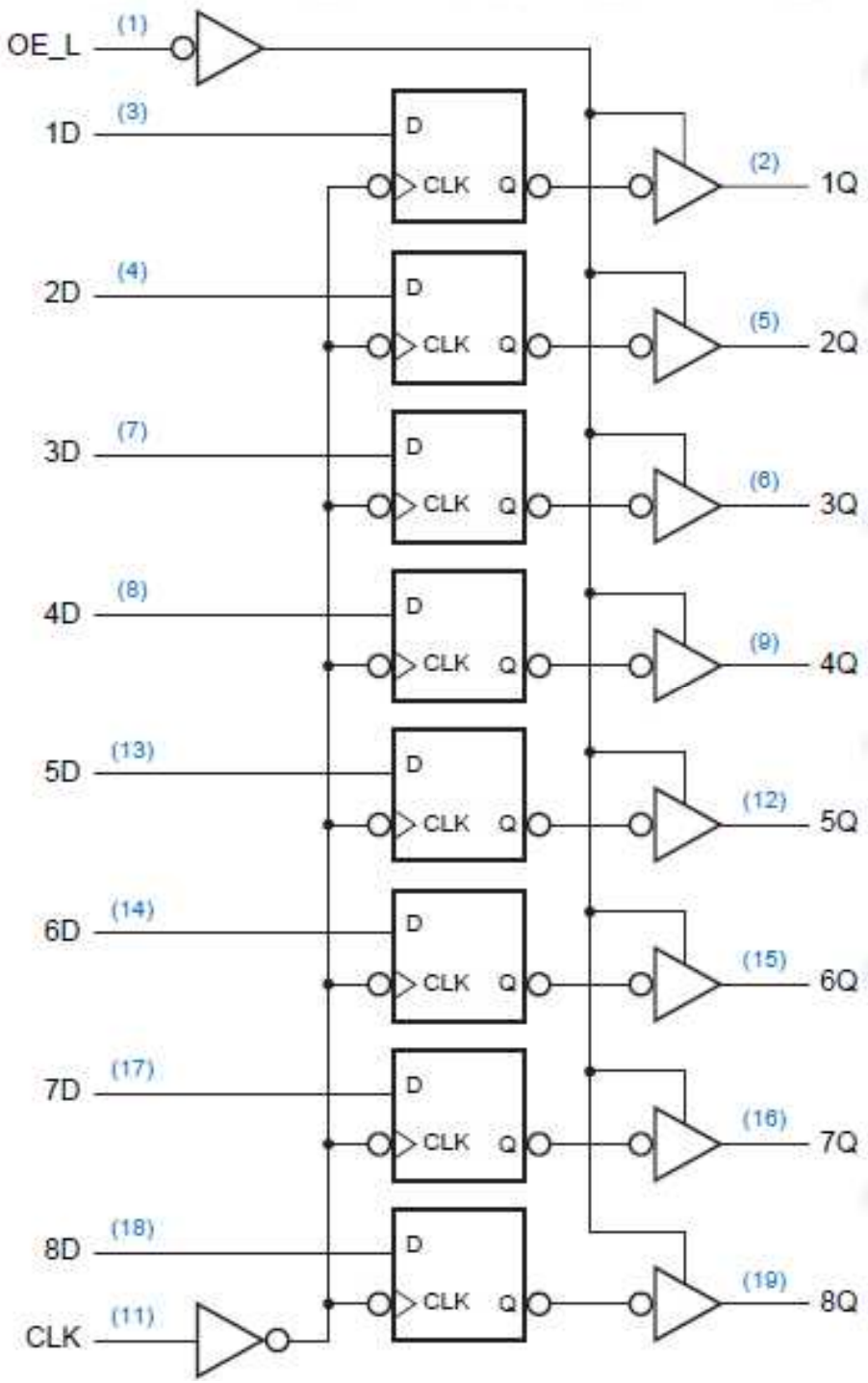
74x373 8-bit l



ymbol

ters

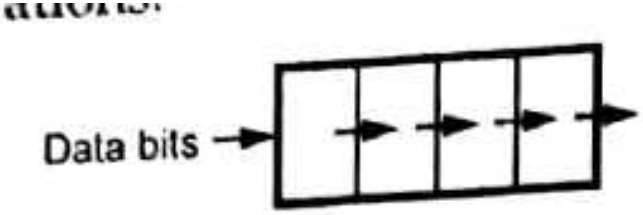
74 8 bit register



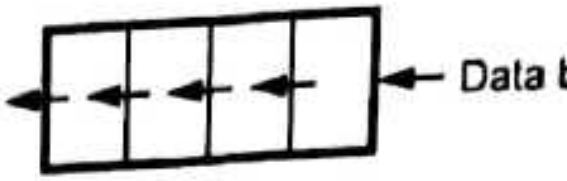
Logic Diagram

Registers

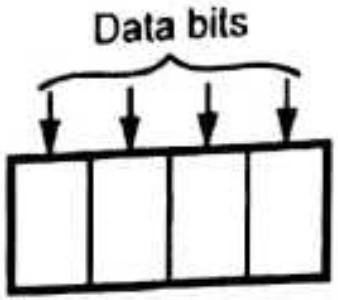
ata Movements in Registers



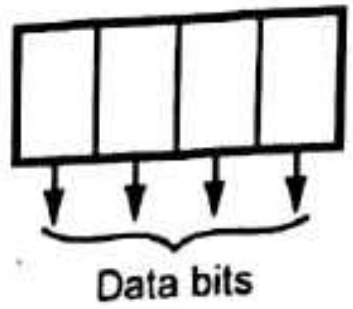
(a) Serial shift right, then out



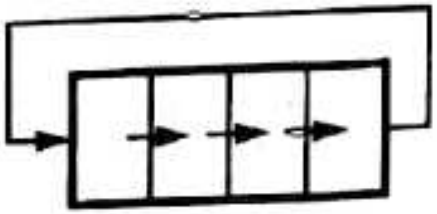
(b) Serial shift left, then out



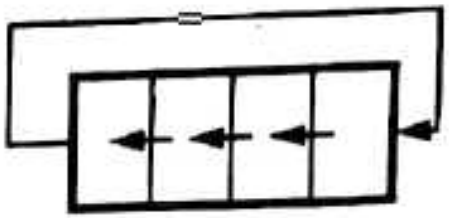
(c) Parallel shift in



(d) Parallel shift out



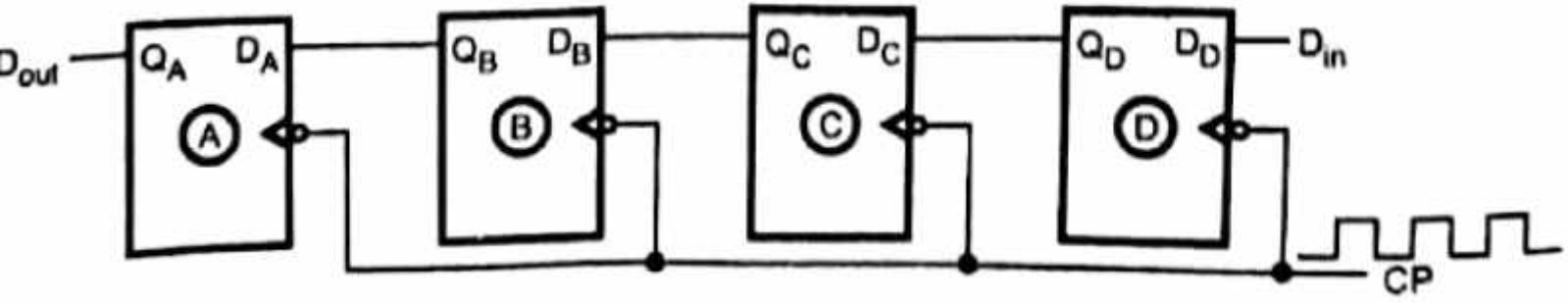
(e) Rotate right



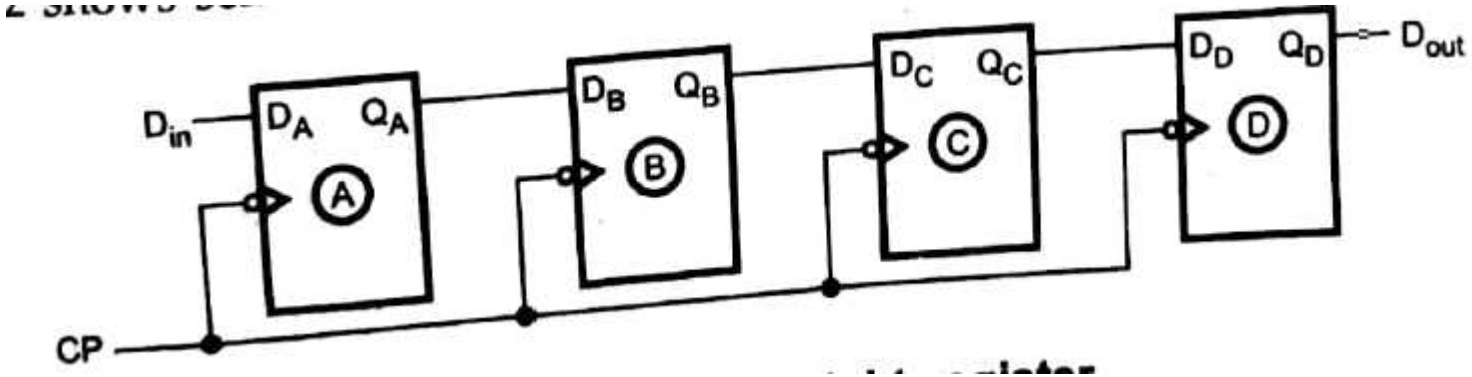
(f) Rotate left

Registers

n Serial Out Shift Register



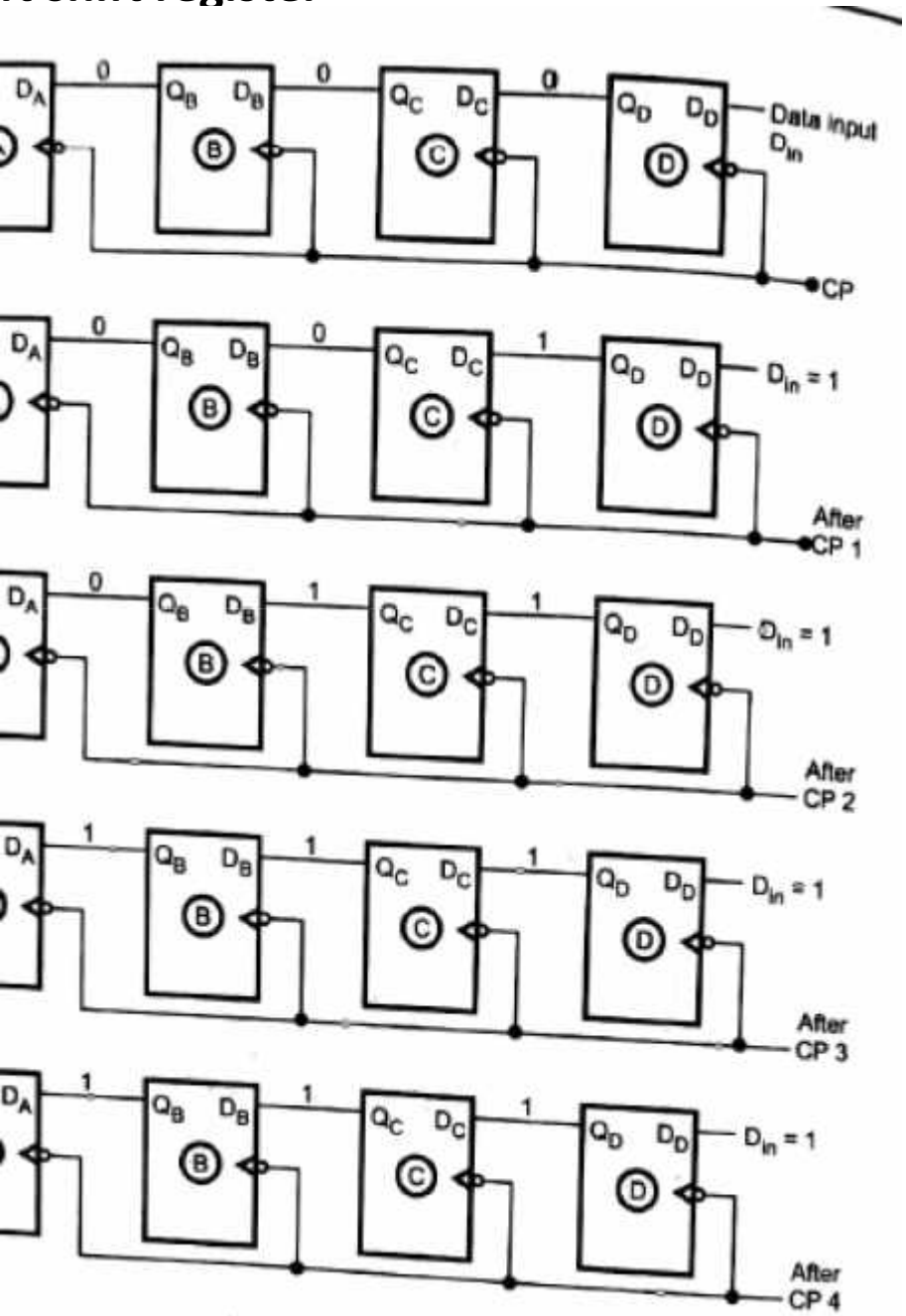
Shift left registe



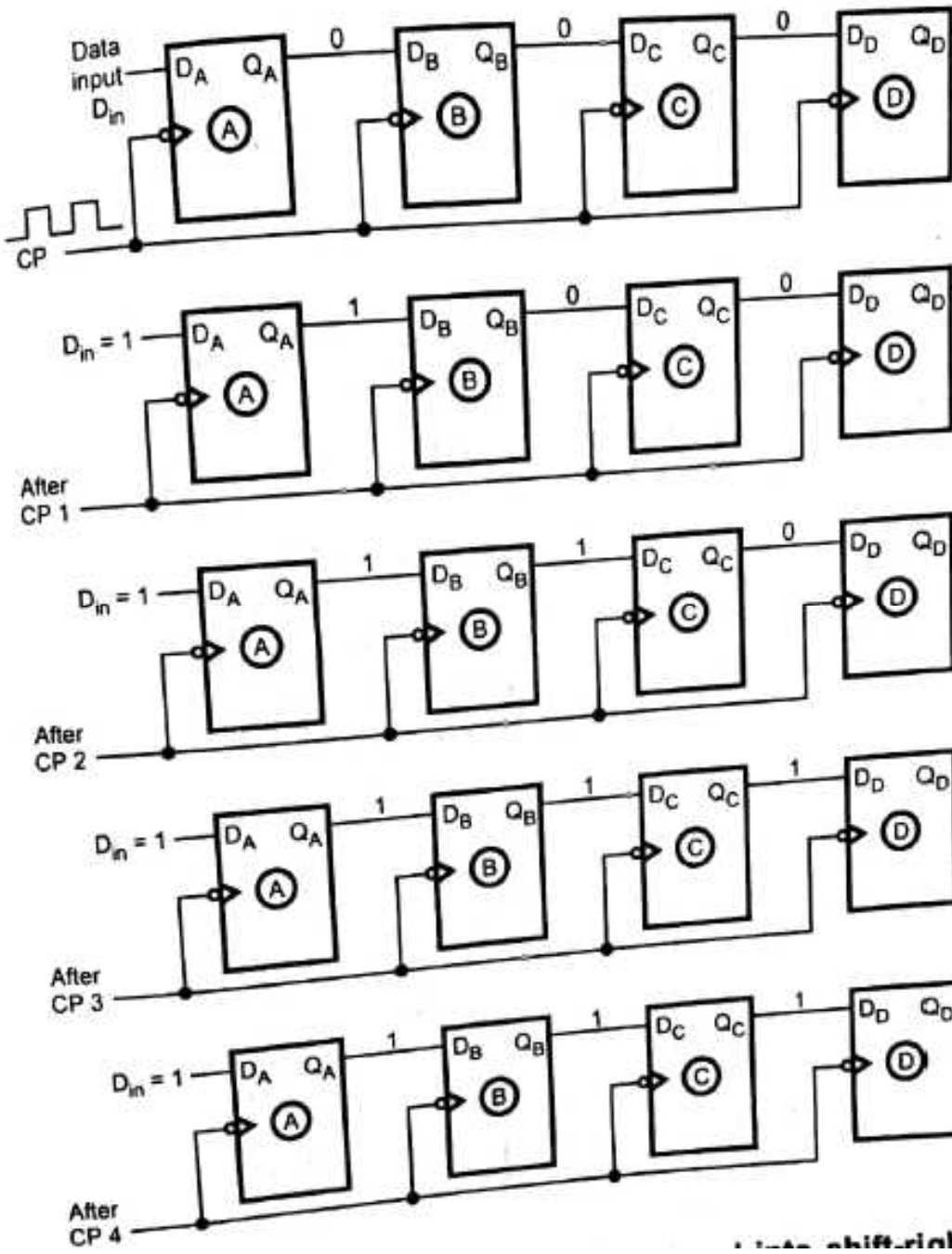
Shift right regist

Serial Out Shift Register

ft shift register

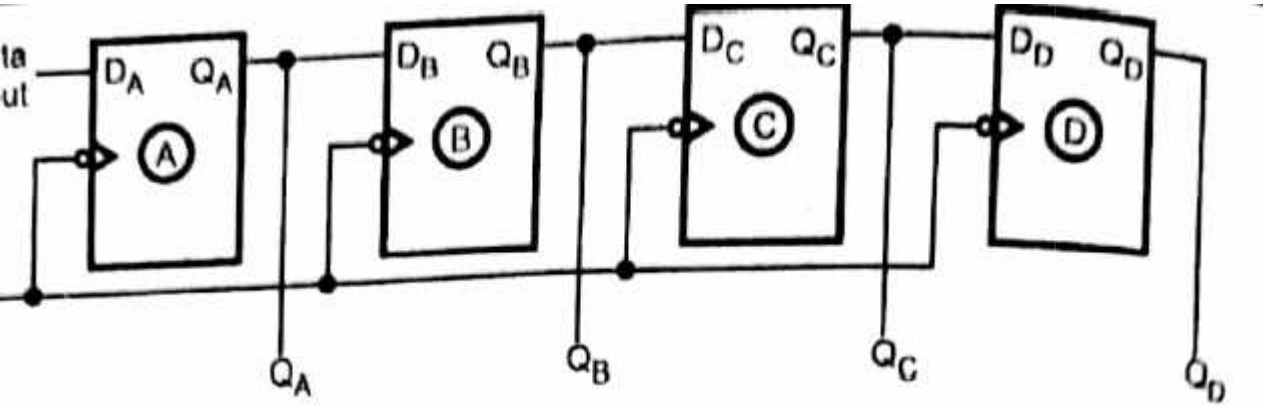


4 bit right shift register

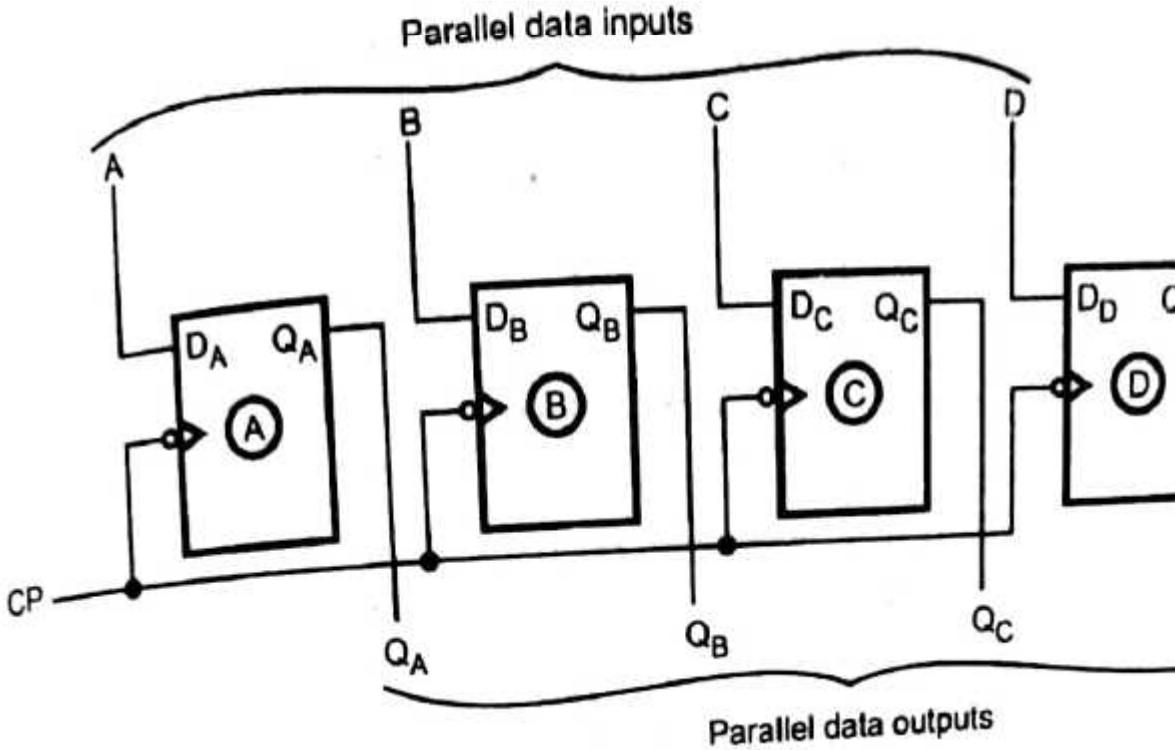


Registers

n Parallel Out Shift Register

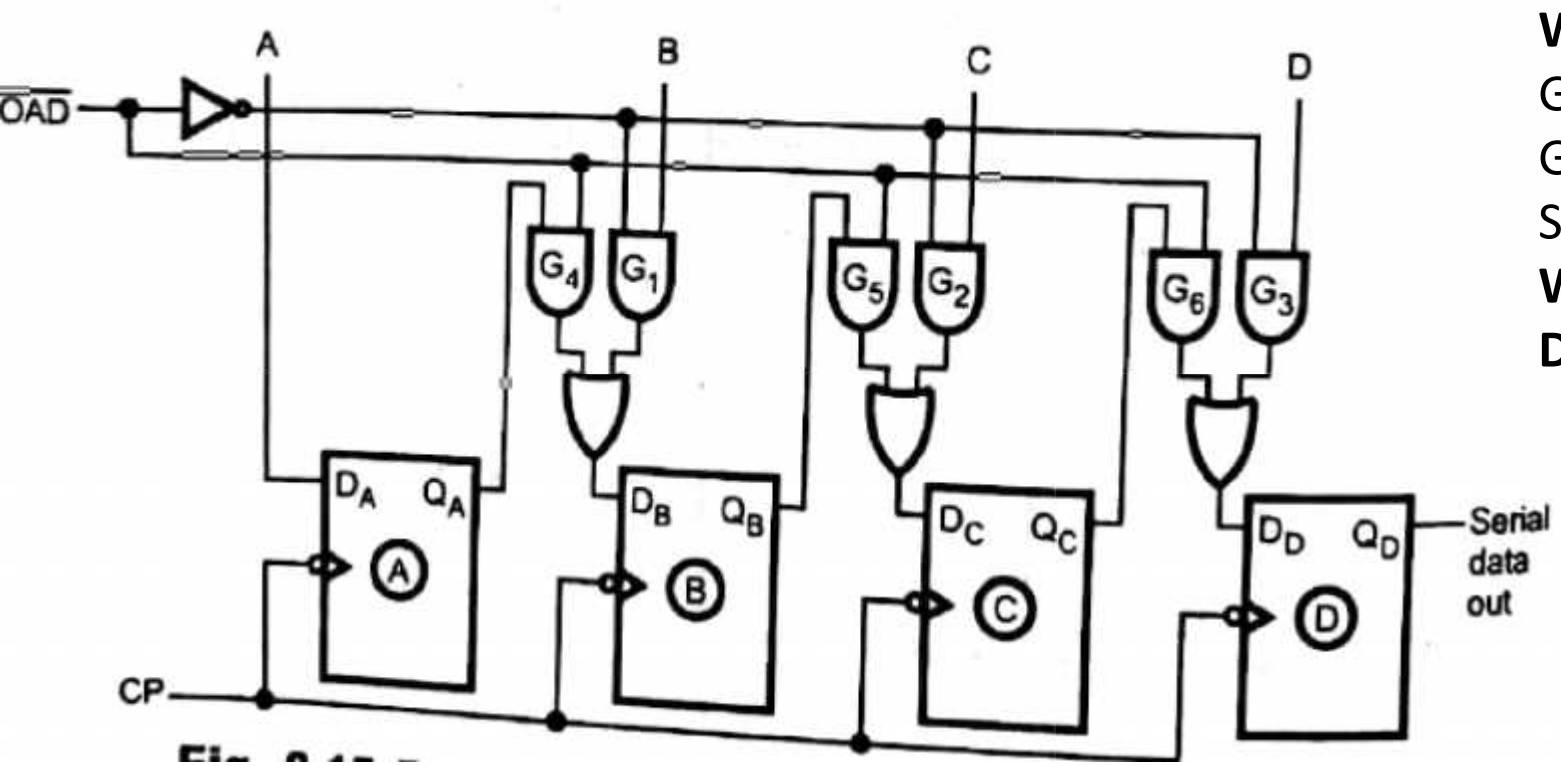


l In Parallel Out Shift Register



Registers

el In Serial Out Shift Register



Operation

When SHIFT/LOADbar=Hig

G1,G2,G3 are disabled

G4.G5,G6 are enabled and

Shift operation is performe

When SHIFT/LOADbar=Lov

Data is loaded

L Code for right shift register

```
ARY IEEE;
 IEEE.std_logic_1164.all;
ITY shiftreg IS
ORT(   Clock   : IN      STD_LOGIC;
       I       : IN      STD_LOGIC;
       Q       : BUFFER  STD_LOGIC_VECTOR (3 DOWNTO 0));
 shiftreg;
HITECTURE Behavior OF shiftreg IS
IN
ROCESS  (Clock, I)
EGIN
  WAIT UNTIL clock'EVENT AND Clock = '1';
                Q(0)<= Q(1);
                Q(1)<= Q(2);
                Q(2)<= Q(3);
                Q(3)<= I;
ND PROCESS;
 Behavior;
```

rogram for Serial In – Parallel Out Shift Registers

```
y ieee;
eee.std_logic_1164.all;
y sipo is
(
clear : in std_logic;
t_Data: in std_logic;
ut std_logic_vector(3 downto 0) );
sipo;
itecture arch of sipo is
n
cess (clk)
n
