Digital IC Applications using VHDL (AEC516)

Prepared By
Dr. Vijay Vallabhuni
Dr. K Nehru
Mr. D Khalandar Basha

UNIT – I

CMOS LOGIC, BIPOLAR LOGIC AND INTERFACING

Logic Families

- There are many, many ways to design an electronic logic circuit.
  - The first electrically controlled logic circuits, developed at Bell Laboratories in 1930s, were based on relays.
  - In the mid-1940s, the first electronic digital computer, the Eniac, used logic circuits based on vacuum tubes. The Eniac had about 18,000 tubes and a similar number of logic gates, not a lot by today's standards of microprocessor chips with tens of millions of transistors.
  - The inventions of the semiconductor diode and the bipolar junction transistor allowed the development of smaller, faster, and more capable computers in the late 1950s.
  - In the 1960s, the invention of the integrated circuit (IC) allowed multiple diodes, transistors, and other components to be fabricated on a single chip, and computers got still better.
- A logic family: is a collection of different integrated-circuit chips that have similar input, output, and internal circuit characteristics, but that perform different logic functions. Chips from the same family can be interconnected to perform any desired logic function.

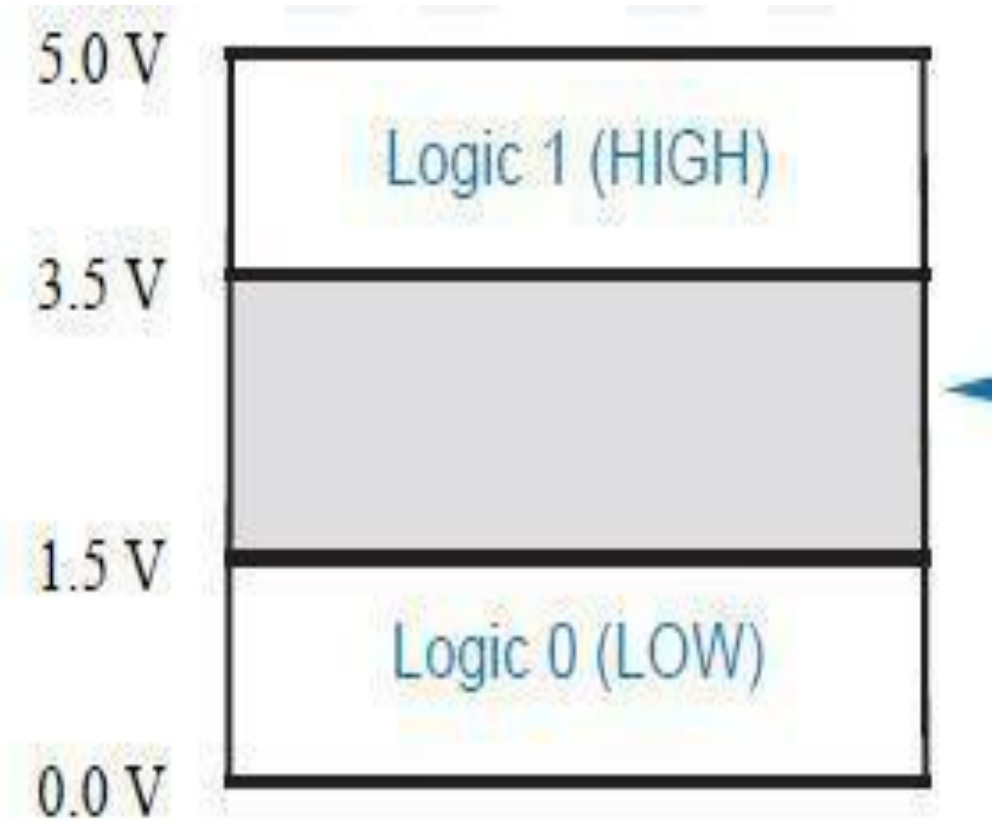
Logic Signals and Gates

- *Digital logic* hides the pitfalls of the analog world by mapping the infinite set of real values for a physical quantity into two subsets corresponding to just two possible numbers or *logic values*—0 and 1.
- A logic value, 0 or 1, is often called a *binary digit*, or *bit*. If an application requires more than two discrete values, additional bits may be used, with a set of *n* bits representing 2*n* different values. With most phenomena, there is an undefined region between the 0 and 1 states (e.g., voltage = 1.8 V, dim light, capacitor slightly charged, etc.). This undefined region is needed so that the 0 and 1 states can be unambiguously defined and reliably detected. Noise can more easily corrupt results if the boundaries separating the 0 and 1 states are too close.

CMOS Logic

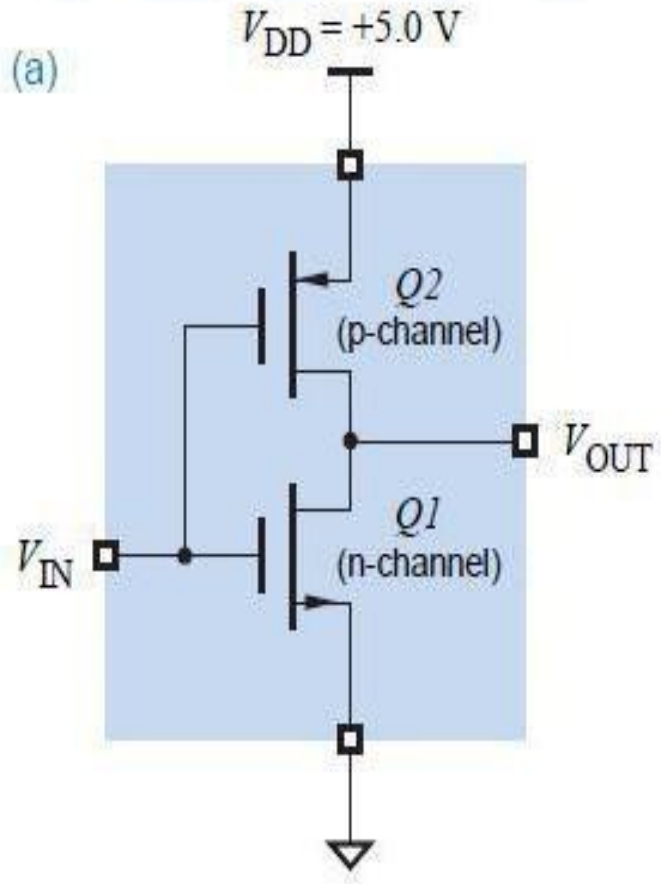
- The functional behavior of a CMOS logic circuit is fairly easy to understand, even if your knowledge of analog electronics is not particularly deep.
- The basic (and typically only) building blocks in CMOS logic circuits are MOS transistors, described shortly. Before introducing MOS transistors and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

CMOS Logic Levels



Basic CMOS Inverter Circuit

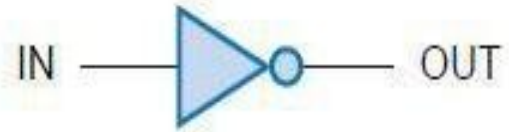
(a)



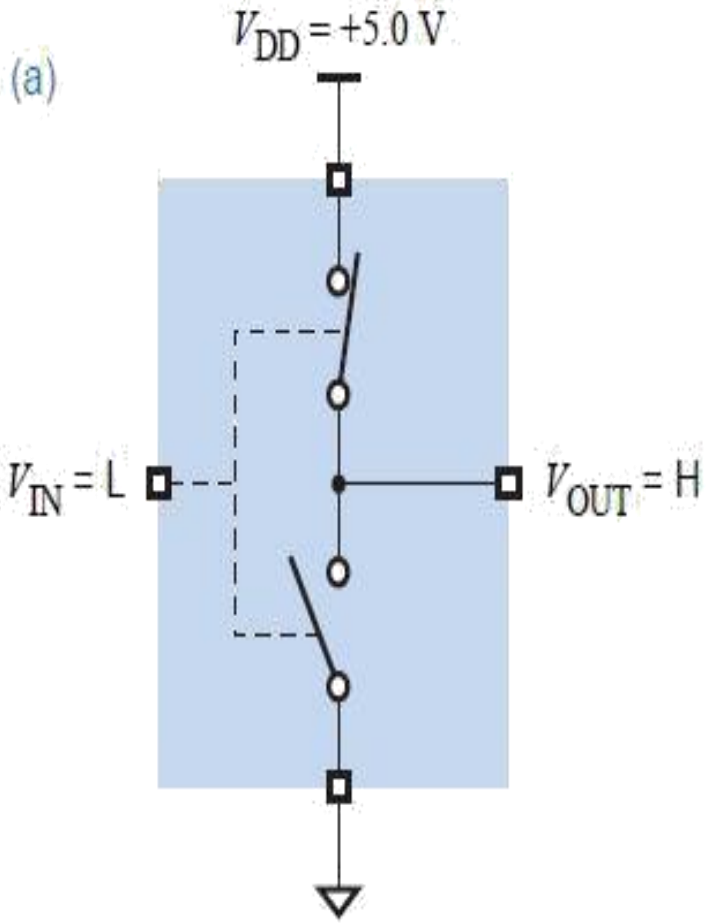
(b)