ar = '1' then
 "0000";
 (CLK'event and CLK='1') then
downto 1) <= Q(2 downto 0);
 <= Input_Data;
if;
process;
arch;
```

. Code for 4-bit Universal shift register

```
 IEEE;
EE.STD_LOGIC_1164.ALL;
EE.STD_LOGIC_unsigned.ALL;
usr74194 is
 ( clk, clr_l, lin, rin : in  STD_LOGIC;
s : in  STD_LOGIC_VECTOR (1 downto 0);
d : in  STD_LOGIC_VECTOR (3 downto 0);
q : inout  STD_LOGIC_VECTOR (3 downto 0));
r74194;

cture Behavioral of usr74194 is


s (clk, clr_l, lin, rin, s, d, q )


 = '0' then q<= "0000";
clk'event and clk = '1') then
```

```
case s is
        when "00" => q <= q;
        when "01" => q <= rin & q (3 do
        when "10" => q <= q (2 downto 0
        when "11" => q <= d;
        when others => q <= "UUUU";
end case;
end if;
end process;
end Behavioral;
```

registers Counters

hift register can be combined with combinational logic to form a state machine wh
e diagram is cycle.
h a circuit is called a Shift register counter.
ike a binary counter, a shift register counter does not count in an ascending or
cending binary sequence, but it is useful in many control applications nonetheless

Counter

mplest shift register counter uses an n bit shift register to obtain a counter with n
es and is called a ring counter.
74x194 universal shift register is wired so that it normally perform a left shift.

unter

Shift registers Cou

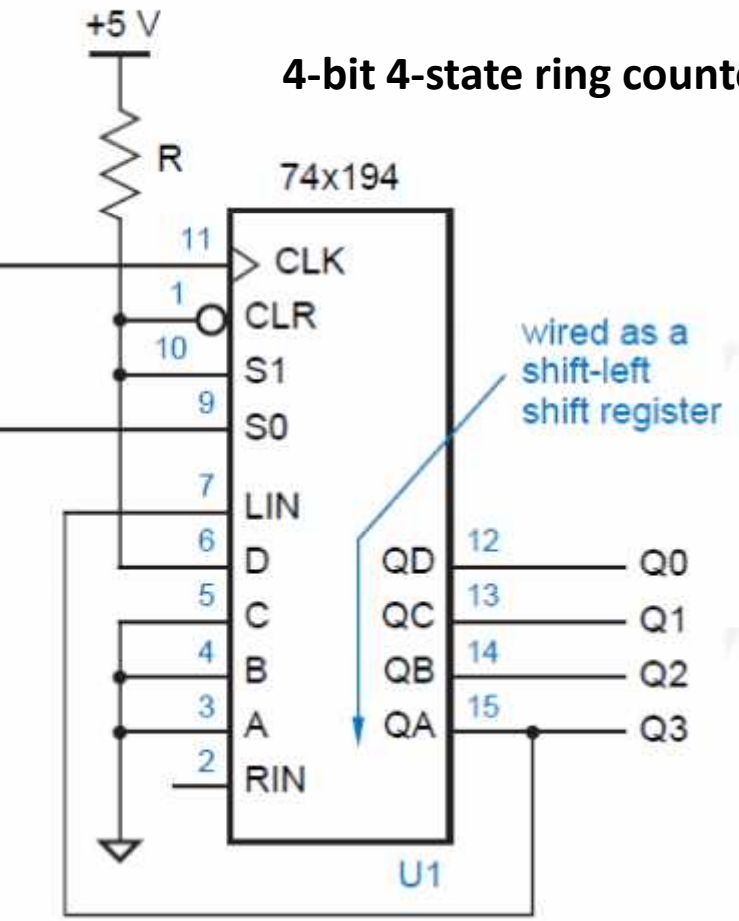
en RESET=1, it loads 0001

ce RESET is negated, the IC 74x194 shifts left on each clock.

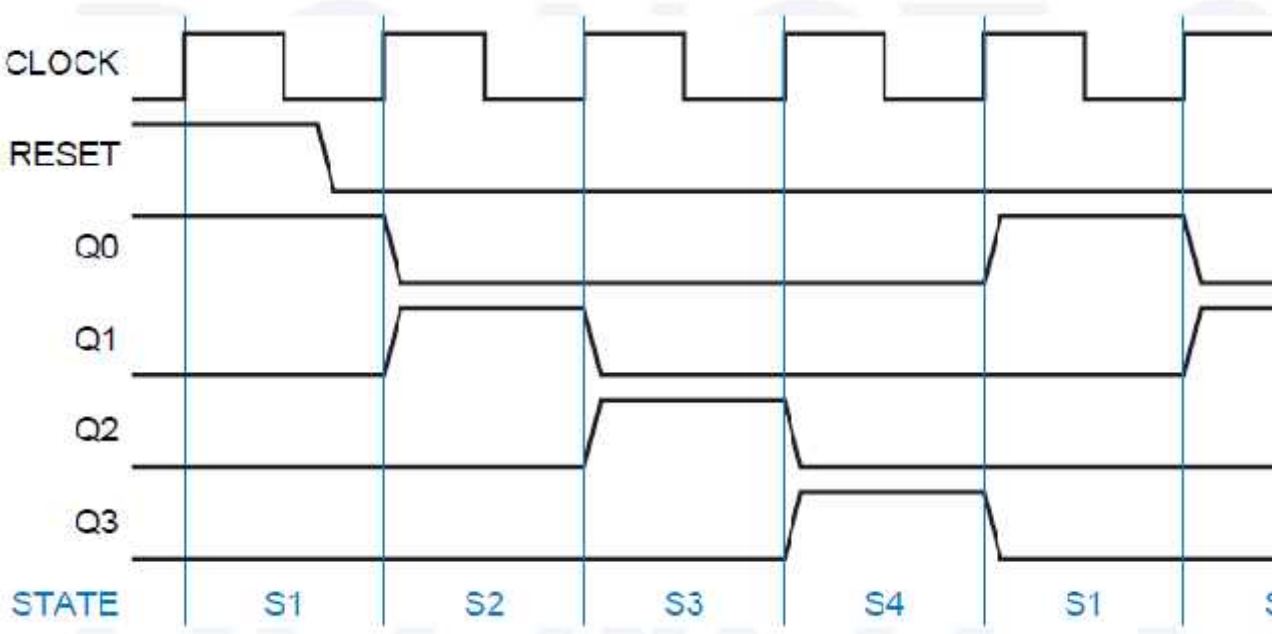
e LIN serial input is connected to the **leftmost** output, so that next states are

0,0100,1000,0001,0010… thus the counter visits four unique states before repeat

4-bit 4-state ring counter



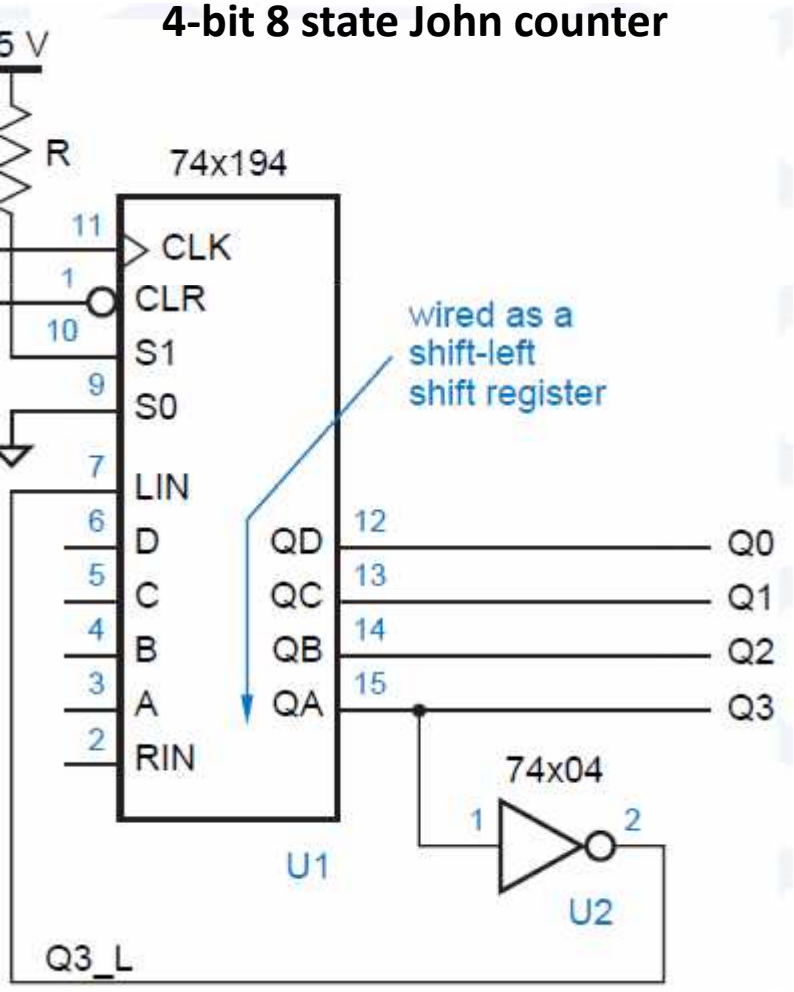
Timing diagram for 4-bit ring cour



on counter

n bit shift register with the complement of the serial output fed back into the se
ut is a counter with 2n states and is called a **twisted ring, Moebius or Johns
nter**