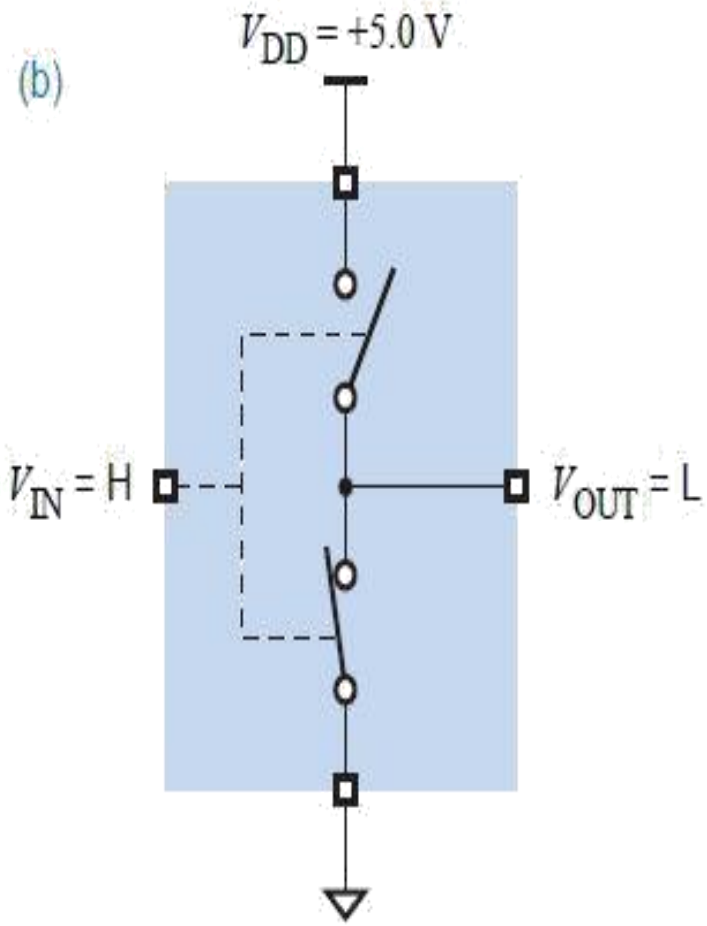
V_{IN}	Q1	Q2	V_{OUT}
0.0 (L)	off	on	5.0 (H)
5.0 (H)	on	off	0.0 (L)

(c)



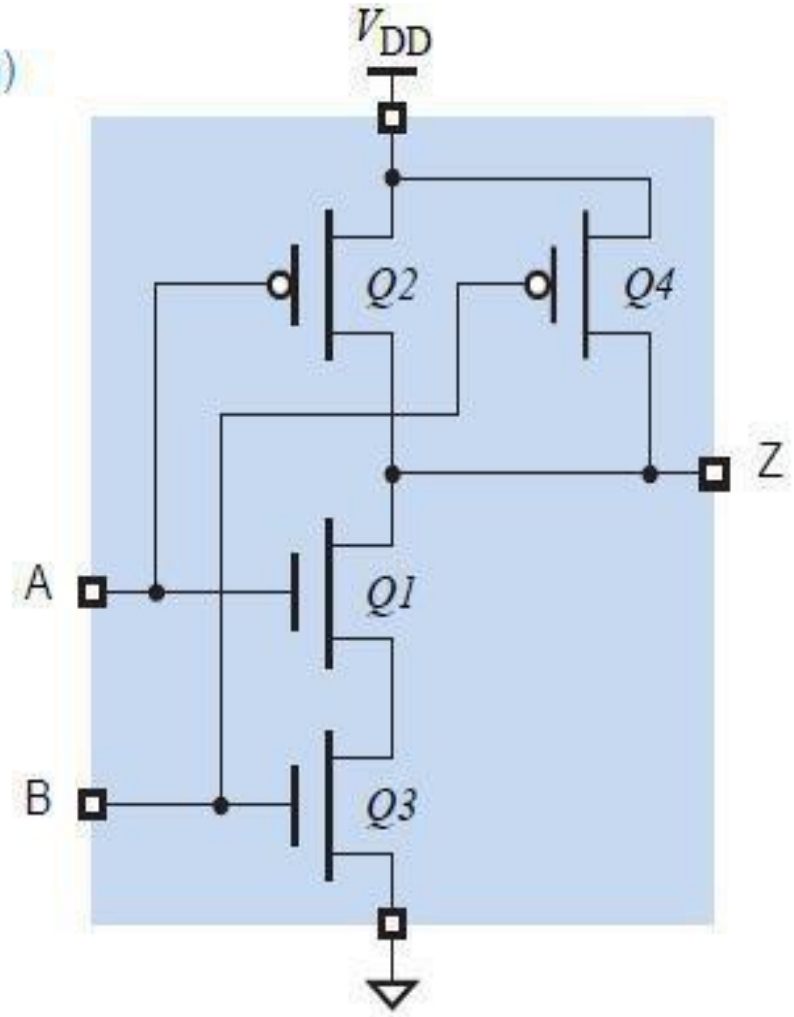
Basic CMOS Inverter Circuit





CMOS NAND Gate

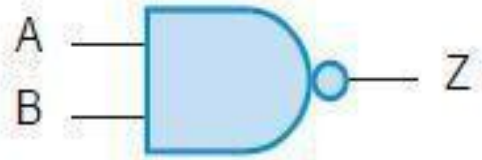
(a)



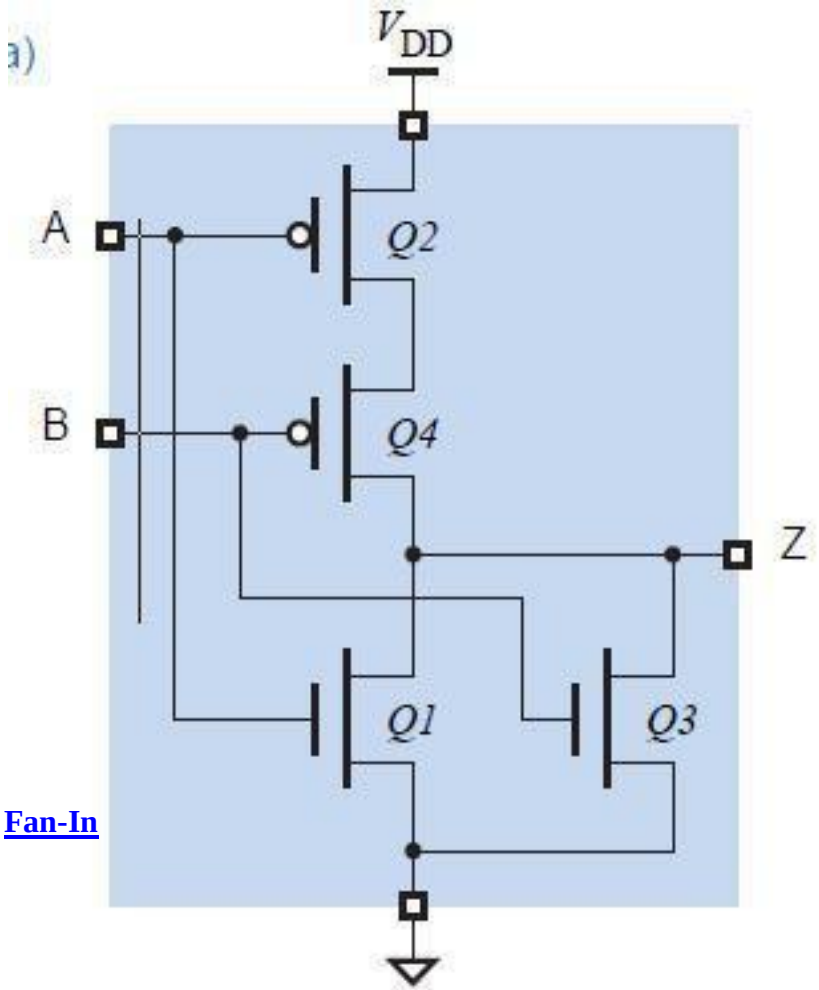
(b)

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

(c)



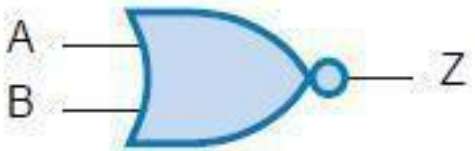
CMOS NOR Gate



(b)

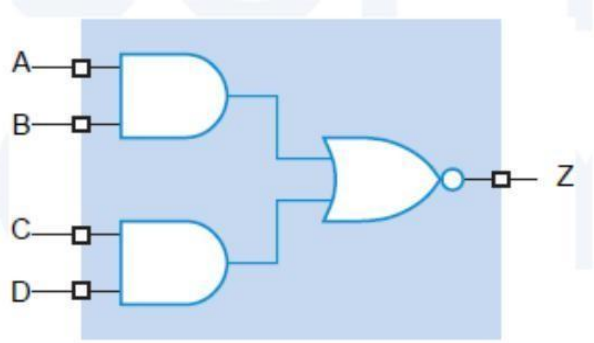
A	B	Q1	Q2	Q3	Q4	Z
L	L	off	on	off	on	H
L	H	off	on	on	off	L
H	L	on	off	off	on	L
H	H	on	off	on	off	L

(c)

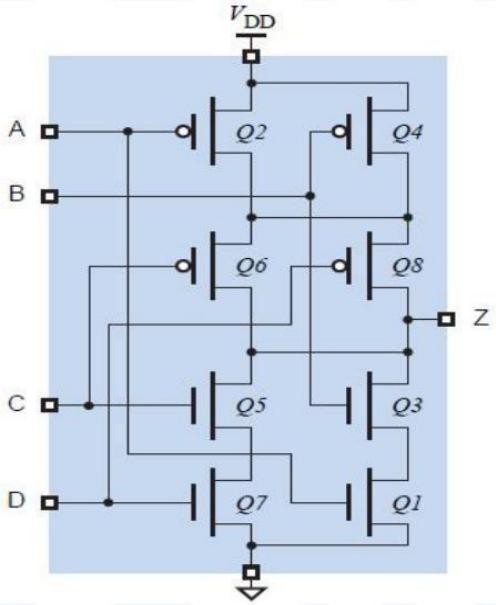


Fan-In

CMOS AOI logic



(a)

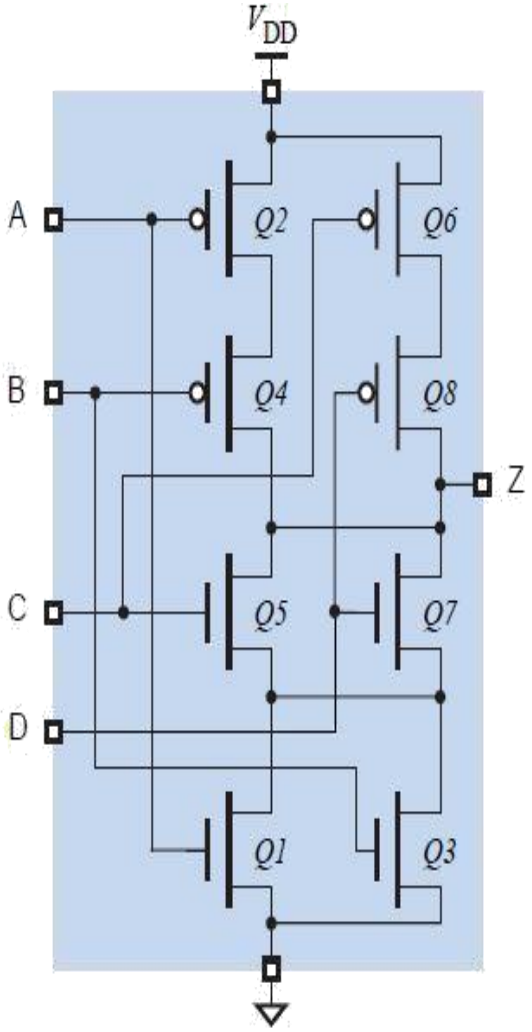


(b)

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

CMOS OAI logic

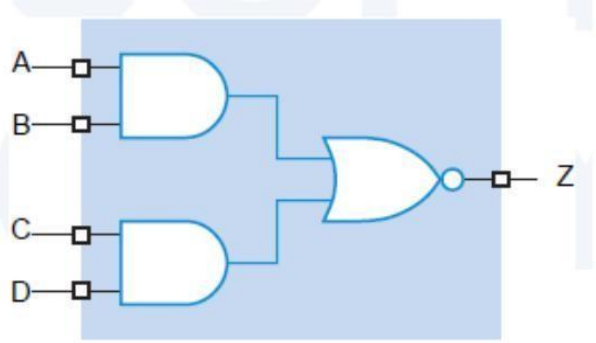
(a)



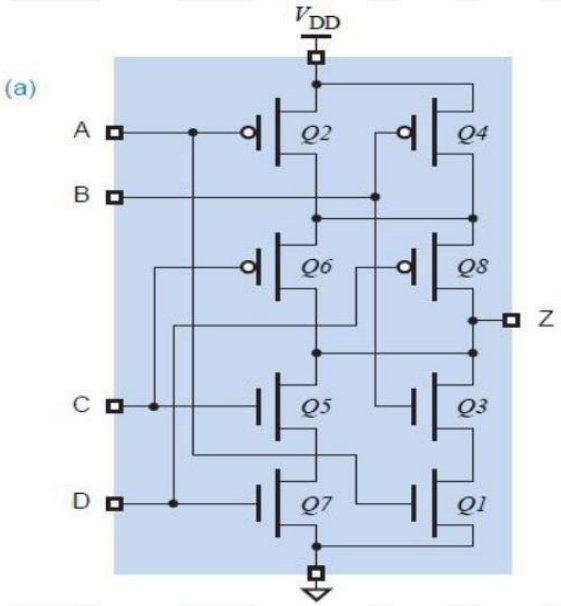
(b)

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

CMOS AND gate



(a)

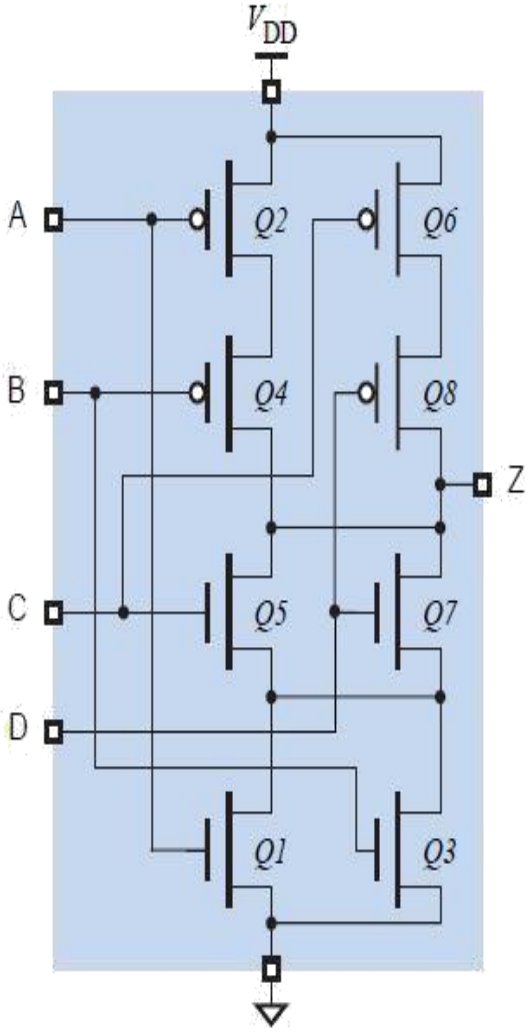


(b)

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

CMOS OR gate

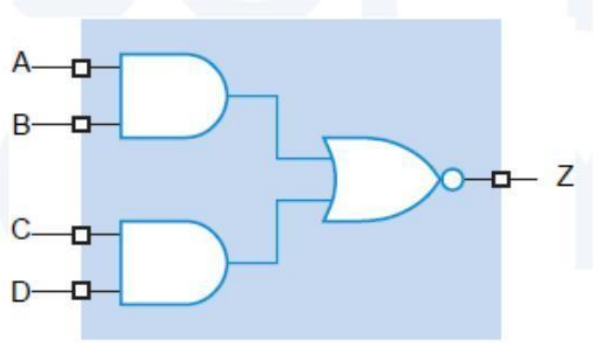
(a)



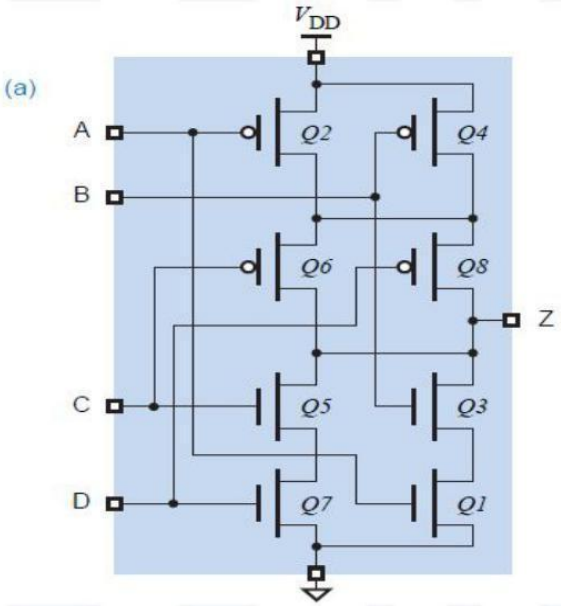
(b)

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

CMOS XOR gate



(a)

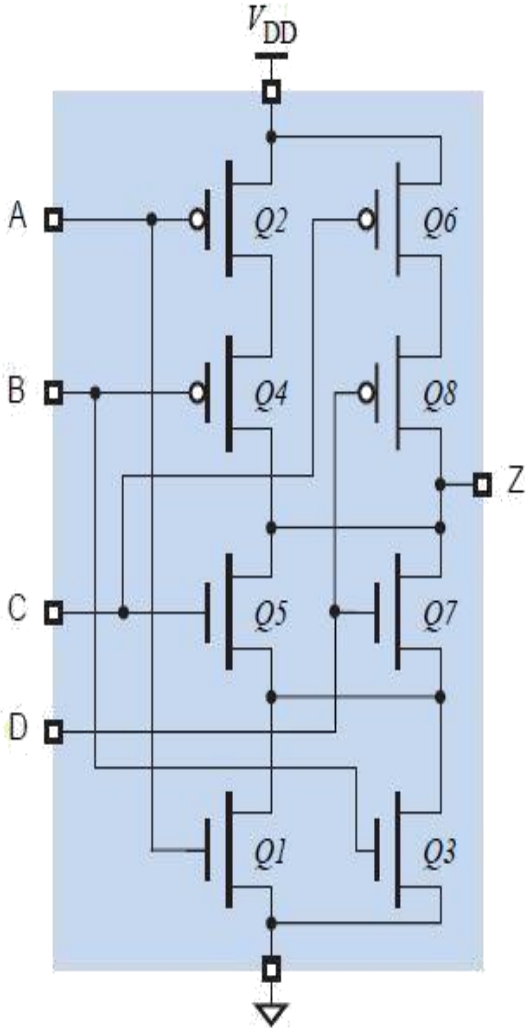


(b)

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

CMOS XNOR gate

(a)



(b)

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

Electrical Behavior of CMOS Circuits

- Logic voltage levels
- DC noise margins
- Fanout: - It affects the speed at which the output changes from one state to another.
- Speed: - It depends on both the internal structure of the device and the characteristics of the other devices that it drives.
- Power consumption: - The power consumed by a CMOS device depends on how often its output changes between LOW and HIGH.
- Noise :- Noise can be generated by a number of sources:
- Electrostatic discharge: - CMOS devices can be destroyed just by touching it.
- Open-drain outputs: - In the HIGH state, such outputs are effectively a “no-connection,” which is useful in some applications.
- Three-state outputs: - Some CMOS devices have an extra “output enable” control input that can be used to disable both the p-channel pull-up transistors and the n-channel pull-down transistors.