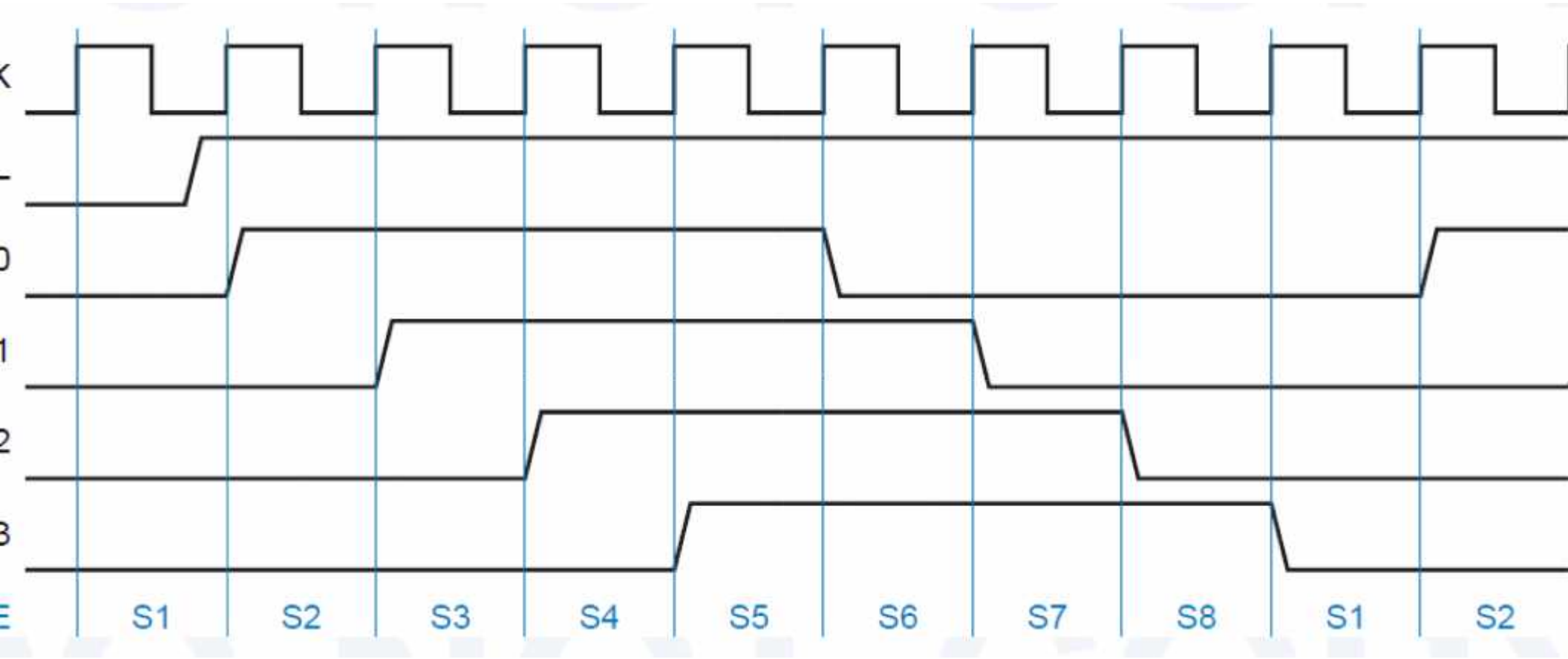
4-bit 8 state John counter



State Name	Q3	Q2	Q1	Q0	Decodi
S1	0	0	0	0	Q3′ · Q
S2	0	0	0	1	Q1′ · Q
S3	0	0	1	1	Q2′ · Q
S4	0	1	1	1	Q3′ · Q
S5	1	1	1	1	Q3 · Q
S6	1	1	1	0	Q1 · Q
S7	1	1	0	0	Q2 · Q
S8	1	0	0	0	Q3 · Q

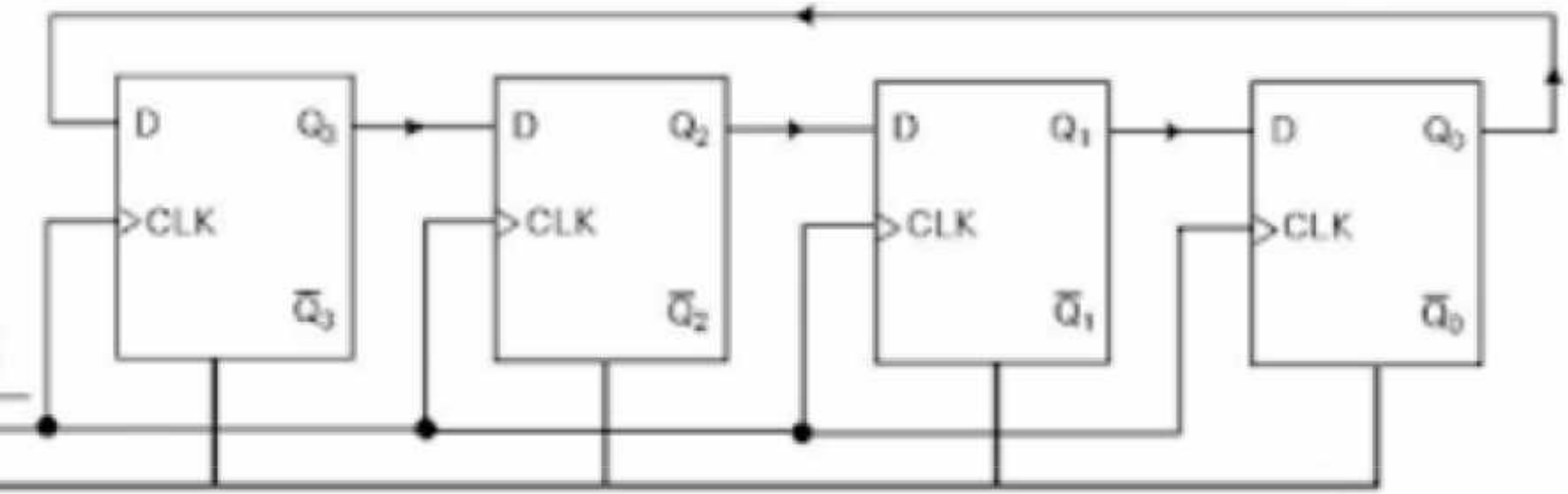
States for 4 bit John counter

on counter



Timing diagram for Johnson counter

Ring Counter Using FFs



Ring Counter

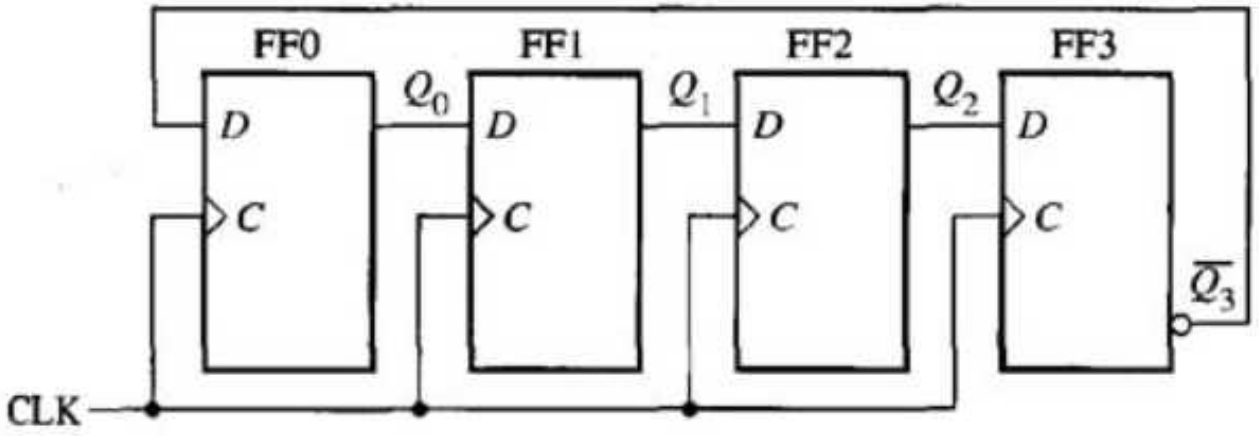
Q_0	Q_1	Q_2
1	0	0
0	1	0
0	0	1
0	0	0

Code for 4-bit Ring Counter Using FFs

```
y ieee;
eee.std_logic_1164.all;
y ring_counter is
clk : in std_logic;
in std_logic;
ut std_logic_vector(3 downto 0));
ing_counter;
tecture behavioural of ring_counter is
l temp : std_logic_vector(3 downto 0);

ess (clk,rst)
n
```

```
if (rst = '1') then
    temp <= "1000";
  elsif (rising_edge(clk)) then
temp <= temp(0) & temp(3 dow
  end if;
 end process;
 Q <= temp;
end behavioural;
```

on Counter Using FFs



(a)

Clock	Q_0	Q_1	Q_2	Q_3
0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1

(b)

Code for 4-bit Johnson Counter Using FFs

```
y ieee;
eee.std_logic_1164.all;
y ring_counter is
clk : in std_logic;
in std_logic;
out std_logic_vector(3 downto 0));
ing_counter;
tecture behavioural of ring_counter is
l temp : std_logic_vector(3 downto 0);

ess (clk,rst)
n
```

```
if (rst = '1') then
    temp <= “0000";
  elsif (rising_edge(clk))
temp(1) <= temp(0);
temp(2) <= temp(1);
temp(3) <= temp(2);
temp(0) <= not temp(3);
end if;
 end process;
 Q <= temp;
end behavioural;
```

r Feedback Shift Register (LFSR) Counters

n bit LFSR counter can have 2^n-1 states, almost maximum. Such a counter is of

ed a maximum length sequence generator .

also called as pseudo random binary sequence generator

e term **random** means that the outputs do not cycle through a normal binary co

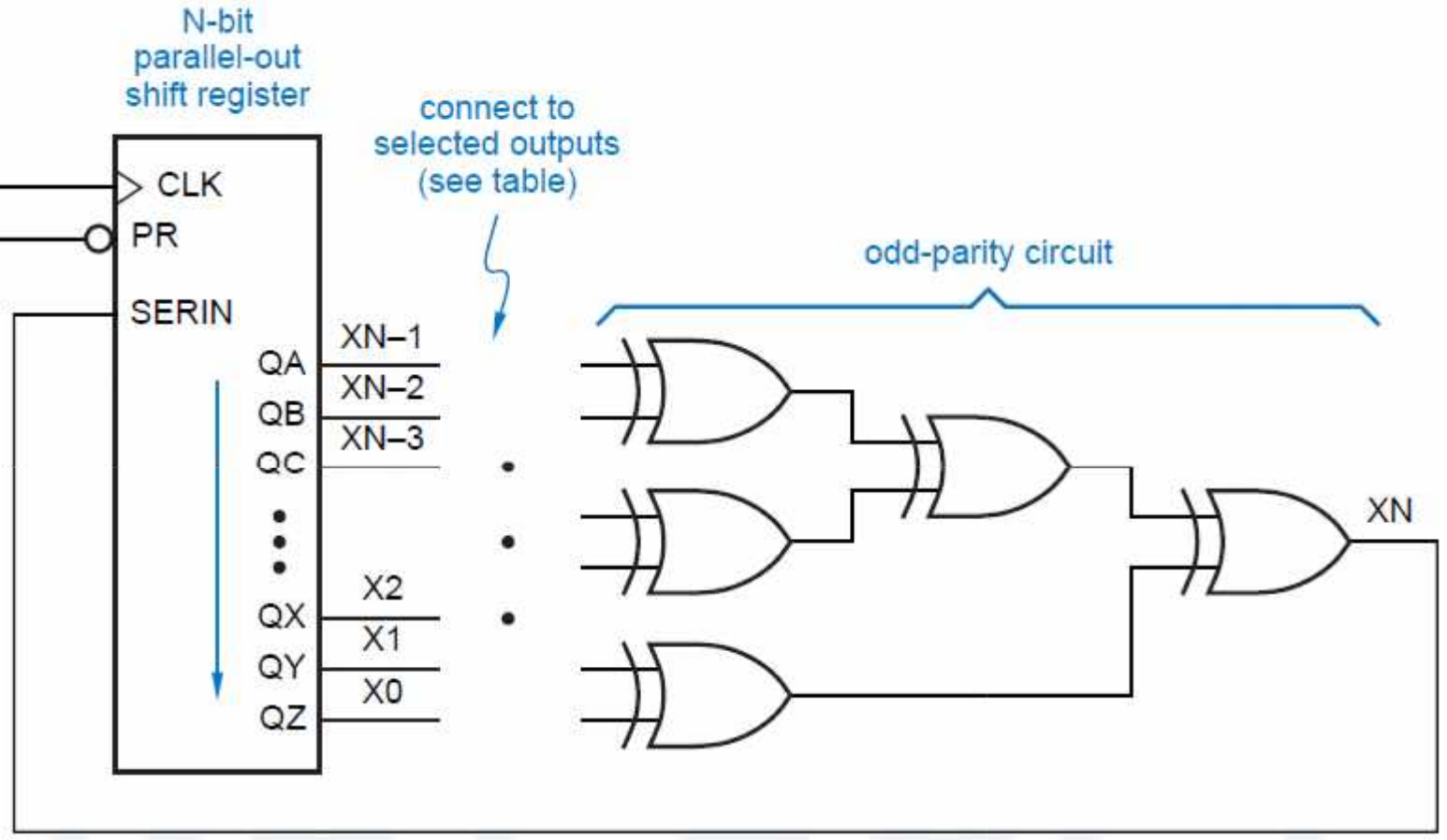
uence.

e term **pseudo** here refers to the fact that the sequence is not truly random becau

loes cycle through all possible combinations once every 2^n-1 clock cycles where

resent the number of shift register stages

r Feedback Shift Register (LFSR) Counters

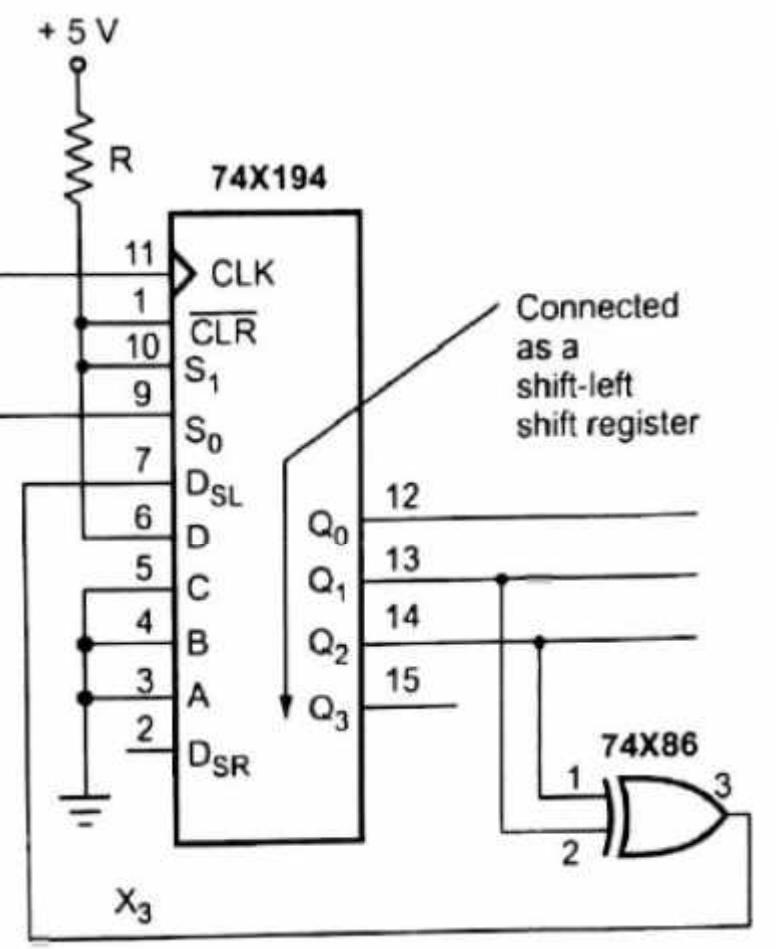


n	Feedback Equatio
2	X2 = X1 ⊕ X0
3	X3 = X1 ⊕ X0
4	X4 = X1 ⊕ X0
5	X5 = X2 ⊕ X0
6	X6 = X1 ⊕ X0
7	X7 = X3 ⊕ X0
8	X8 = X4 ⊕ X3 ⊕ X2 ⊕
12	X12 = X6 ⊕ X4 ⊕ X1 ⊕
16	X16 = X5 ⊕ X4 ⊕ X3 ⊕
20	X20 = X3 ⊕ X0
24	X24 = X7 ⊕ X2 ⊕ X1 ⊕
28	X28 = X3 ⊕ X0
32	X32 = X22 ⊕ X2 ⊕ X1