CMOS Steady-State Electrical Behavior

- **Logic Levels and Noise Margins**:- The power-supply voltage VCC and ground are often called the power supply rails.
- **Circuit Behavior with Resistive Loads**: - CMOS gate inputs have very high impedance and consume very little current from the circuits that drive them.
- **Circuit Behavior with Non-ideal Inputs**:- If the input voltage is not close to the power-supply rail, then the "on" transistor may not be fully "on" and its resistance may increase and the "off" transistor may not be fully "off' and its resistance.

CMOS Steady-State Electrical Behavior

- **Fanout:-** It affects the speed at which the output changes from one state to another
- **Effects of Loading:-** Loading an output beyond its rated fanout has several effects:
- **Unused Inputs**:- Connects to power rails.
- **Current Spikes and Decoupling Capacitors:-** When a CMOS output switches between LOW and HIGH, current flows from VCC to ground through the partially-on p- and n-channel transistors. These currents, often called current spikes

CMOS Dynamic Electrical Behavior

- Speed depends on two characteristics, transition time and propagation delay
- Transition Time:- The amount of time that the output of a logic circuit takes to change from one state to another is called the transition time.
- Propagation Delay:- The propagation delay tp of a signal path is the amount of time that it takes for a change in the input signal to produce a change in the output signal.

CMOS Dynamic Electrical Behavior

- Speed depends on two characteristics, transition time and propagation delay
- Transition Time:- The amount of time that the output of a logic circuit takes to change from one state to another is called the transition time.
- Propagation Delay:- The propagation delay tp of a signal path is the amount of time that it takes for a change in the input signal to produce a change in the output signal.

Electrical Behavior of CMOS Circuits

- Logic voltage levels
- DC noise margins
- Fanout: - It affects the speed at which the output changes from one state to another.
- Speed: - It depends on both the internal structure of the device and the characteristics of the other devices that it drives.
- Power consumption: - The power consumed by a CMOS device depends on how often its output changes between LOW and HIGH.
- Noise :- Noise can be generated by a number of sources:
- Electrostatic discharge: - CMOS devices can be destroyed just by touching it.
- Open-drain outputs: - In the HIGH state, such outputs are effectively a “no-connection,” which is useful in some applications.
- Three-state outputs: - Some CMOS devices have an extra “output enable” control input that can be used to disable both the p-channel pull-up transistors and the n-channel pull-down transistors.

CMOS Steady-State Electrical Behavior

- **Logic Levels and Noise Margins**:- The power-supply voltage VCC and ground are often called the power supply rails.
- **Circuit Behavior with Resistive Loads**: - CMOS gate inputs have very high impedance and consume very little current from the circuits that drive them.
- **Circuit Behavior with Non-ideal Inputs**:- If the input voltage is not close to the power-supply rail, then the "on" transistor may not be fully "on" and its resistance may increase and the "off" transistor may not be fully "off' and its resistance.

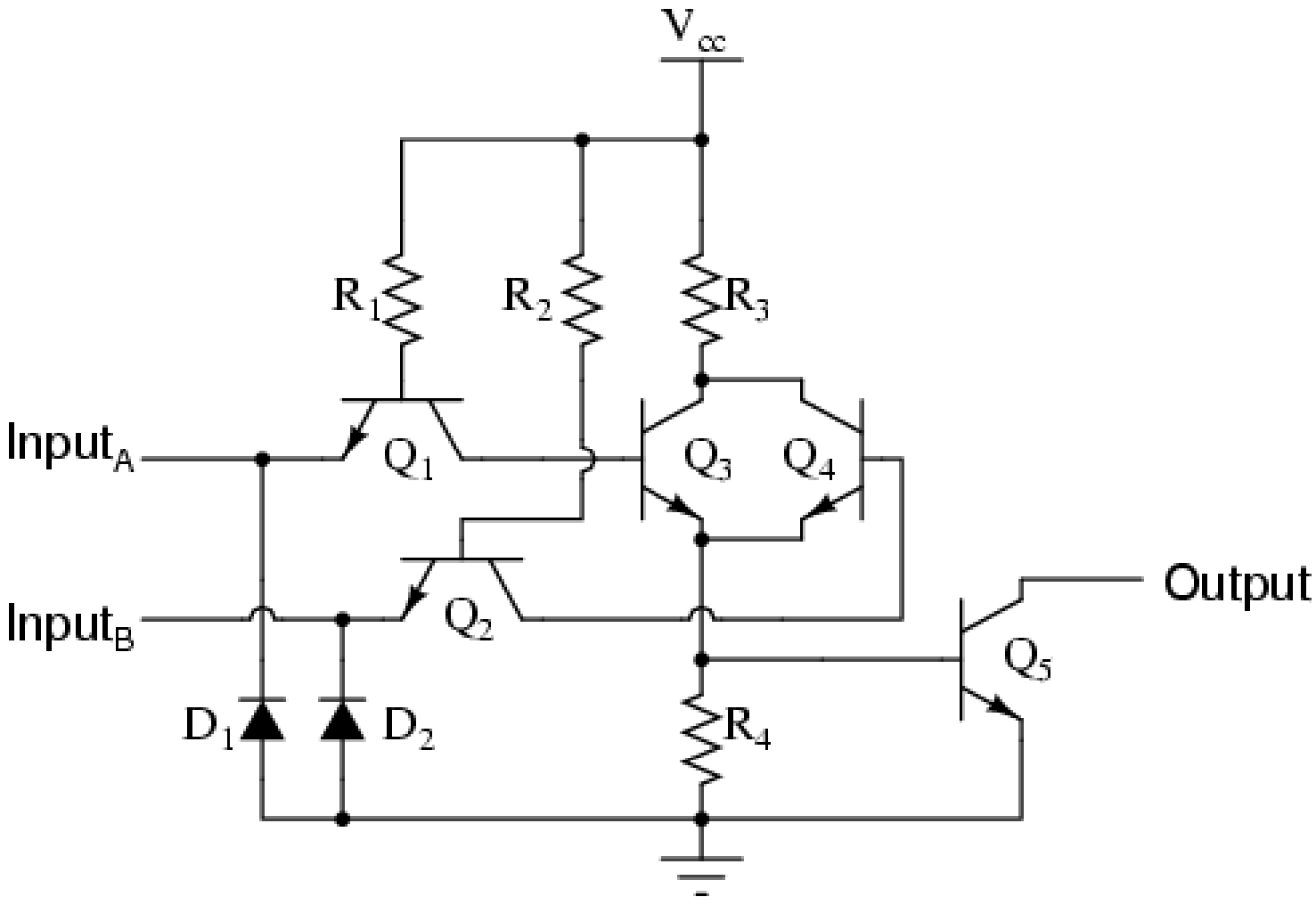
CMOS Steady-State Electrical Behavior

- **Fanout:-** It affects the speed at which the output changes from one state to another
- **Effects of Loading:-** Loading an output beyond its rated fanout has several effects:
- **Unused Inputs**:- Connects to power rails.
- **Current Spikes and Decoupling Capacitors:-** When a CMOS output switches between LOW and HIGH, current flows from VCC to ground through the partially-on p- and n-channel transistors. These currents, often called current spikes

CMOS Dynamic Electrical Behavior

- Speed depends on two characteristics, transition time and propagation delay
- Transition Time:- The amount of time that the output of a logic circuit takes to change from one state to another is called the transition time.
- Propagation Delay:- The propagation delay tp of a signal path is the amount of time that it takes for a change in the input signal to produce a change in the output signal.