General Structure of LFSR counter

Feed back equa

LFSR Counter



3-bit LFSR counter

Q_2	Q_1	Q_0
0	0	1
0	1	0
1	0	1
0	1	1
1	1	1
1	1	0
1	0	0
0	0	1

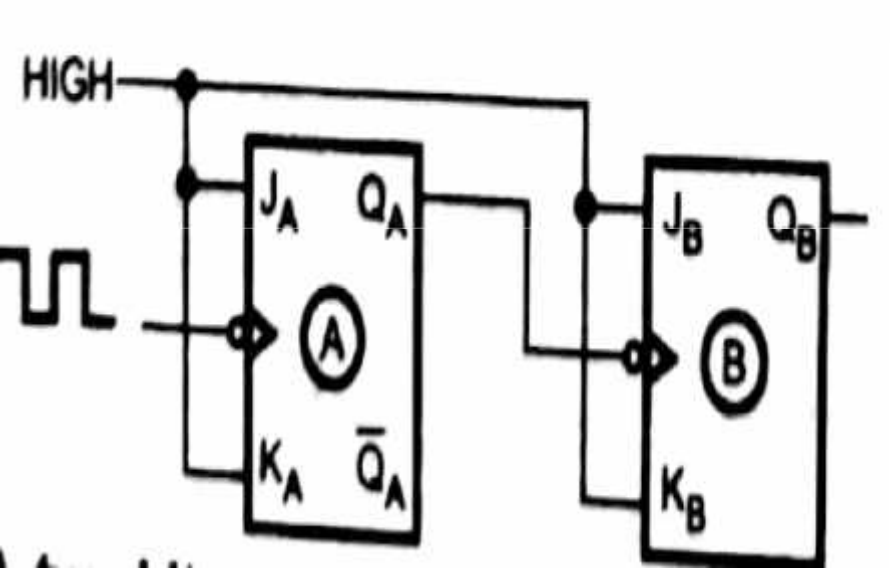
3-bit LFSR state sequence

COUNTERS

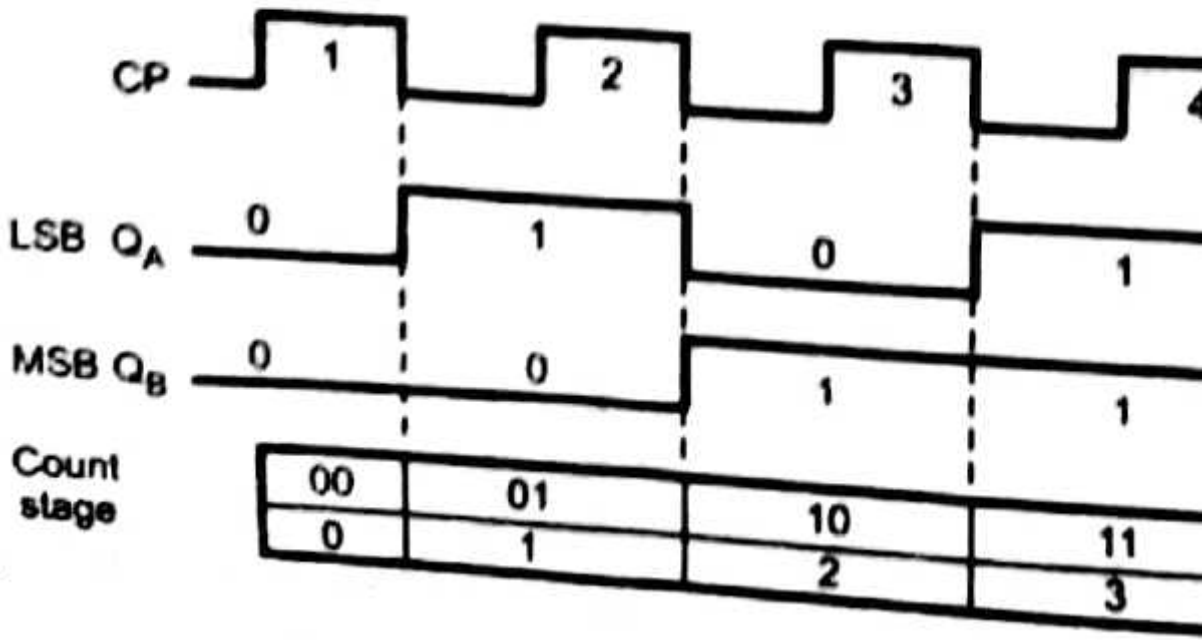
ynchronous counter

nchronous counter

nchronous up counter



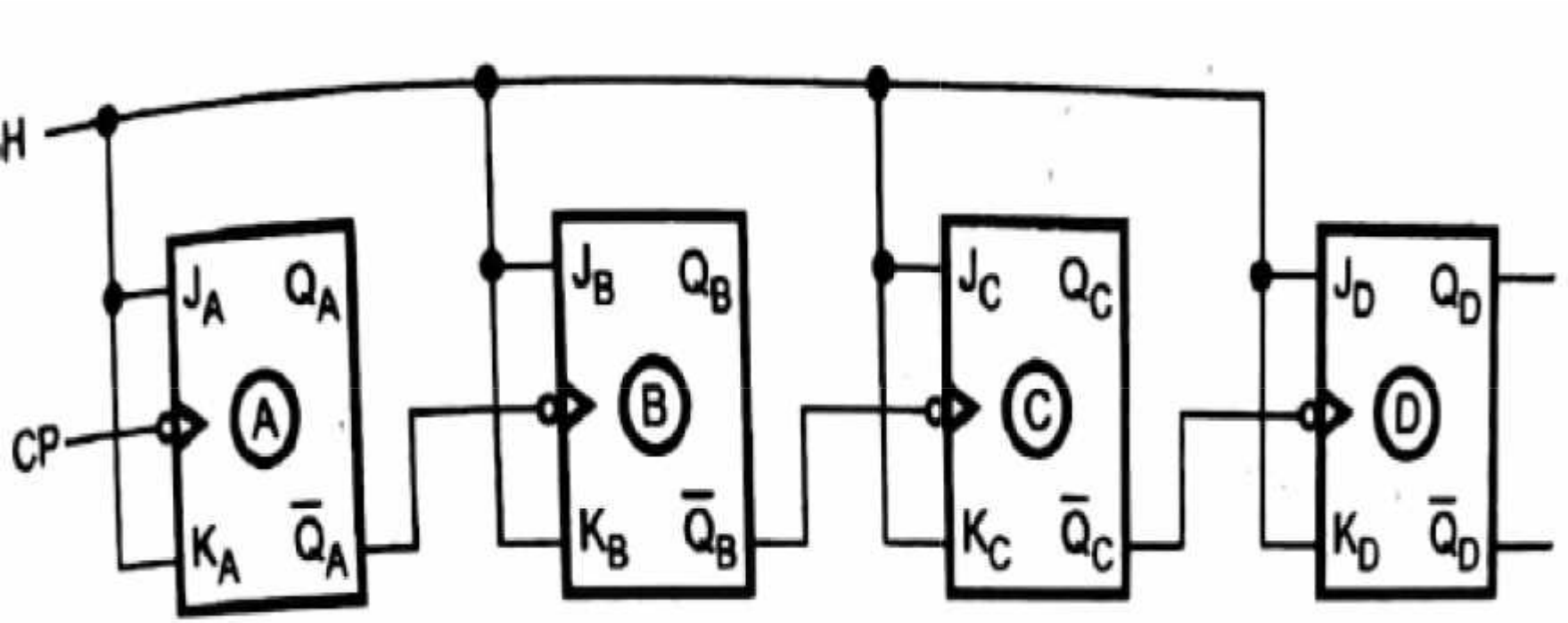
2 bit asynchronous up counter



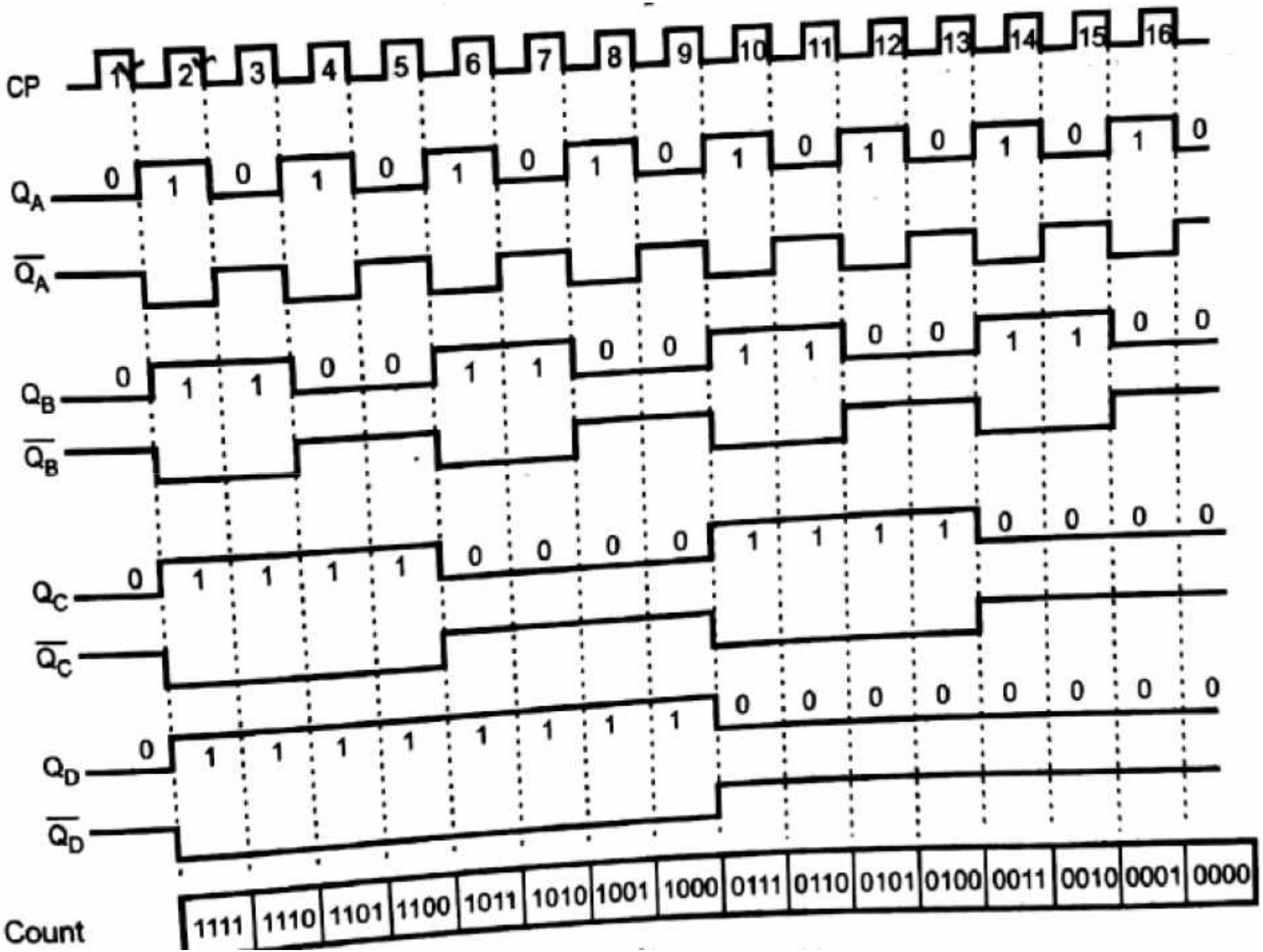
Timing diagram

COUNTERS

chronous/ripple down counter



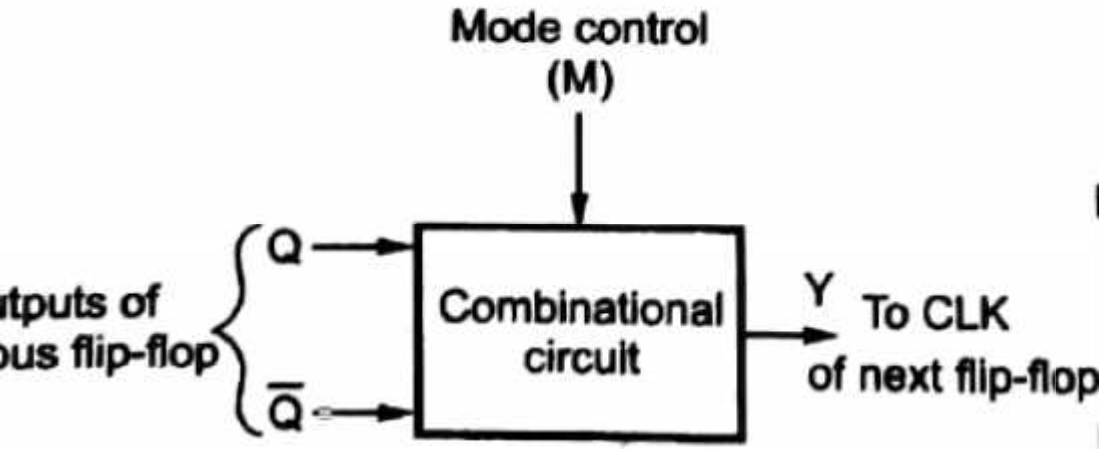
4 bit asynchronous down counter


CP
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Q_A
$\overline{Q_A}$
Q_B
$\overline{Q_B}$
Q_C
$\overline{Q_C}$
Q_D
$\overline{Q_D}$
Count
1111 1110 1101 1100 1011 1010 1001 1000 0111 0110 0101 0100 0011 0010 0001 0000

Timing

COUNTERS

hronous up/ down counter



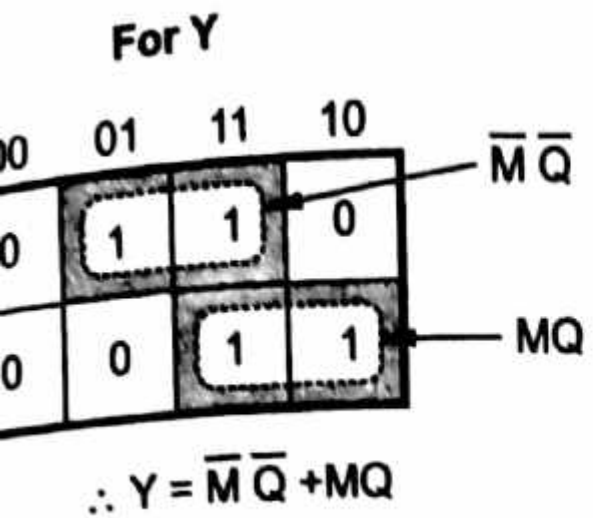
(a) The block diagram of combinational circuit

	Inputs M	Q	$\overline{Q}$	Output Y	
M = 0	0	0	0	0	Y = $\overline{Q}$ for dow counti
	0	0	1	1	
	0	1	0	0	
	0	1	1	1	
M = 1	1	0	0	0	Y = Q for up counti
	1	0	1	0	
	1	1	0	1	
	1	1	1	1	