TTL NOR gate



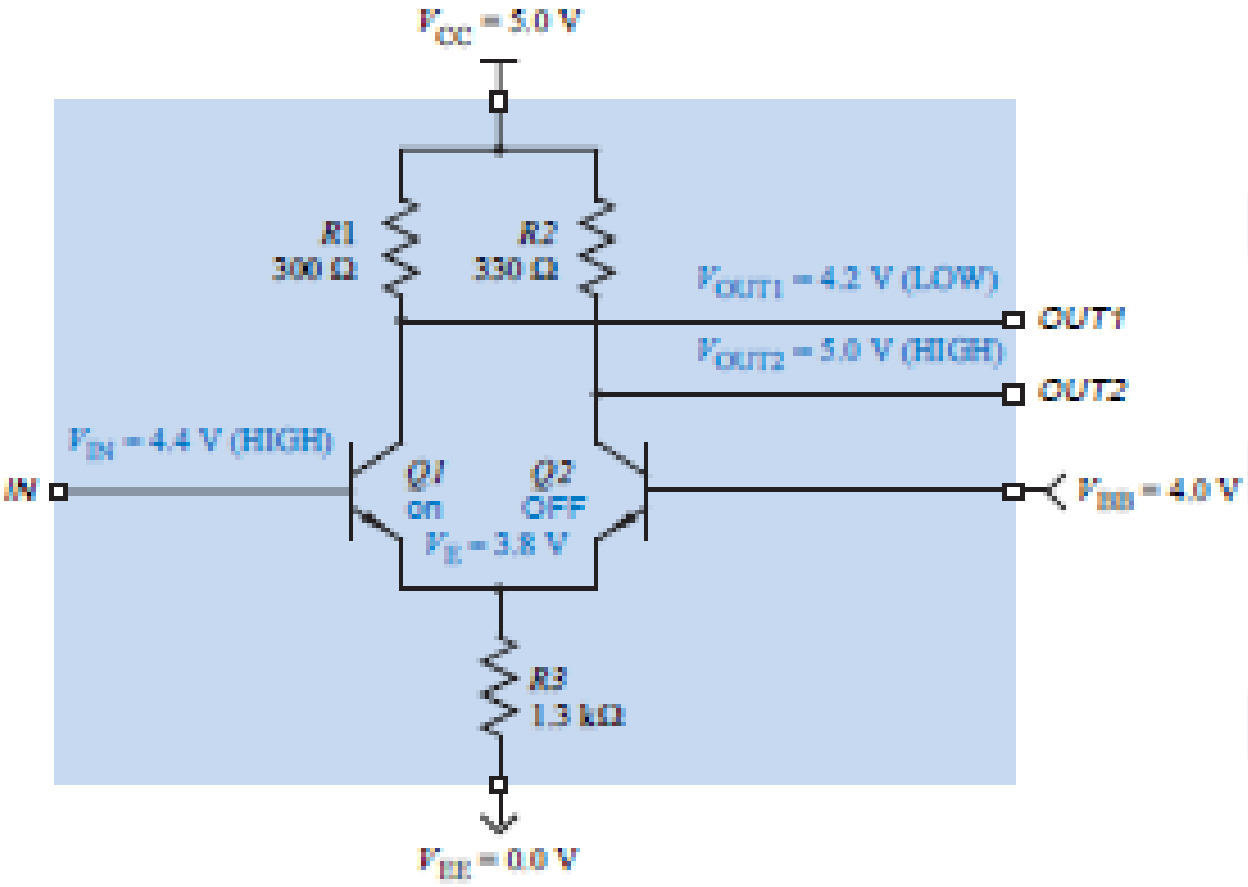
CMOS/TTL interfacing

When 5V supply is given to **TTL** and **CMOS** ICs, logic levels of **TTL** and **CMOS** are different.

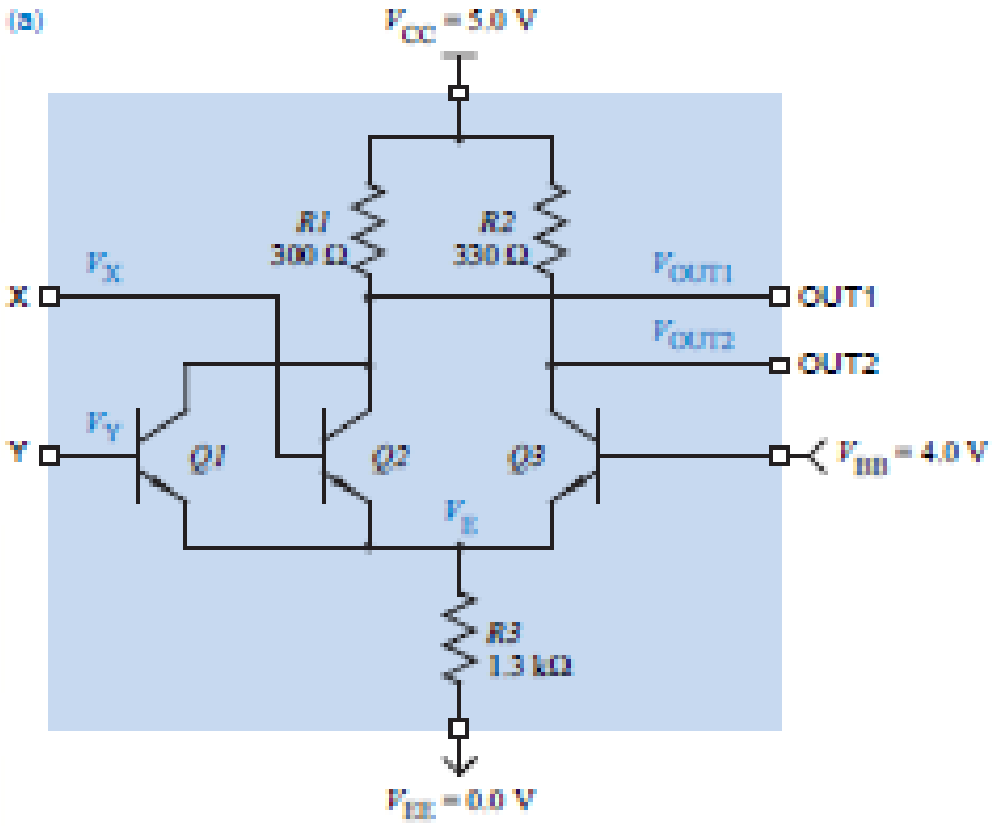
One **TTL** IC can drive any number of **CMOS** ICs.

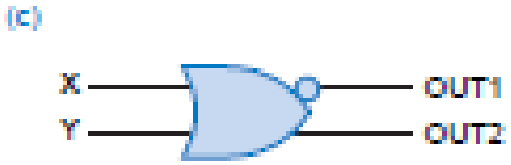
However, **TTL** output in 'high state' yields 2.4 Volt which is lower than the minimum voltage required by **CMOS** IC (which is 3.5V)

Emitter Coupled Logic



Emitter Coupled Logic





Emitter Coupled Logic

X	Y	V_X	V_Y	$Q1$	$Q2$	$Q3$	V_E	V_{OUT1}	V_{OUT2}	OUT1	OUT2
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

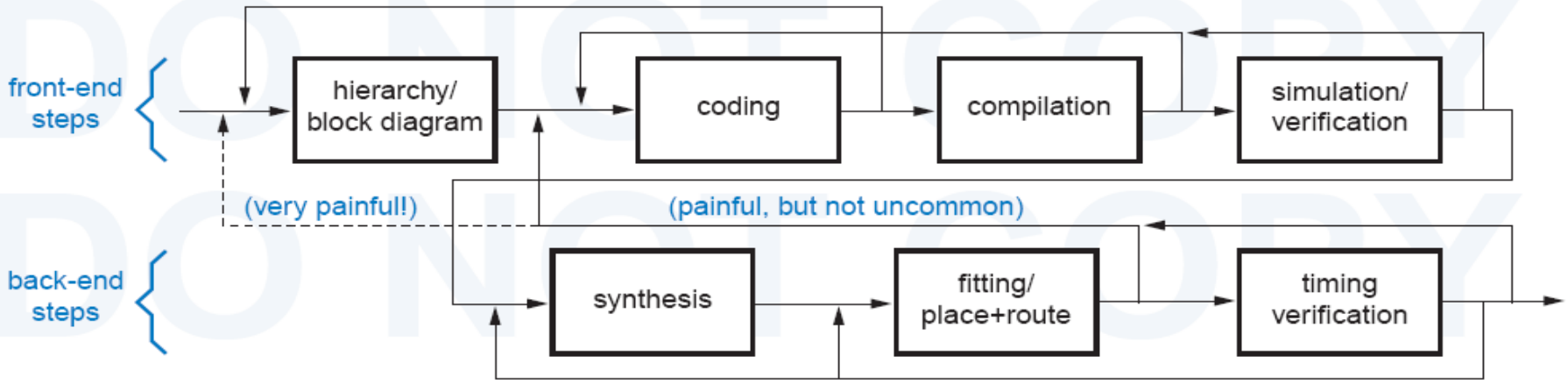
X	Y	OUT1	OUT2
0	0	1	0
0	1	0	1
1	0	0	1
1	1	0	1

UNIT – II

THE VHDL HDL AND ITS ELEMENTS

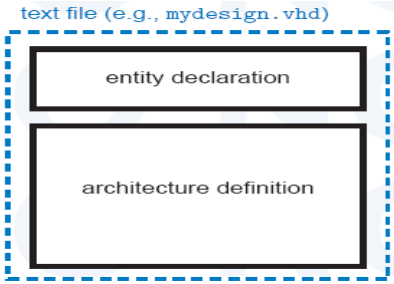
Design Flow

- There are several steps in a VHDL-based design process, often called the design flow

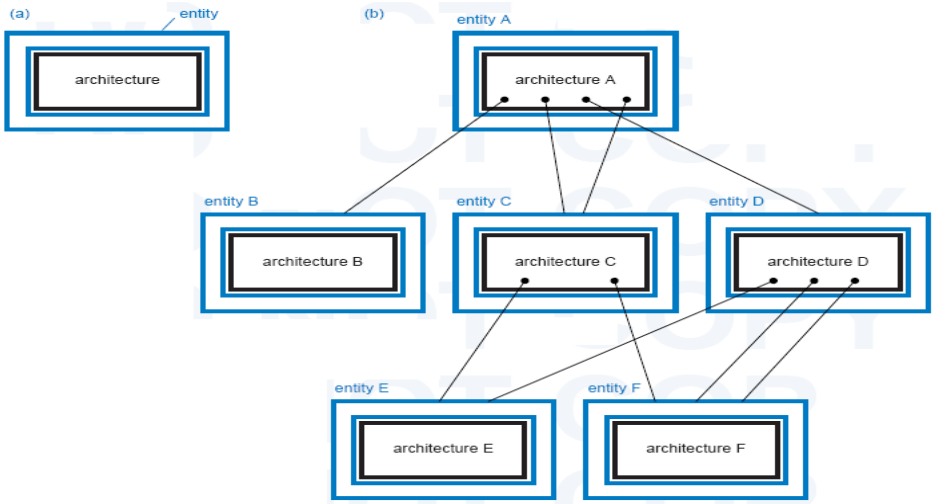


Steps in a VHDL or other HDL-based design flow.