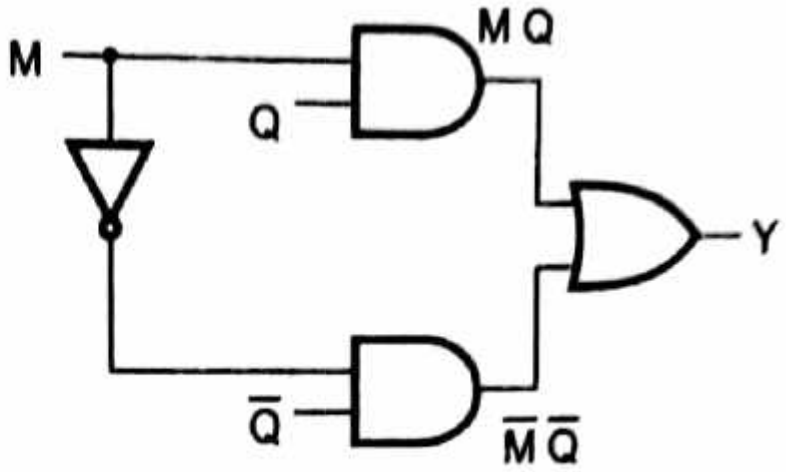
(b) Truth table

hronous up/ down counter

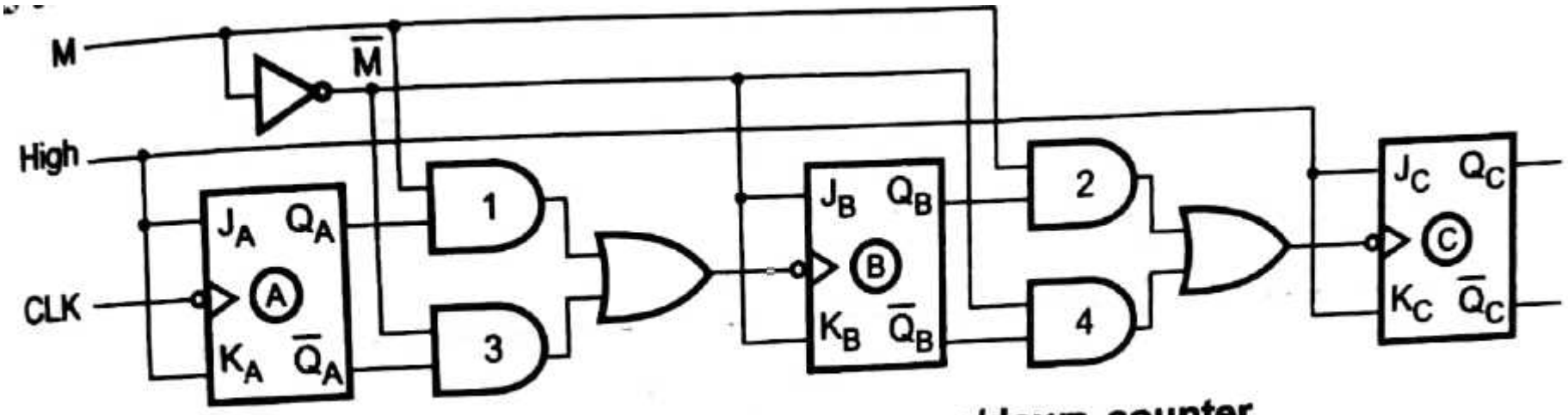
COUNTE



a) K-map simplification

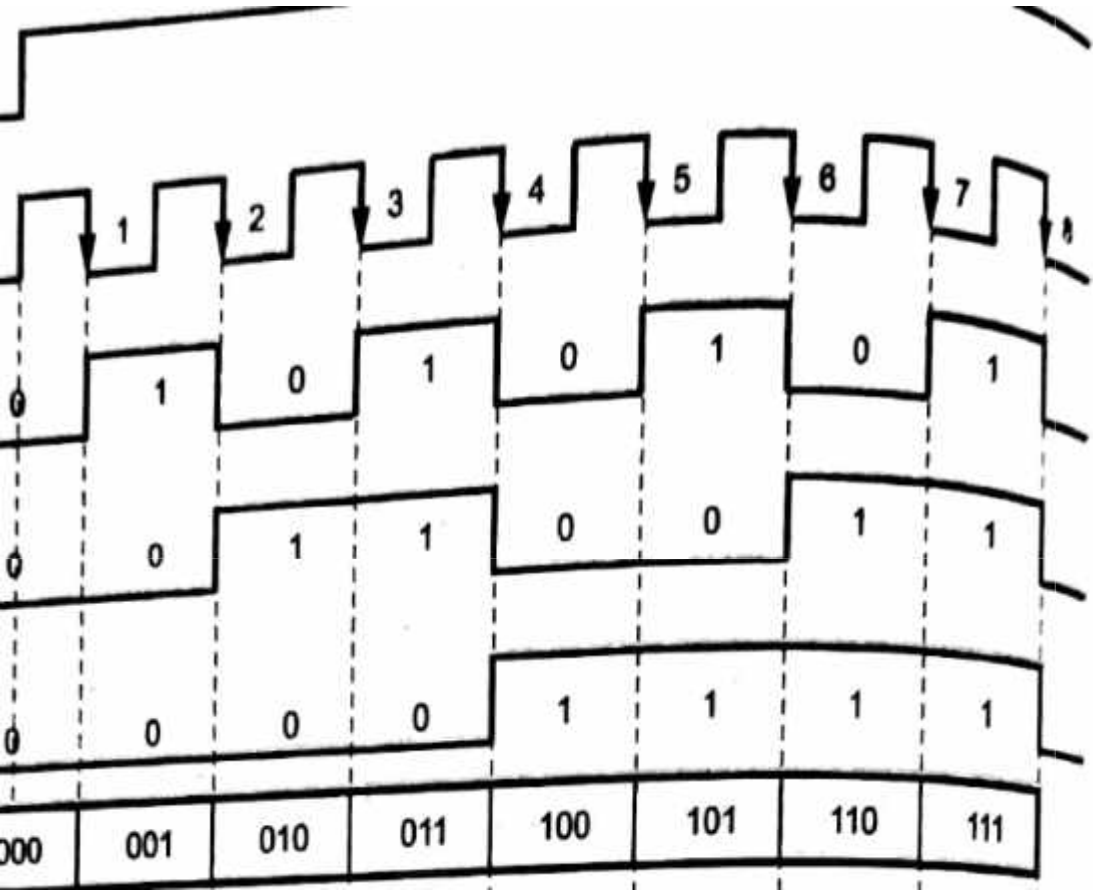


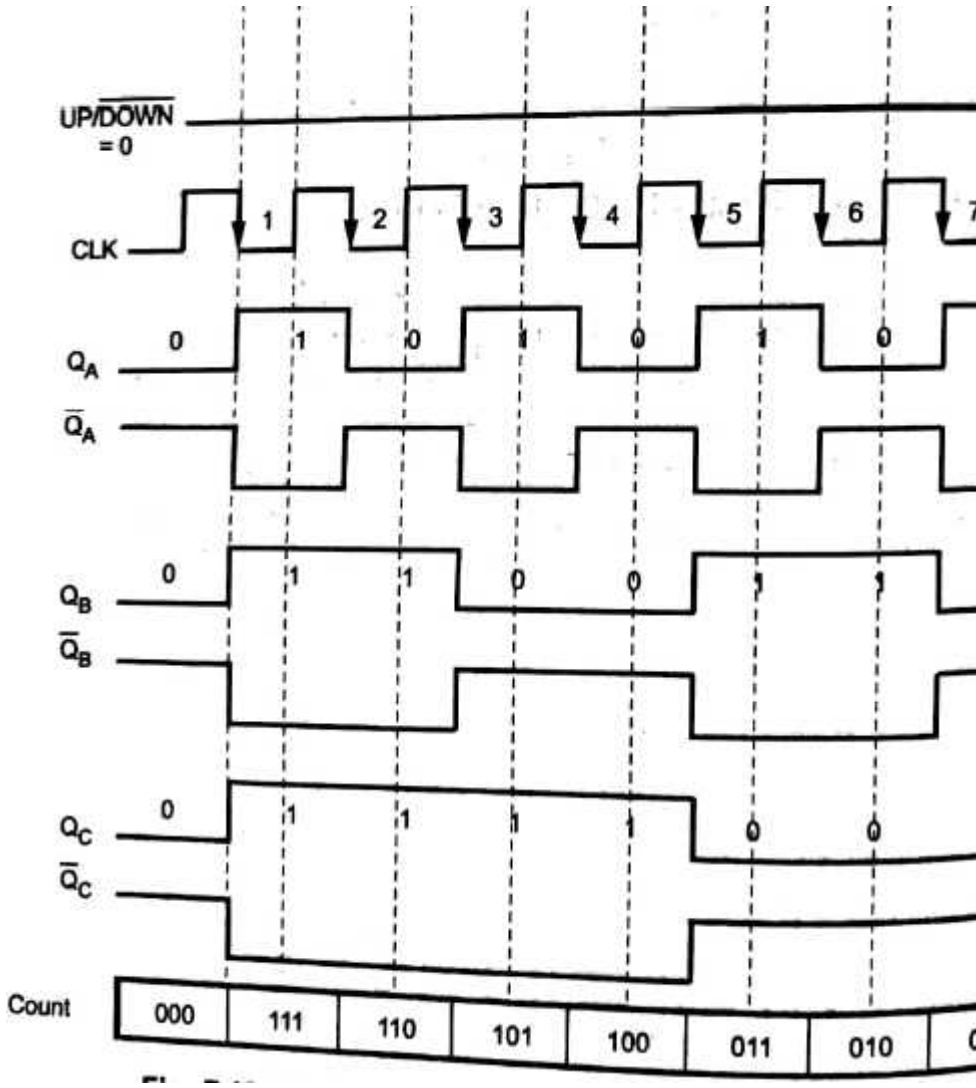
b) Logic diagram



3-bit asynchronous up/down counter

COUNTE

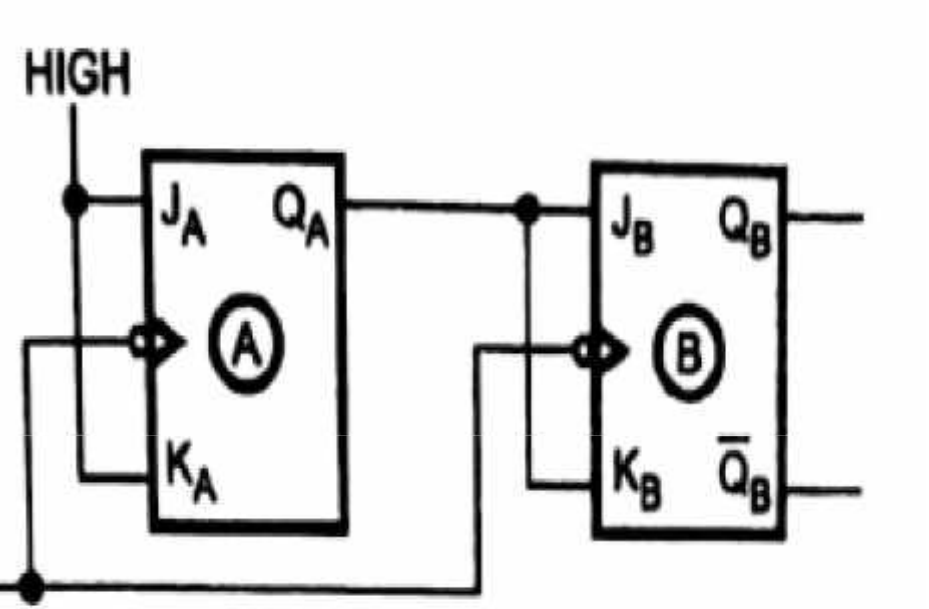




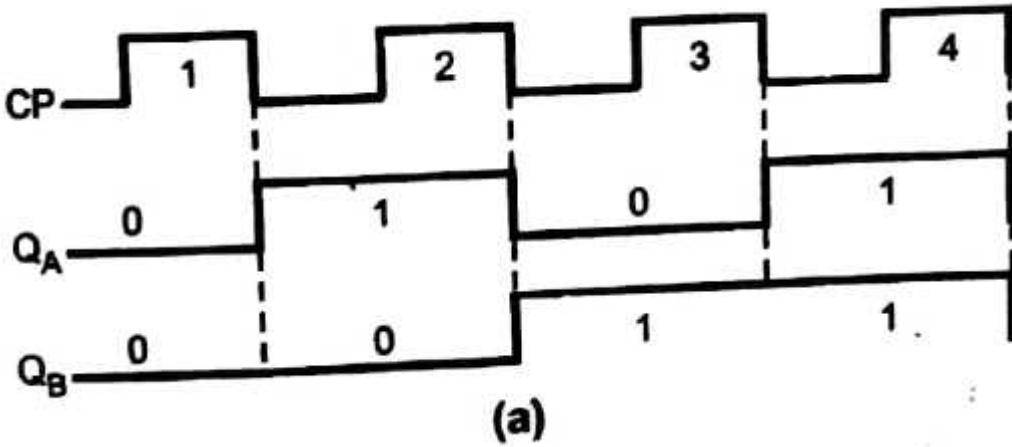
Timing Diagram for up/down counter

COUNTERS

ronous Counter



2 bit synchronous up counter



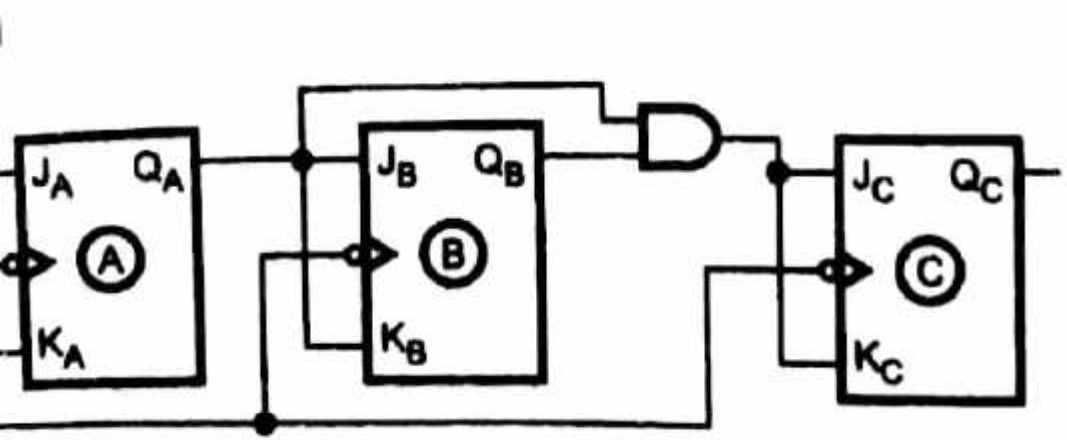
(a)

CP	Q_B	Q_A
0	0	0
1	0	1
2	1	0
3	1	1

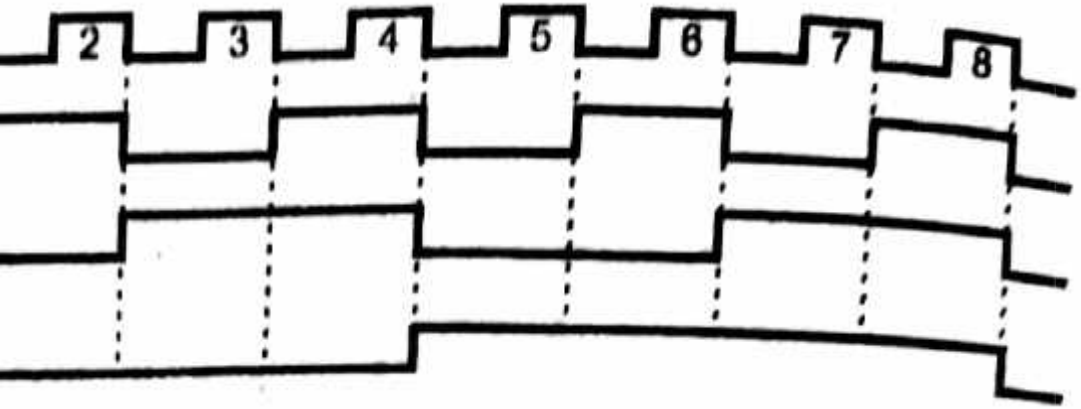
(b)

COUNTERS

ynchronous Counter



synchronous up counter



3 bit synchronous up counter timing diagram

CP	Q_C	Q_B	
0	0	0	
1	0	0	
2	0	1	
3	0	1	
4	1	0	
5	1	0	
6	1	1	
7	1	1	

3 bit synchronous up counter state sequ

COUNTERS

ronous Up/Down Counter

n

n=0

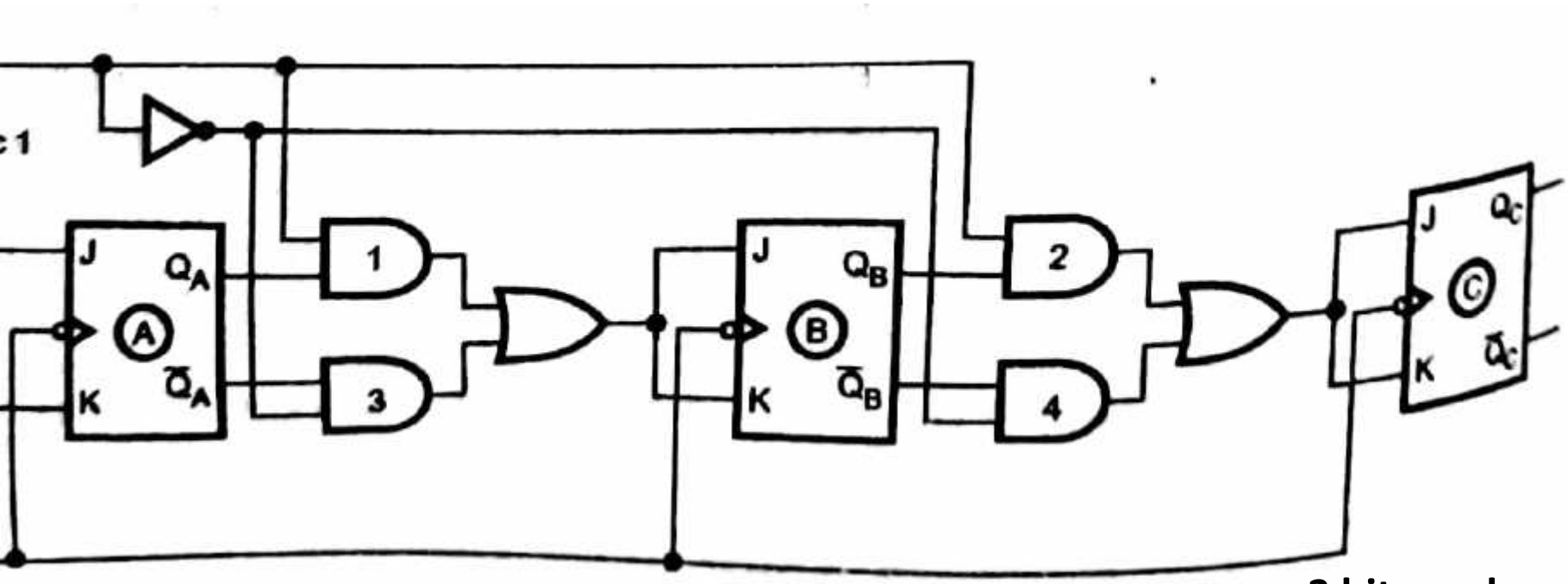
es 1 &2 are disabled and 3 & 4 are enabled

s connected as input to the other FF, it will work as down counter

n=1

es 3 &4 are disabled and 1 & 2 are enabled

onnected as input to the other FF, it will work as up counter



3 bit synchronous up/down

COUNTERS

Sr. No.	Asynchronous counters	Synchronous counters
1.	In this type of counter flip-flops are connected in such a way that output of first flip-flop drives the clock for the next flip-flop.	In this type there is no connection between output of first flip-flop and clock input of the next flip-flop.
2.	All the flip-flops are not clocked simultaneously.	All the flip-flops are clocked simultaneously.
3.	Logic circuit is very simple even for more number of states.	Design involves complex logic circuit as number of states increases.
4.	Main drawback of these counters is their low speed as the clock is propagated through number of flip-flops before it reaches last flip-flop.	As clock is simultaneously given to all flip-flops there is no problem of propagation delay. Hence they are high speed counters and are preferred when number of flip-flops increases in the given design.

Code for 4 bit Up counter

COUN

```
RY IEEE;
EEE.std_logic_1164.all;
EEE.std_logic_unsigned.all;
Y upctr IS
T(Clock,Resetn,EN  : IN  STD_LOGIC;
                Q  : OUT STD_LOGIC_VECTOR (3 DOWNTO 0));

pctr;
TECTURE Behavior OF upctr IS
SIGNAL Count : STD_LOGIC_VECTOR(3 DOWNTO 0);
N
OCESS(Clock, Resetn)
GIN
IF   Resetn = '0' THEN
     Count <= "0000";
ELSIF(Clock'EVENT AND Clock = '1') THEN
   IF EN = '1' THEN
                Count<=Count+1;
            ELSE
                Count<=Count;
            END IF;
        END IF;
    END PROCESS;
     Q<=Count;
    END Behavior;
```

VHDL Code for 4 bit Up counter using asynchro clear input

```
ary ieee;
ieee.std_logic_1164.all;
ieee.std_logic_unsigned.all;
ty counter is
rt(C, CLR, UP_DOWN : in std_logic;
    Q : out std_logic_vector(3 downto 0));
counter;
itecture archi of counter is
gnal tmp: std_logic_vector(3 downto 0);
gin
process (C, CLR)
  begin
    if (CLR='1') then
      tmp <= "0000";
    elsif (C'event and C='1') then
      if (UP_DOWN='1') then
        tmp <= tmp + 1;
      else
        tmp <= tmp - 1;
      end if;
    end if;
end process;
Q <= tmp;
archi;
```

Modulus N Synchronous counter

ynchronous counter

required are

$2^n \geq N$

$N = 5$

$n = 3$

ble

JK FF Excitation Tab

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

sent State		Next State			Flip-flop Inputs					
Q_B	Q_A	Q_{c+1}	Q_{B+1}	Q_{A+1}	J_C	K_C	J_B	K_B	J_A	K_A
0	0	0	0	1	0	X	0	X	1	X
0	1	0	1	0	0	X	1	X	X	1
1	0	0	1	1	0	X	X	0	1	X
1	1	1	0	0	1	X	X	1	X	1
0	0	0	0	0	X	1	0	X	0	X
0	1	X	X	X	X	X	X	X	X	X
1	0	X	X	X	X	X	X	X	X	X
1	1	X	X	X	X	X	X	X	X	X

Synchronous counter

For J_C

	01	11	10
0	1	0	
X	X	X	

$J_C = Q_B Q_A'$

For K_C

Q_C \ Q_BQ_A	00	01	11	10
0	X	X	X	X
1	1	X	X	X

$K_C = 1$

For J_A

Q_C \ Q_BQ_A	00	01	11	10
0	1	X	X	1
1	0	X	X	X

$J_A = \bar{Q}_C$

For K_A

Q_C \ Q_BQ_A	00	01	11
0	X	1	1
1	X	X	X

$K_A = 1$

For J_B

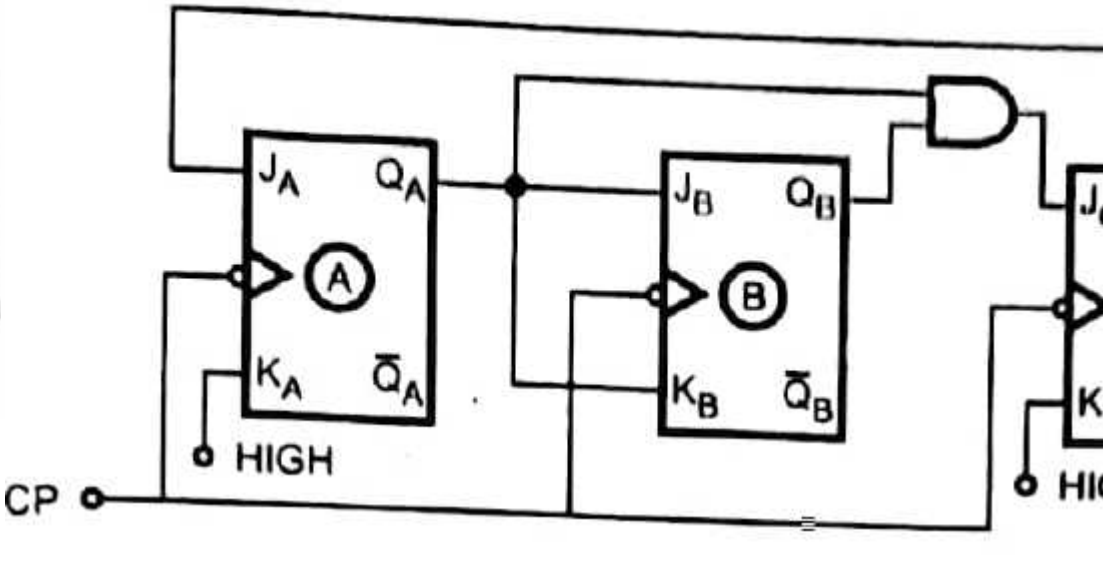
	01	11	10
	1	X	X
	X	X	X

$J_B = Q_A$

For K_B

Q_C \ Q_BQ_A	00	01	11	10
0	X	X	1	0
1	X	X	X	X

$K_B = Q_A$



Mod-5 Synchronous Counter

Synchronou
Decade cou

Present State			Next State				Flip-flop Inputs			
Q_C	Q_B	Q_A	Q_{D+1}	Q_{C+1}	Q_{B+1}	Q_{A+1}	T_D	T_C	T_B	T_A
0	0	0	0	0	0	1	0	0	0	1
0	0	1	0	0	1	0	0	0	1	1
0	1	0	0	0	1	1	0	0	0	1
0	1	1	0	1	0	0	0	1	1	1
1	0	0	0	1	0	1	0	0	0	1
1	0	1	0	1	1	0	0	0	1	1
1	1	0	0	1	1	1	0	0	0	1
1	1	1	1	0	0	0	1	1	1	1
0	0	0	1	0	0	1	0	0	0	1
0	0	1	0	0	0	0	1	0	0	1
0	1	0	X	X	X	X	X	X	X	X
0	1	1	X	X	X	X	X	X	X	X
1	0	0	X	X	X	X	X	X	X	X
1	0	1	X	X	X	X	X	X	X	X
1	1	0	X	X	X	X	X	X	X	X
1	1	1	X	X	X	X	X	X	X	X

Mod-10 Synch
Counter Sate t

Synchronou
Decade cou

For T_B

$Q_D Q_C$ \ $Q_B Q_A$	01	11	10
00	1	1	0
01	1	1	0
11	X	X	X
10	0	X	X

$_B = Q_A \overline{Q}_D$

For T_A

$Q_D Q_C$ \ $Q_B Q_A$	00	01	11	10
00	1	1	1	1
01	1	1	1	1
11	X	X	X	X
10	1	1	X	X

$T_A = 1$

For T_D

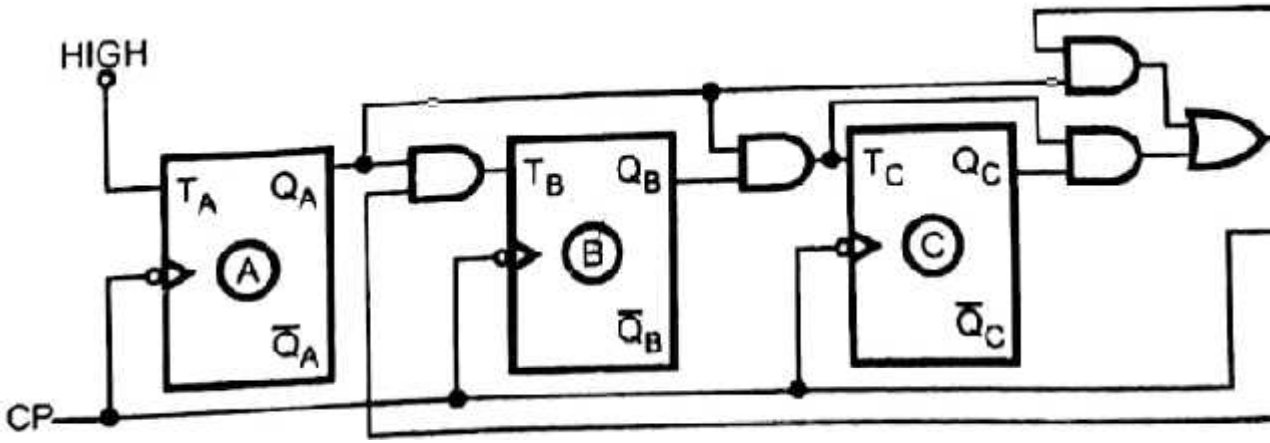
$Q_D Q_C$ \ $Q_B Q_A$	01	11	10
00	0	0	0
01	0	1	0
11	X	X	X
10	1	X	X

$Q_D + Q_A Q_B Q_C$

For T_C

$Q_D Q_C$ \ $Q_B Q_A$	00	01	11	10
00	0	0	1	0
01	0	0	1	0
11	X	X	X	X
10	0	0	X	X

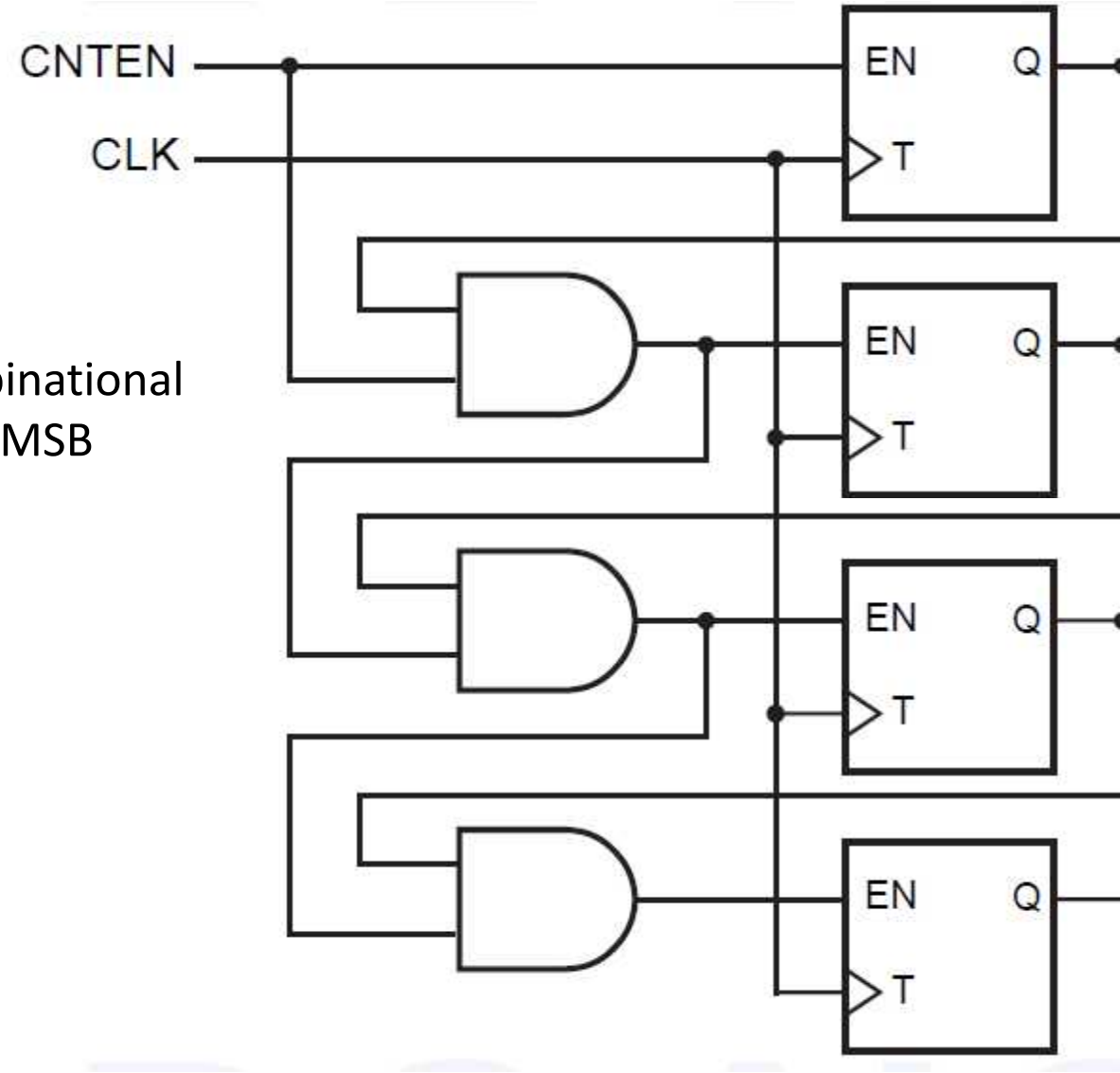
$T_C = Q_A Q_B$



Decade counter logic diagram

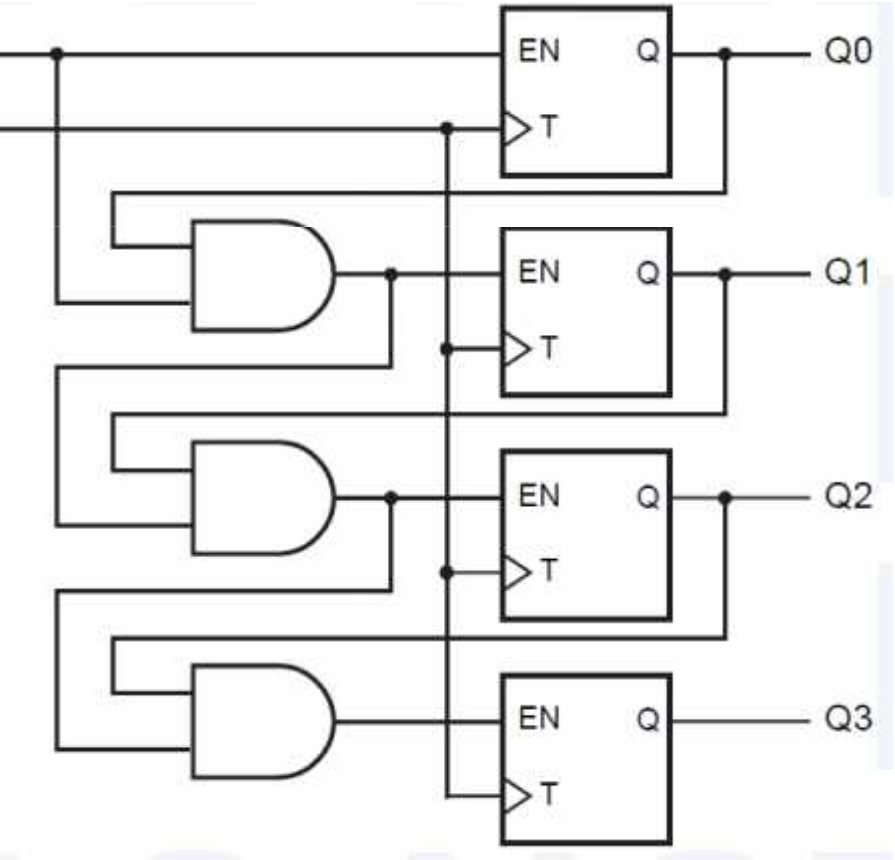
chronous 4-bit binary counter serial enable logic

l is the master count enable signal
FFs toggles if and only if CNTEN is high and
he lower order counter bits are 1.
nchronous serial counter because the combinational
e signals propagate serially from the LSB to MSB

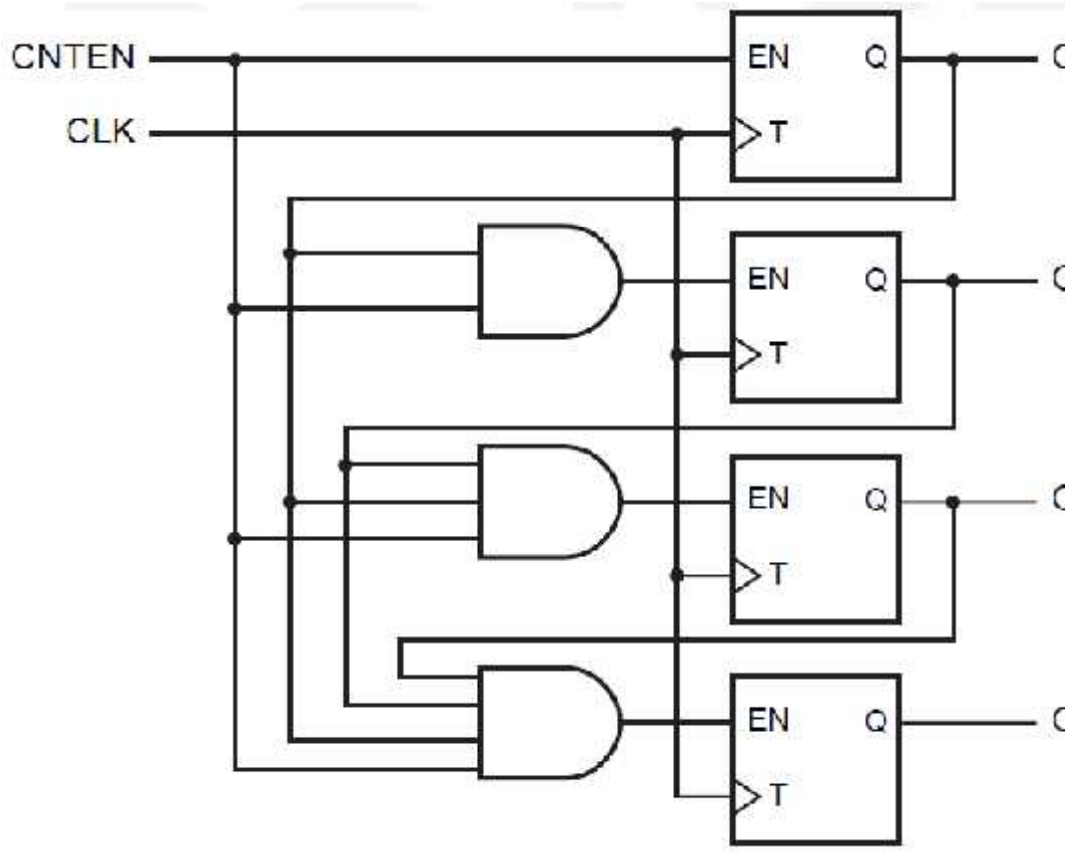


chronous 4-bit binary counter serial/Parallel enable logic

l is the master count enable signal
FFs toggles if and only if CNTEN is high and
he lower order counter bits are 1.
nchronous serial counter because the combinational
e signals propagate serially from the LSB to MSB



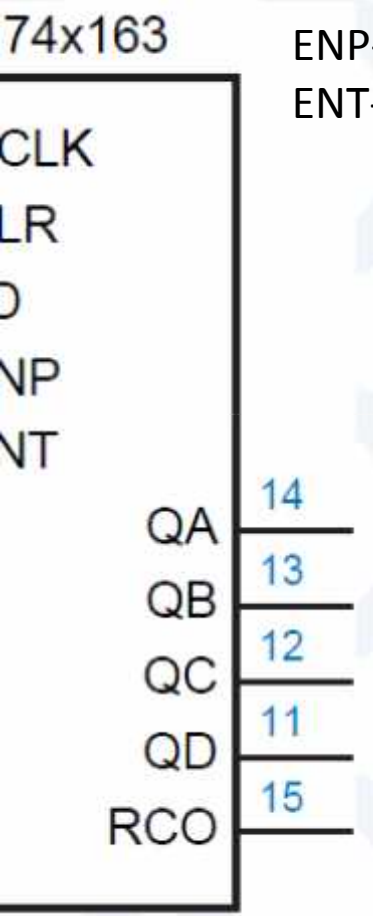
nary synchronous counter with serial enable signal