VHDL Program Structure



VHDL program file structure



Syntax of a VHDL entity declaration

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

Architecture body

- The architecture body looks as follows,
- **architecture** architecture_name **of** NAME_OF_ENTITY **is**
- **--** Declarations
- -- components declarations
- -- signal declarations
- -- constant declarations
- -- function declarations
- -- procedure declarations
- -- type declarations
- **begin**
- -- Statements
- **end** architecture_name;

Types and Constants

- *VHDL predefined types*

bit	character	severity_level
bit_vector	integer	string
Boolean	real	time

std_logic type

- Definition of VHDL std_logic type
-
- 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;

TYPE and CONSTANTS

- type *type-name* is array (*start* to *end*) of *element-type*;
- type *type-name* is array (*start* downto *end*) of *element-type*;
- type *type-name* is array (*range-type*) of *element-type*;
- type *type-name* is array (*range-type* range *start* to *end*) of *element-type*;
- type *type-name* is array (*range-type* range *start* downto *end*) of *element-type*;
-
- constant BUS_SIZE: integer := 32; -- width of component
- constant MSB: integer := BUS_SIZE-1; -- bit number of MSB
- constant Z: character := 'Z'; -- synonym for Hi-Z value

Structural Design Elements

- In VHDL, each *concurrent statement* executes simultaneously with the other concurrent statements in the Same architecture body.
- The most basic of VHDL's concurrent statements is the component *statement*.

Syntax of a VHDL component statement

- *label*: *component-name* port map(*signal1*, *signal2*, ..., *signaln*);
- *label*: *component-name* port map(*port1=>signal1*, *port2=>signal2*, ..., *portn=>signaln*);

Syntax of a VHDL component declaration

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

Example Structural VHDL program

```
library IEEE;          use IEEE.std_logic_1164.all;
entity prime is
port ( N: in STD_LOGIC_VECTOR (3 downto 0);
F: out STD_LOGIC );
end prime;
architecture prime1_arch of prime is
signal N3_L, N2_L, N1_L: STD_LOGIC;
signal N3L_N0, N3L_N2L_N1, N2L_N1_N0, N2_N1L_N0: STD_LOGIC;
component INV port (I: in STD_LOGIC; O: out STD_LOGIC); end component;
component AND2 port (I0,I1: in STD_LOGIC; O: out STD_LOGIC); end component;
component AND3 port (I0,I1,I2: in STD_LOGIC; O: out STD_LOGIC); end component;
component OR4 port (I0,I1,I2,I3: in STD_LOGIC; O: out STD_LOGIC); end component;
begin
U1: INV port map (N(3), N3_L);       U2: INV port map (N(2), N2_L);
U3: INV port map (N(1), N1_L);       U4: AND2 port map (N3_L, N(0), N3L_N0);
U5: AND3 port map (N3_L, N2_L, N(1), N3L_N2L_N1);
U6: AND3 port map (N2_L, N(1), N(0), N2L_N1_N0);
U7: AND3 port map (N(2), N1_L, N(0), N2_N1L_N0);
U8: OR4 port map (N3L_N0, N3L_N2L_N1, N2L_N1_N0, N2_N1L_N0, F);
end prime1_arch;
```

Data flow design elements

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

entity V74x138 is
port (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
end V74x138;

architecture V74x138_a of V74x138 is signal Y_L_i: STD_LOGIC_VECTOR (0 to 7);
begin
with A select Y_L_i <= "01111111" when "000",
"10111111" when "001",
"11011111" when "010",
"11101111" when "011",
"11110111" when "100",
"11111011" when "101",
"11111101" when "110",
"11111110" when "111",
"11111111" when others;
Y_L <= Y_L_i when (G1 and not G2A_L and not G2B_L)='1' else "11111111"; end V74x138_a
```

Data flow design elements

- library IEEE;
- use IEEE.std_logic_1164.all;
- decoder.
- entity V3to8dec is
- port (G1, G2, G3: in STD_LOGIC;
- A: in STD_LOGIC_VECTOR (2 downto 0); Y: out STD_LOGIC_VECTOR (0 to 7));
- end V3to8dec;
- architecture V3to8dec_a of V3to8dec is signal Y_s: STD_LOGIC_VECTOR (0 to 7);
- begin
- with A select Y_s <= "10000000" when "000",
- "01000000" when "001",
- "00100000" when "010",
- "00010000" when "011",
- "00001000" when "100",
- "00000100" when "101",
- "00000010" when "110",
- "00000001" when "111",
- "00000000" when others;
- Y <= Y_s when (G1 and G2 and G3)='1' else "00000000";
- end V3to8dec_a;

Data flow design elements

- library IEEE;
- use IEEE.std_logic_1164.all;
- decoder.
- entity V3to8dec is
- port (G1, G2, G3: in STD_LOGIC;
- A: in STD_LOGIC_VECTOR (2 downto 0); Y: out STD_LOGIC_VECTOR (0 to 7));
- end V3to8dec;
- architecture V3to8dec_a of V3to8dec is signal Y_s: STD_LOGIC_VECTOR (0 to 7);
- begin
- with A select Y_s <= "10000000" when "000",
- "01000000" when "001",
- "00100000" when "010",
- "00010000" when "011",
- "00001000" when "100",
- "00000100" when "101",
- "00000010" when "110",
- "00000001" when "111",
- "00000000" when others;
- Y <= Y_s when (G1 and G2 and G3)='1' else "00000000";
- end V3to8dec_a;

Data flow design elements

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

entity V74x138 is
port (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
end V74x138;

architecture V74x138_a of V74x138 is signal Y_L_i: STD_LOGIC_VECTOR (0 to 7);
begin
with A select Y_L_i <= "01111111" when "000",
"10111111" when "001",
"11011111" when "010",
"11101111" when "011",
"11110111" when "100",
"11111011" when "101",
"11111101" when "110",
"11111110" when "111",
"11111111" when others;
Y_L <= Y_L_i when (G1 and not G2A_L and not G2B_L)='1' else "11111111"; end V74x138_a
```

Data flow design elements

- library IEEE;
- use IEEE.std_logic_1164.all;
- decoder.
- entity V3to8dec is
- port (G1, G2, G3: in STD_LOGIC;
- A: in STD_LOGIC_VECTOR (2 downto 0); Y: out STD_LOGIC_VECTOR (0 to 7));
- end V3to8dec;
- architecture V3to8dec_a of V3to8dec is signal Y_s: STD_LOGIC_VECTOR (0 to 7);
- begin
- with A select Y_s <= "10000000" when "000",
- "01000000" when "001",
- "00100000" when "010",
- "00010000" when "011",
- "00001000" when "100",
- "00000100" when "101",
- "00000010" when "110",
- "00000001" when "111",
- "00000000" when others;
- Y <= Y_s when (G1 and G2 and G3)='1' else "00000000";
- end V3to8dec_a;

Time dimension and simulation synthesis

- Rise and fall times only partially describe the dynamic behavior of a logic element; we need additional parameters to relate output timing to input timing.

•

A *signal path* is the electrical path from a particular input signal to a particular output signal of a logic element. The *propagation delay t*p of a signal path is the amount of time that it takes for a change in the input signal to produce a change in the output signal.

•

A complex logic element with multiple inputs and outputs may specify a different value of *t*p for each different signal path. Also, different values may be specified for a particular signal path, depending on the direction of the output