4 bit binary synchronous counter with parallel en

OUNTERS

4-bit binary counter

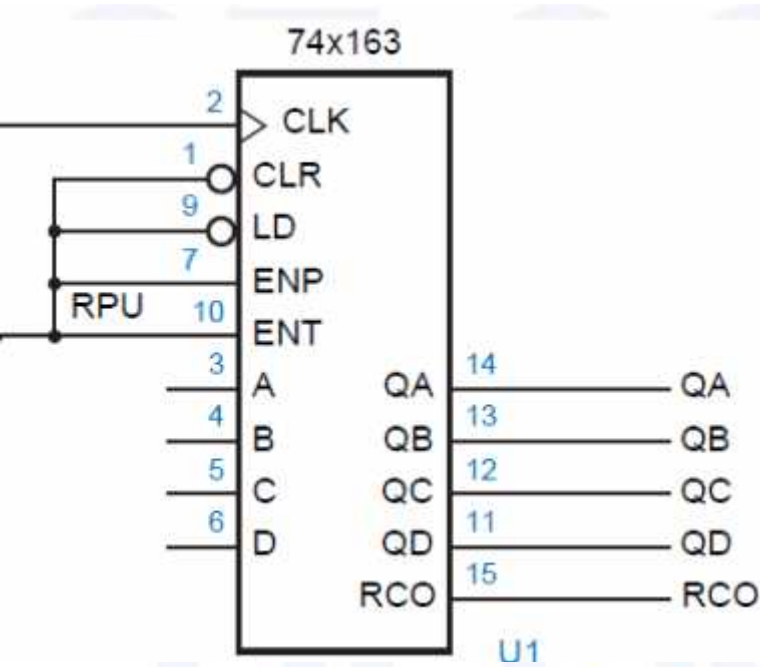


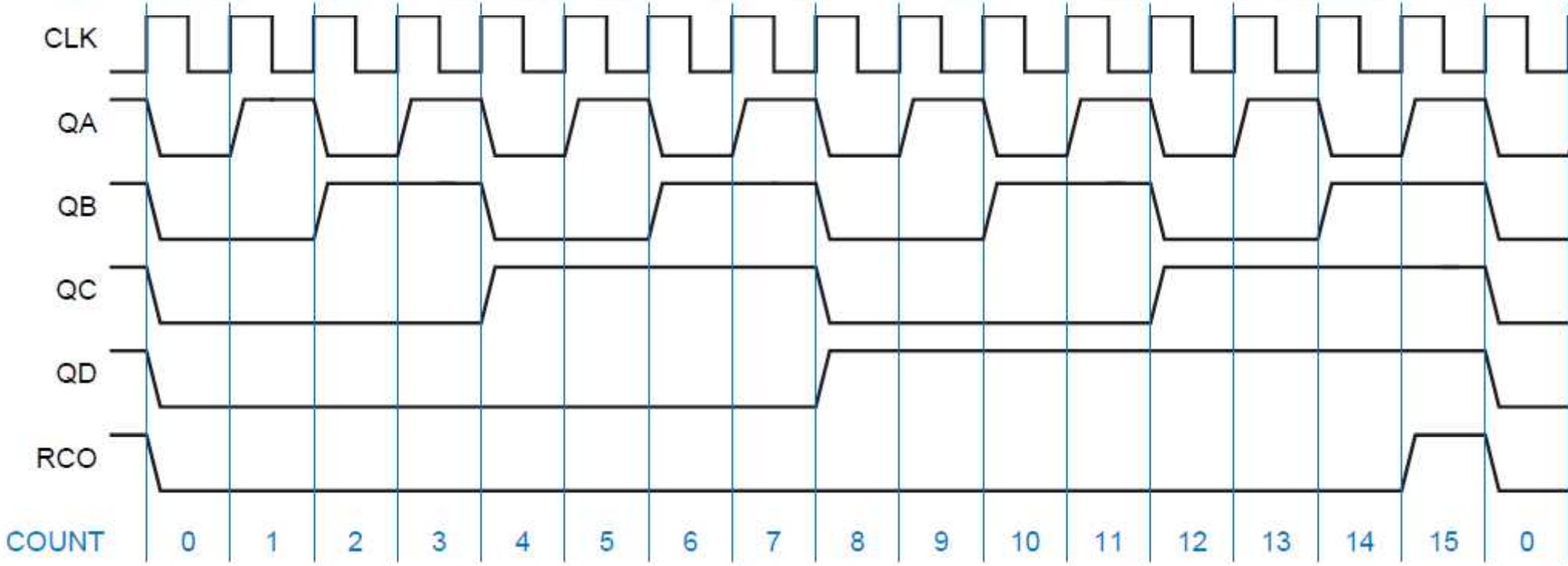
ENP-Count enable parallel input
ENT-ENP-Count enable trickle input

Inputs				Current State				Next Sta		
CLR_L	LD_L	ENT	ENP	QD	QC	QB	QA	QD*	QC*	Q
0	x	x	x	x	x	x	x	0	0	
1	0	x	x	x	x	x	x	D	C	
1	1	0	x	x	x	x	x	QD	QC	C
1	1	x	0	x	x	x	x	QD	QC	C
1	1	1	1	0	0	0	0	0	0	
1	1	1	1	0	0	0	1	0	0	
1	1	1	1	0	0	1	0	0	0	
1	1	1	1	0	0	1	1	0	1	
1	1	1	1	0	1	0	0	0	1	
1	1	1	1	0	1	0	1	0	1	
1	1	1	1	0	1	1	0	0	1	
1	1	1	1	0	1	1	1	1	0	
1	1	1	1	1	0	0	0	1	0	
1	1	1	1	1	0	0	1	1	0	
1	1	1	1	1	0	1	0	1	0	
1	1	1	1	1	0	1	1	1	1	
1	1	1	1	1	1	0	0	1	1	
1	1	1	1	1	1	0	1	1	1	
1	1	1	1	1	1	1	0	1	1	
1	1	1	1	1	1	1	1	0	0	

State table for 74x163 4-bit binary counter

Free running mode for the 74x163 4 bit cou





chronous Sequential Circuits

synchronous or clocked sequential circuits, clocked flip-flops are used as mem
nts, which change their individual states in synchronism with the periodic c
. Therefore, the change in states of flip-flops and change in state of the entire ci
s at the transition of the clock signal.

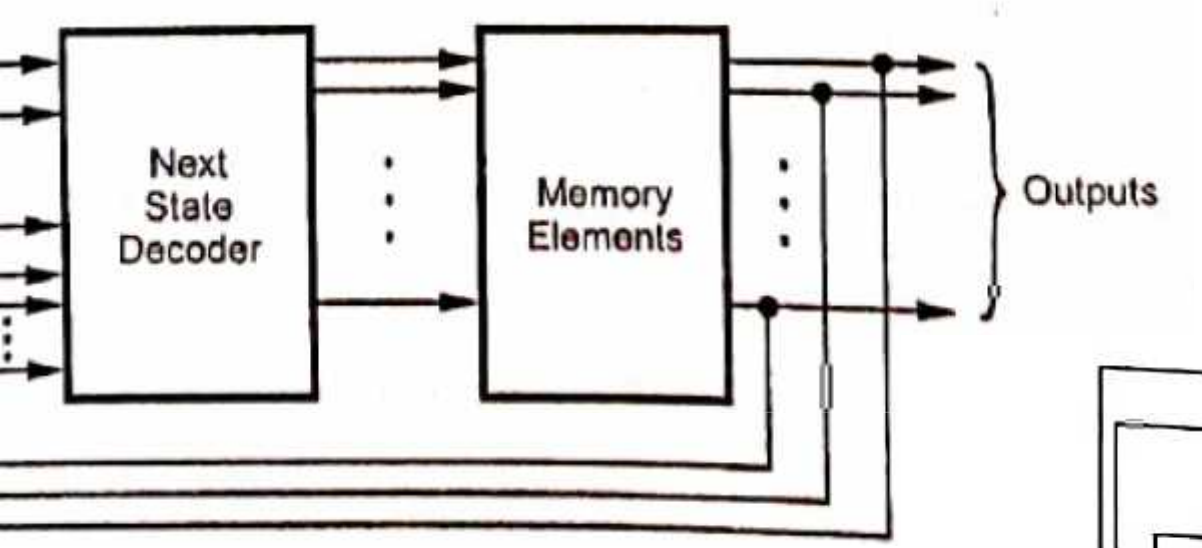
he synchronous or clocked sequential circuits are represented by two models.

- **Moore circuit :** The output depends only on the present state of the flip-flops
- **Mealy circuit :** The output depends on both the present state of the flip-flop and on the input(s).

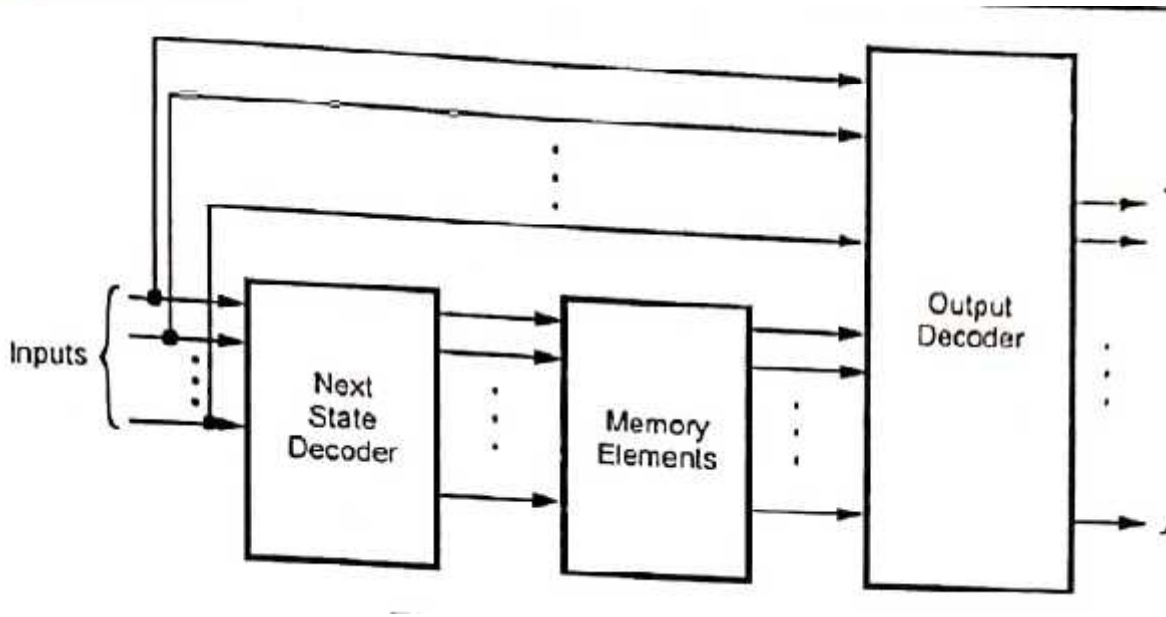
equential circuits are also called **finite state machines (FSMs)**, which is more fo
e that is often found in technical literature. The name derives from the fact that
ional behaviour of these circuits can be represented using a finite number of states

chronous Sequential Circuits

e Circuit Model



Mealy Circuit Model



chronous Sequential Circuits

Moore Vs Mealy Circuit Models

Sr. No.	Moore Circuit	Mealy Circuit
1.	Its output is a function of present state only.	Its output is a function of present state as well as present input.
2.	Input changes does not affect the output.	Input changes may affect the output of the circuit.
3.	Moore circuit requires more number of states for implementing same function.	It requires less number of states for implementing same function.

UNIT-V

gn Examples (using VHDL): Barrel shifter, comparators, floating-

der, and dual parity encoder.

uential logic Design: Latches & flip flops, PLDs, counters, shift reg

heir VHDL models.

BARREL SHIFTER

arrel shifter is a combinational logic circuit with **n data inputs**, **n data outputs** and a

control inputs that specify how to shift the data between input and output.

arrel shifter that is part of a microprocessor CPU can typically specify the directio

t (left or right), the typed of shift (circular, arithmetic, logical), and the amount of

pically **0 to n-1** bits or **0 to n bits**

bit barrel shifter that does **left circular shifts only**, using a **4-bit control input S[3:0**

cify the amount of shift.

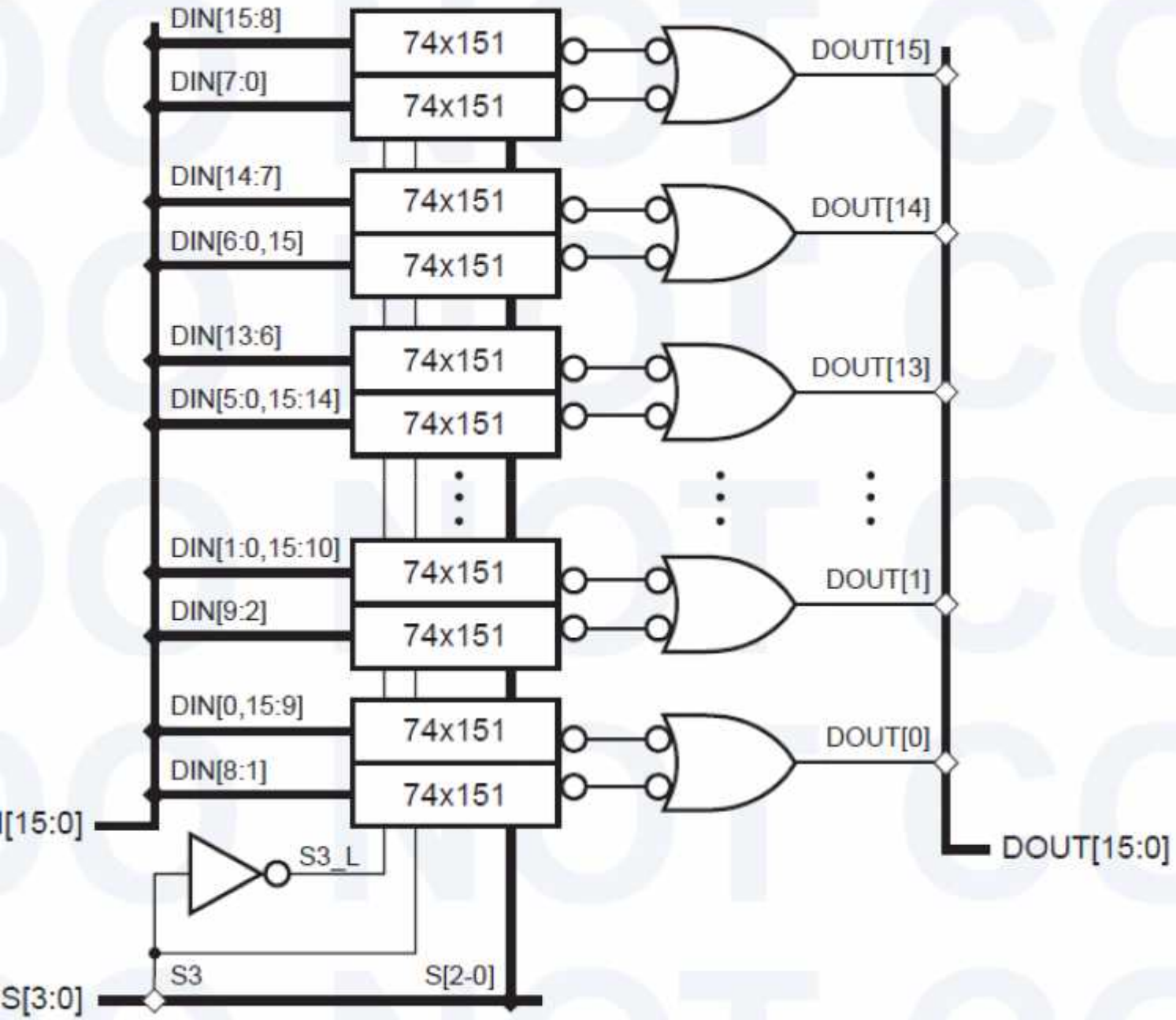
example, if the input word is **ABCDEFGHGIHKLMNOP** (where each letter represents

, and the control input is **0101 (5)**, then the output word is **FGHGIHKLMNOPABCDE**.

16-input, one-bit multiplexer can be built using two 74x151s, by applying S3 and

mplement to the EN_L inputs and combining the Y_L data outputs with a NAND gate,

BARREL SHIFTER



The top ’151 of each p
enabled by S3_L, and the botto
by S3; the remaining select b
connected to all 32 ’151s. Data
D0–D7 of each ’151 are connec
the DIN inputs in the listed orde
left to right.

16-bit barre

BARREL SHIF

ode for 16-bit barrel shifter for left circular shift

```
ry IEEE;
EEE.std_logic_1164.all;

y rol16 is
ort (
   DIN: in STD_LOGIC_VECTOR(15 downto 0);  -- Data inputs
   S: in STD_LOGIC_VECTOR (3 downto 0);    -- Shift amount, 0-15
   DOUT: out STD_LOGIC_VECTOR(15 downto 0) -- Data bus output
;
ol16;

tecture rol16_arch of rol16 is

ss(DIN, S)
iable X, Y, Z: STD_LOGIC_VECTOR(15 downto 0);
in
f S(0)='1' then X := DIN(14 downto 0) & DIN(15); else X := DIN; end if;
f S(1)='1' then Y := X(13 downto 0) & X(15 downto 14); else Y := X; end if;
f S(2)='1' then Z := Y(11 downto 0) & Y(15 downto 12); else Z := Y; end if;
f S(3)='1' then DOUT <= Z(7 downto 0) & Z(15 downto 8); else DOUT <= Z; end if;
 process;
ol16_arch;
```

cading Comparators

74x682 8-bit comparator, it doesn't have cascading inputs and outputs at all.

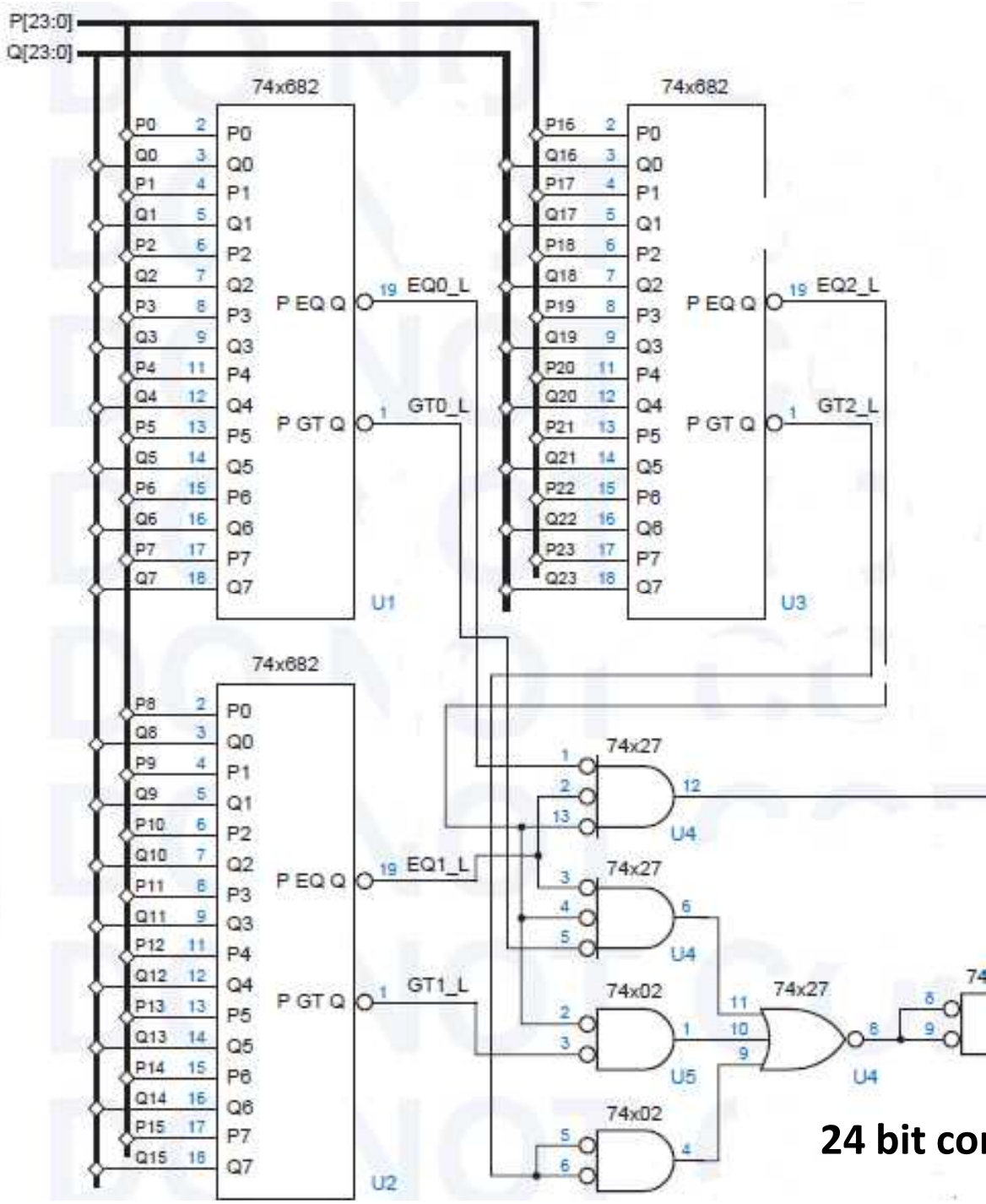
, at first glance, you might think that it be used to build larger comparators. But s not true.

e 74x682 8-bit comparators to do lity and greater-than comparison on two it operands.

24-bit results are derived from the idual 8-bit results using combinational for the following equations:

$$Q = EQ2 \cdot EQ1 \cdot EQ0$$

$$Q = GT2 + EQ2 \cdot GT1 + EQ2 \cdot EQ1 \cdot GT0$$



al VHDL Code for 64 bit Comparator

```
 IEEE;
E.std_logic_1164.all;
E.std_logic_unsigned.all;

comp64 is
t ( A, B: in STD_LOGIC_VECTOR (63 downto 0);
    EQ, GT: out STD_LOGIC );
p64;

cture comp64_arch of comp64 is

 '1' when A = B else '0';
 '1' when A > B else '0';
p64_arch;
```

Behavioral VHDL Code for 8 bit Comparator

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

entity comp8 is
    port ( A, B: in STD_LOGIC_VECTOR (7 downto
           EQ, GT: out STD_LOGIC );
end comp8;

architecture comp8_arch of comp8 is
begin
  EQ <= '1' when A = B else '0';
  GT <= '1' when A > B else '0';
end comp8_arch;
```

al VHDL Code for 32 bit mode dependent Comparator

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

entity Vmodecmp is
    port ( M: in STD_LOGIC_VECTOR (1 downto 0);      -- mode
           A, B: in STD_LOGIC_VECTOR (31 downto 0); -- unsigned integers
           EQ, GT: out STD_LOGIC );                  -- comparison results
end Vmodecmp;

architecture Vmodecmp_arch of Vmodecmp is
begin
  process (M, A, B)
  begin
    case M is
      when "00" =>
        if A = B then EQ <= '1'; else EQ <= '0'; end if;
        if A > B then GT <= '1'; else GT <= '0'; end if;
      when "01" =>
        if A(31 downto 1) = B(31 downto 1) then EQ <= '1'; else EQ <= '0';
        if A(31 downto 1) > B(31 downto 1) then GT <= '1'; else GT <= '0';
      when "10" =>
        if A(31 downto 2) = B(31 downto 2) then EQ <= '1'; else EQ <= '0';
        if A(31 downto 2) > B(31 downto 2) then GT <= '1'; else GT <= '0';
      when others => EQ <= '0';  GT <= '0';
    end case;
  end process;
end Vmodecmp_arch;
```

Floating Point Encoder

unsigned binary integer **B** in the range **0 < B < 2^{11}** can be represented by **11**
xed-point" format.
b10b9…..b1b0. It can represent numbers in the same range with less precision
ly **7 bits** in a floating-point notation.
***M* × 2^E** where M is a **4-bit mantissa m3m2m1m0** and **E is a 3-bit exponent e2e1e**
e smallest integer in this format is 0.2^0 and the largest is $(2^4-1).2^7$.
ven an 11-bit fixed-point integer B, we can convert it to our 7-bit floating point no
"picking off" four high-order bits beginning with the most significant 1, for examp

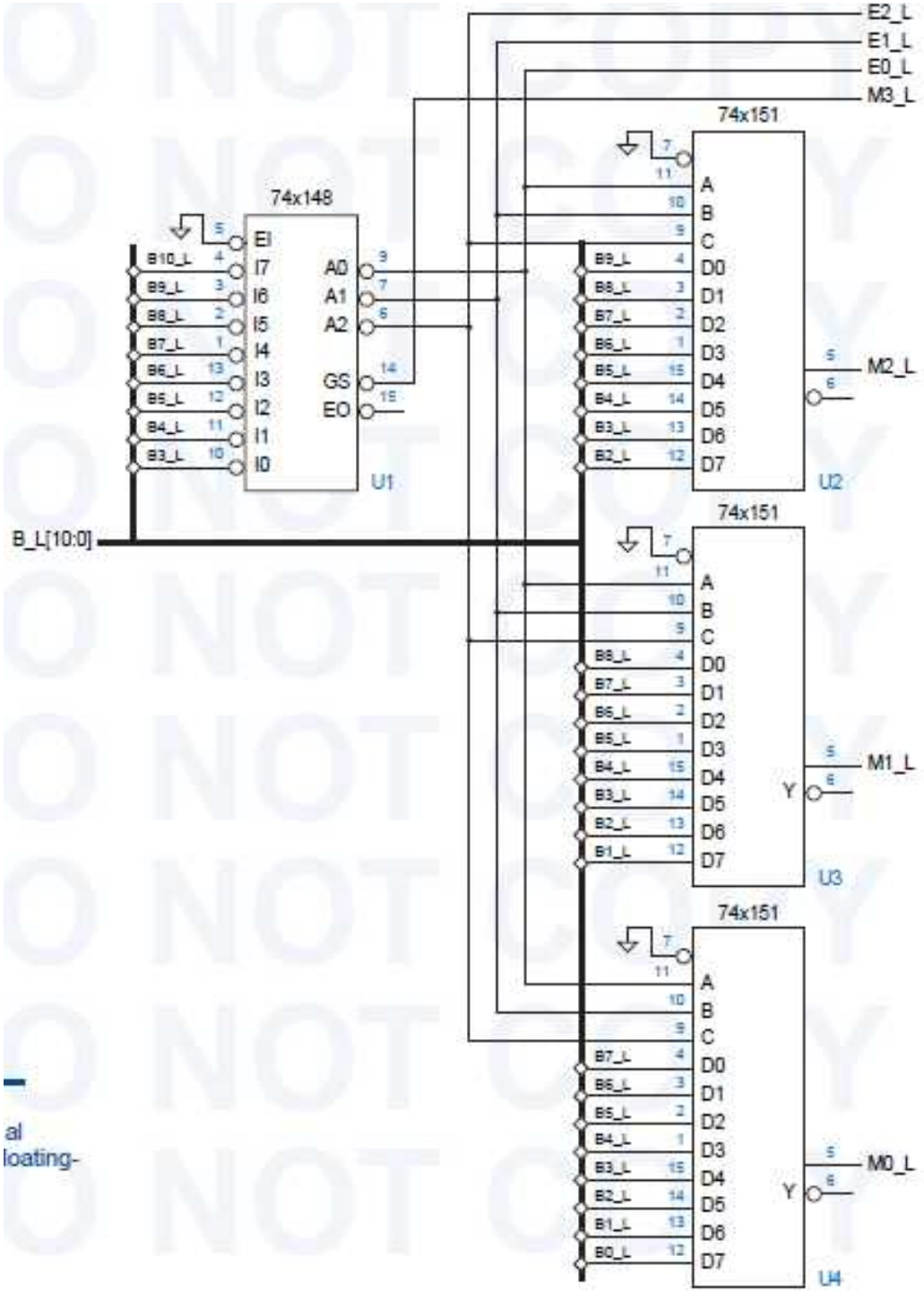
combinational circuit is to convert an 11-bit signed binary integer B into a 7-bit floating-nt number *M*, *E*, where *M* and *E* have 4 and bits, respectively. The numbers have the ationship $\boldsymbol{B = M \times 2^E + T}$,

where T is the truncation error, $0 \le T < 2^E$.

$$11010110100 = 1101 \cdot 2^7 + 01$$
$$00100101111 = 1001 \cdot 2^5 + 01$$
$$00000111110 = 1111 \cdot 2^2 + 10$$
$$00000001011 = 1011 \cdot 2^0 + 0$$
$$00000000010 = 0010 \cdot 2^0 + 0$$

B=1716-fixed Point
B=1664+52-floating

g Point Encoder



g Point Encoder

priority encoder, the 74x148, has active-low inputs, the input number B is umed to be available on an active-low bus B_L[10:0].

ou think about the conversion operation a while, you'll realize that the most ificant bit of the mantissa, m3, is always 1, except in the no-1- found case. The 8 has a GS_L output that indicates this case, allowing us to eliminate the tiplexer for m3.

'148 has active-low outputs, so the exponent bits (E0_L–E2_L) are produced in ve-low form. Naturally, three inverters could be used to produce an active-high sion.

ce everything else is active-low, active-low mantissa bits are used too. Active- n bits are also readily available on the '148 EO_L and the '151 Y_L outputs.

al VHDL Code for fixed point to floating point conversion

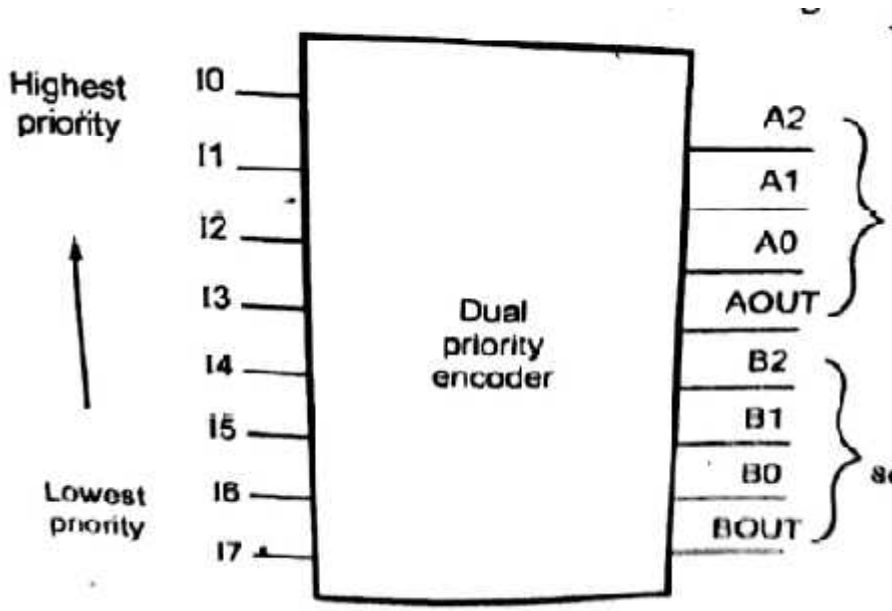
```
EEE;
std_logic_1164.all;
std_logic_arith.all;

enc is

 STD_LOGIC_VECTOR(10 downto 0); -- fixed-point number
t STD_LOGIC_VECTOR(3 downto 0); -- floating-point mantissa
t STD_LOGIC_VECTOR(2 downto 0)  -- floating-point exponent

;

ure fpenc_arch of fpenc is

(B)
e BU: UNSIGNED(10 downto 0);

 UNSIGNED(B);
 BU < 16   then M <= B( 3 downto 0); E <= "000";
 BU < 32   then M <= B( 4 downto 1); E <= "001";
 BU < 64   then M <= B( 5 downto 2); E <= "010";
 BU < 128  then M <= B( 6 downto 3); E <= "011";
 BU < 256  then M <= B( 7 downto 4); E <= "100";
 BU < 512  then M <= B( 8 downto 5); E <= "101";
 BU < 1024 then M <= B( 9 downto 6); E <= "110";
                M <= B(10 downto 7); E <= "111";
f;
cess;
_arch;
```

```
architecture fpence_arch of fpenc is
begin
  process(B)
  begin
    if   B(10) = '1' then M <= B(10 downto 7); E <
    elsif B(9) = '1' then M <= B( 9 downto 6); E <
    elsif B(8) = '1' then M <= B( 8 downto 5); E <
    elsif B(7) = '1' then M <= B( 7 downto 4); E <
    elsif B(6) = '1' then M <= B( 6 downto 3); E <
    elsif B(5) = '1' then M <= B( 5 downto 2); E <
    elsif B(4) = '1' then M <= B( 4 downto 1); E <
    else                  M <= B( 3 downto 0); E <
    end if;
  end process;
end fpence_arch;
```

DUAL PRIORITY ENCODER

iority encoder, if two or more inputs are equal to 1 at the same time, the input having the h
ity will take precedence.
se of dual-priority encoder, the circuit identifies both the highest-priority and the second-hi
ity asserted signal among a set of input signals

s set of 8 inputs I0 to I7, where I0 has the highest
ity
0, B2-B0, AOUT and BOUT are the output signals of
ncoder
0 and AOUT are used to identify the highest priority
t
T goes high only if highest priority input is present.
0 and BOUT are used to identify the second highest
ity input
T goes high only if second highest priority input is
ent.



Dual priority encoder

VHDL Code for the dual priority encoder using

```
Y IEEE;
E.std_logic_1164.all;
E.std_logic_arith.all;
TITY Dualprien IS
        PORT ( I   : IN STD_LOGIC_VECTOR (0 TO 7);
               A,B  :  OUT STD_LOGIC_VECTOR (2 DOWNTO 0);
      AOUT, BOUT   :  BUFFER STD_LOGIC);
D Dualprien;
CHITECTURE Dualprien_arch OF Dualpri IS
GIN
    PROCESS (I, AOUT, BOUT)
    BEGIN
        AOUT <= '0'; BOUT <='0';
        A<="000"; B<="000";
        FOR i IN 0 TO 7 LOOP
           If I(i) = '1' AND AOUT ='0' THEN
              A<= CONV_STD_LOGIC_VECTOR (i,3);
              AOUT <= '1';
           ELSEIF I(i) = '1' AND BOUT = '0' THEN
              B<= CONV_STD_LOGIC_VECTOR(i,3);
              BOUT<='1';
                                          ENDIF;
                                      END LOOP;
                                END PROCESS;
                                END Dualprien_arch;
```

VHDL Code for the dual priority encoder using

```
RY IEEE;
EEE.std_logic_1164.all;
EEE.std_logic_arith.all;
TY Dualprien IS
 PORT ( I     : IN STD_LOGIC_VECTOR (0 TO 7);
        A,B   : OUT STD_LOGIC_VECTOR (2 DOWNTO 0
LID,BVALID    : BUFFER STD_LOGIC);
Dualprien;
CTURE Dualprien_arch OF Dualpri IS
IN
  PROCESS (I, A, AOUT, BOUT)
  BEGIN
    IF      I(0)= '1' THEN A<= "000"; AOUT <='1';
    ELSEIF  I(1) = '1' THEN A<="001"; AOUT <= '1';
    ELSEIF  I(2) = '1' THEN A<="010"; AOUT <= '1';
    ELSEIF  I(3) = '1' THEN A<="011"; AOUT <= '1';
    ELSEIF  I(4) = '1' THEN A<="100"; AOUT <= '1';
    ELSEIF  I(5) = '1' THEN A<="101"; AOUT <= '1';
    ELSEIF  I(6) = '1' THEN A<="110"; AOUT <= '1';
    ELSEIF  I(7) = '1' THEN A<="111"; AOUT <= '1';
    ELSE
```

```
IF      I(1) = '1' AND A/="001" THEN B<="001"
                                          BOUT<='
ELSEIF  I(2) = '1' AND A/="010" THEN B<="010
                                          BOUT<
ELSEIF  I(3) = '1' AND A/="011" THEN B<="011
                                          BOUT<
ELSEIF  I(4) = '1' AND A/="100" THEN B<="100
                                          BOUT<
        ELSEIF  I(5) = '1' AND A/="101" THEN B<="1
                                          BO
        ELSEIF I(6)  = '1' AND A/="110" THEN B<="1
                                          BO
        ELSEIF I(7)  = '1' AND A/="111" THEN B<="1
                                          BO
        ELSE
                                          BO
        END IF;
    END PROCESS;
END Dualprien_arch;
```