- change. Ignoring rise and fall times,

Synthesis

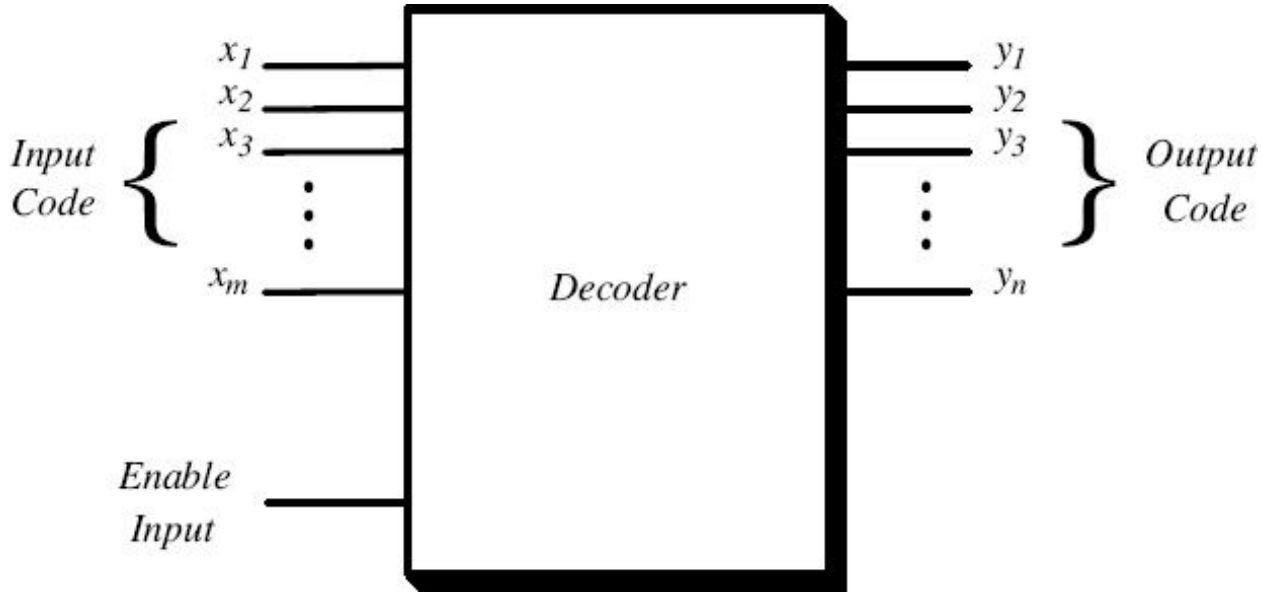
- Loproblem we usu- **DESIGN** ally start out with an informal (word or thought) description of the circuit. Often the most challenging and creative part of design is to formalize the circuit description, the circuit's input and output signals and specifying its functional behavior by means of truth tables and equations.
- Once we've created the formal circuit description, we can usually follow a "turn-the-crank" synthesis procedure to obtain a logic diagram for a circuit with the required functional behavior.
- The material in the first four sections of this chapter is the basis for "turn-the-crank" procedures, the crank is turned by hand or by a computer. The last two sections describe actual design languages, ABEL and VHDL. When we create a design using one of these languages, a computer program can perform the synthesis steps.
- Logic circuit design is a superset of synthesis, since in a real design

UNIT – III

COMBINATIONAL LOGIC DESIGN USING VHDL

Decoders

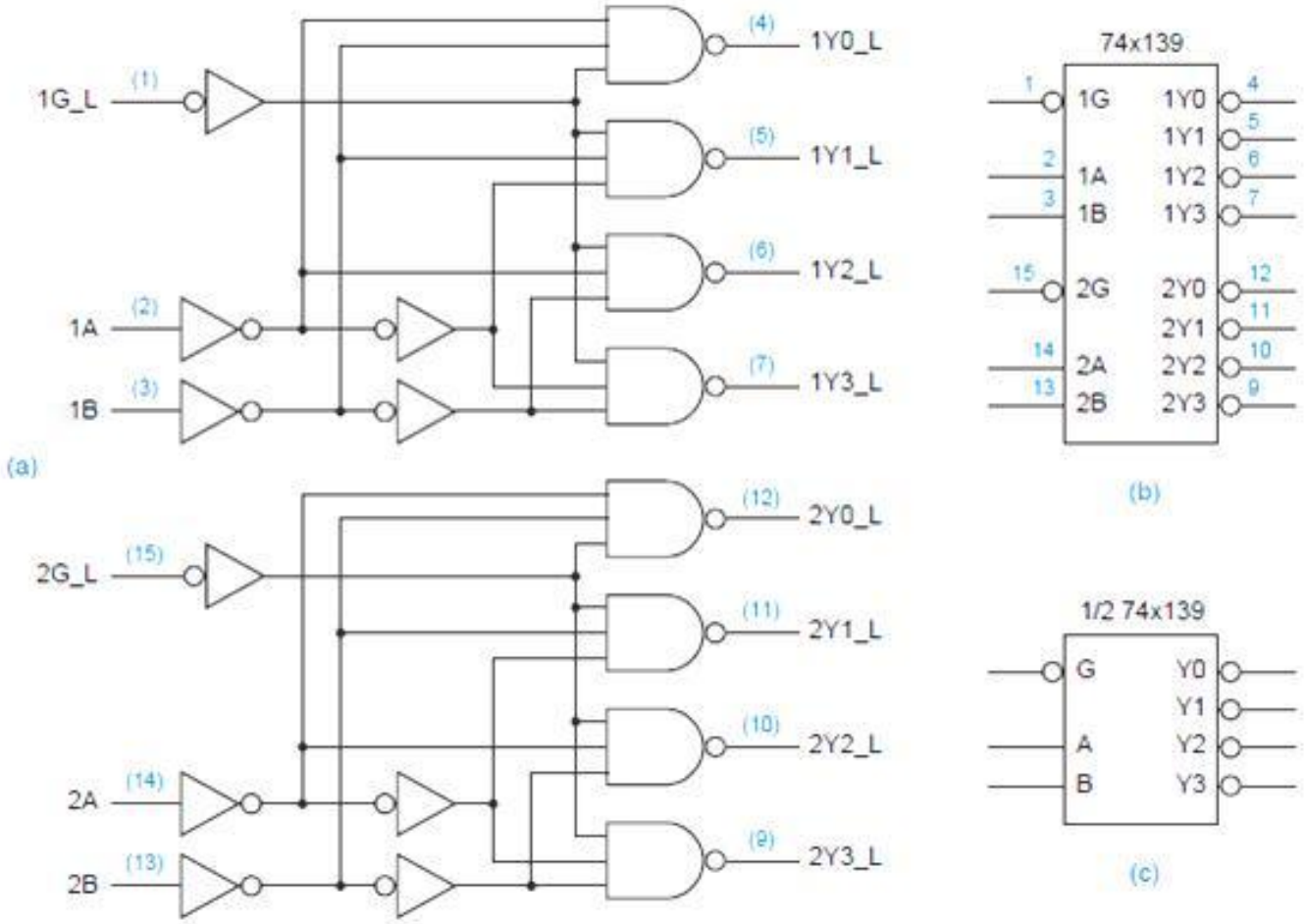
- A *decoder* is a multiple-input, multiple-output logic circuit that converts coded inputs into coded outputs.
- The general structure of a Decoder circuit is shown in figure.



- *The most common decoder circuit is an n-to-2n decoder or binary decoder*

74x139 Dual 2-to-4 Decoder

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



74x139 Dual 2-to-4 Decoder

- The Truth Table for one-half of a 74x139 dual 2-to-4 Decoder.

Inputs			*Outputs*			
G_L	B	A	Y3_L	Y2_L	Y1_L	Y0_L
1	x	x	1	1	1	1
0	0	0	1	1	1	0
0	0	1	1	1	0	1
0	1	0	1	0	1	1
0	1	1	0	1	1	1

- The outputs and the enable input of the 74x139 are active-low.
- Most MSI decoders were originally designed with active-low outputs, since TTL inverting gates are generally faster than non-inverting ones.

VHDL CODE FOR 74x139

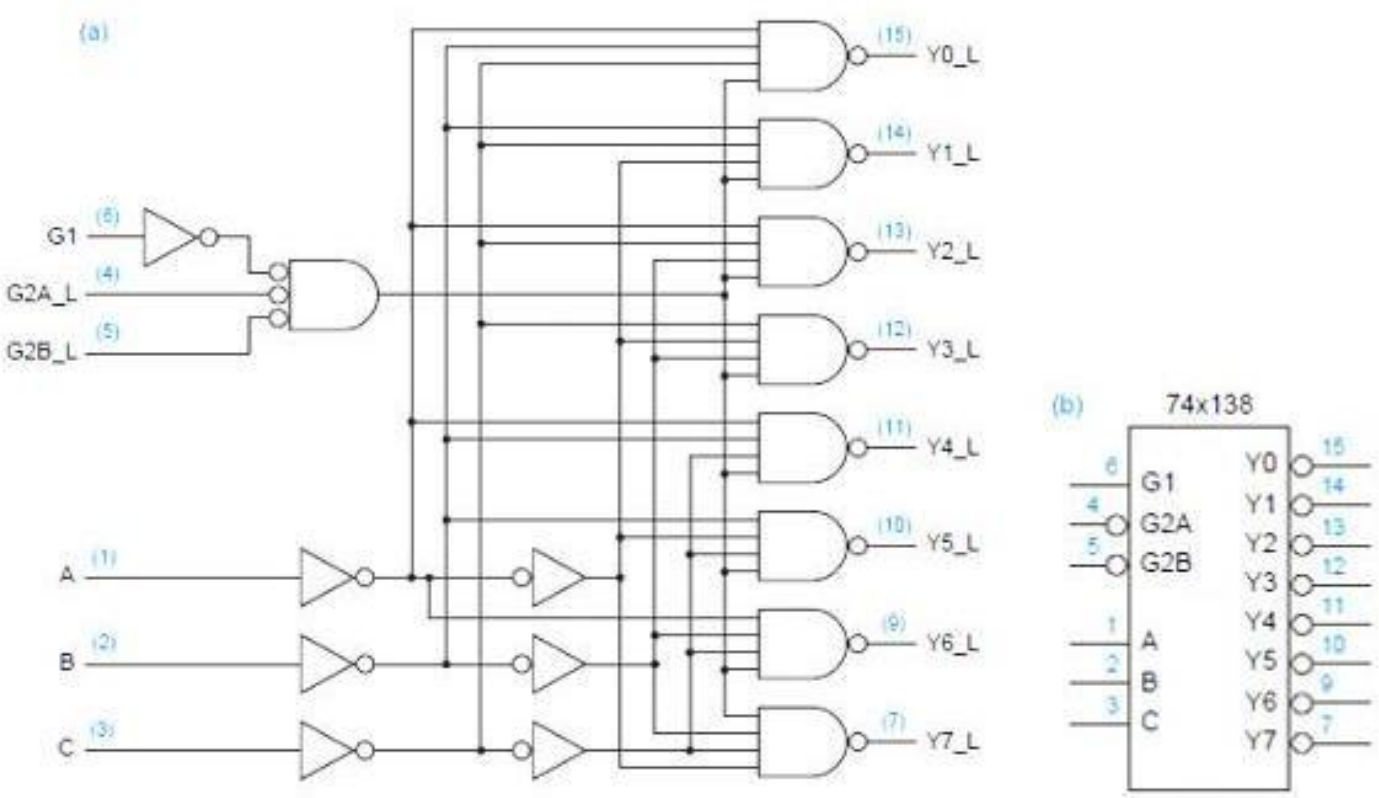
```
library IEEE;
use IEEE.std_logic_1164.all;

entity dec74x139 is
port (    EN_L:    in       STD_LOGIC;
          A:       in       STD_LOGIC_VECTOR (1 downto 0);
          Y_L:     out      STD_LOGIC_VECTOR (3 downto 0) );
end dec74x139;

Architecture arch_dec74x139 of dec74x139 is
    signal Y_s: STD_LOGIC_VECTOR (3 downto 0);
begin
```

VHDL CODE FOR 74x139

```
process(EN_L, A, Y_s)
begin
      case A is
            when "00"         => Y_s <= "1110";
            when "01"         => Y_s <= "1101";
            when "10"         => Y_s <= "1011";
            when "11"         => Y_s <= "0111";
            when others       => Y_s <= "1111";
      end case;

      if EN_L='0' then Y_L <= Y_s;
      else Y_L <= "1111";
      end if;

end process;
end behavior;
```

74x138 3-to-8 Decoder

The *74x138* is a commercially available MSI 3-to-8 decoder. 74x138 has active-low outputs, and it has three enable inputs (G1, /G2A, /G2B), all of which must be asserted for the selected output to be asserted.



74x138 3-to-8 Decoder Truth Table

Truth Table for a 74x138 3-to-8 Decoder

Inputs						Outputs							
G1	G2A_L	G2B_L	C	B	A	Y7_L	Y6_L	Y5_L	Y4_L	Y3_L	Y2_L	Y1_L	Y0_L
0	x	x	x	x	x	1	1	1	1	1	1	1	1
x	1	x	x	x	x	1	1	1	1	1	1	1	1
x	x	1	x	x	x	1	1	1	1	1	1	1	1
1	0	0	0	0	0	1	1	1	1	1	1	1	0
1	0	0	0	0	1	1	1	1	1	1	1	0	1
1	0	0	0	1	0	1	1	1	1	1	0	1	1
1	0	0	0	1	1	1	1	1	1	0	1	1	1
1	0	0	1	0	0	1	1	1	0	1	1	1	1
1	0	0	1	0	1	1	1	0	1	1	1	1	1
1	0	0	1	1	0	1	0	1	1	1	1	1	1
1	0	0	1	1	1	0	1	1	1	1	1	1	1

VHDL CODE FOR 74x138

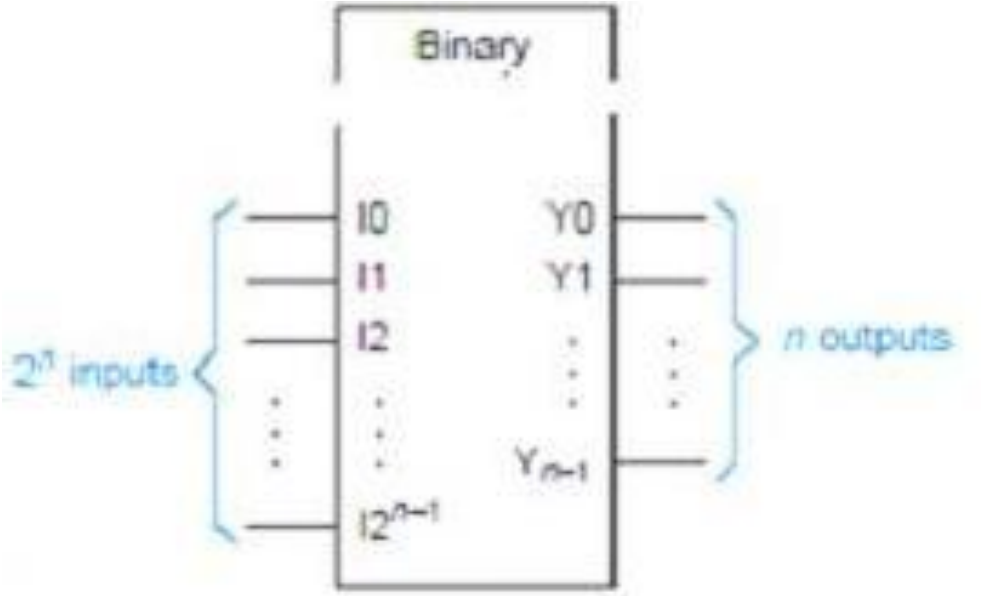
```
library IEEE;
use IEEE.std_logic_1164.all;

entity Dec74x138 is
port (    G1, G2A_L, G2B_L:      in STD_LOGIC;
          A:                     in STD_LOGIC_VECTOR (2 downto 0);
          Y_L:                   out STD_LOGIC_VECTOR (0 to 7) );
end Dec74x138;

architecture behavior of V3to8dec is
    signal Y_s: STD_LOGIC_VECTOR (0 to 7);
begin
```

VHDL CODE FOR 74x138

```
process(A, G1, G2A_L, G2B_L, Y_s)
begin
      case A is
            when "000"        => Y_s <= "01111111";
            when "001"        => Y_s <= "10111111";
            when "010"        => Y_s <= "11011111";
            when "011"        => Y_s <= "11101111";
            when "100"        => Y_s <= "11110111";
            when "101"        => Y_s <= "11111011”;
            when "110"        => Y_s <= "11111101";
            when "111"        => Y_s <= “11111110";
            when others       => Y_s <= "11111111";
       end case;
    if (G1 and not G2A_L and not G2B_L)='1' then          Y <= Y_s;
    else              Y_L <= "11111111";
    end if;
end process;
end behavior;
```

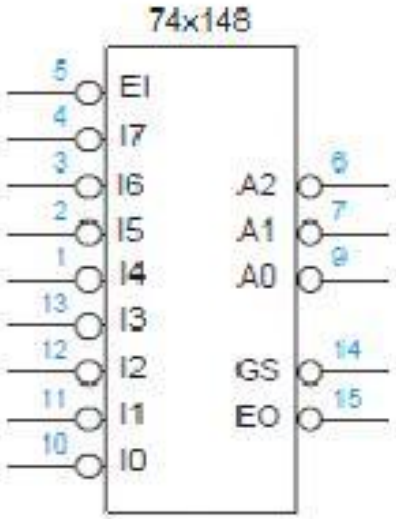
ENCODERS

Encoder to build is a 2*n*-to-*n* or *binary encoder*. Its input code is the 1 out-of-2*n* code and its output code is *n*-bit binary.



The 74x148 Priority Encoder

- The *74x148* is a commercially available, MSI 8-input priority encoder.



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

The 74x148 Priority Encoder

- It has an enable input, EI_L, that must be asserted for any of its outputs to be asserted.

- 74x148 has a GS_L output that is asserted when the device is enabled and one or more of the request inputs is asserted, called as “Group Select”.

- The EO_L signal is an enable *output* designed to be connected to the EI_L input of another ’148 that handles lower-priority requests. EO_L is asserted if EI_L is asserted but no request input is asserted; thus, a lower-priority ’148 may be enabled.

VHDL Code for 74x148 Priority Encoder

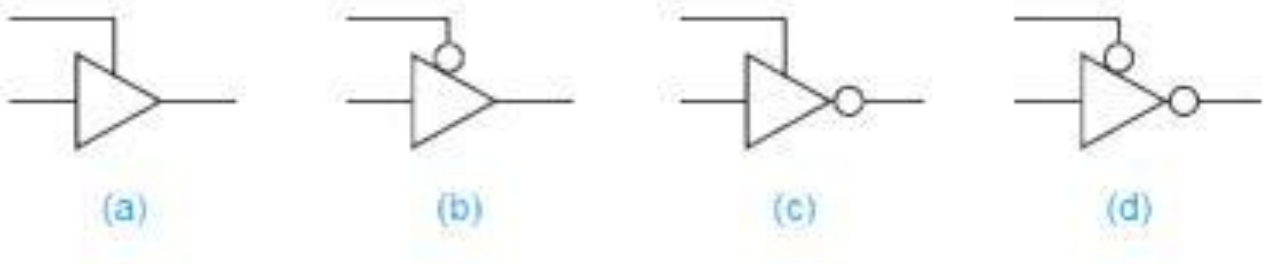
```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity enc_74x148 is
port (    EI_L:              in       STD_LOGIC;
          I_L:               in       STD_LOGIC_VECTOR (7 downto 0);
          A_L:               out      STD_LOGIC_VECTOR (2 downto 0);
          EO_L, GS_L:        out STD_LOGIC);
end enc_74x148;

Architecture arch_enc_74x148 of enc_74x148 is
     signal EI:              STD_LOGIC;
     signal I:               STD_LOGIC_VECTOR (7 downto 0);
     signal EO, GS:          STD_LOGIC;
     signal A:               STD_LOGIC_VECTOR (2 downto 0);
begin
```

VHDL Code for 74x148 Priority Encoder

```
process (EI_L, I_L, EI, EO, GS, I, A)
    variable j: INTEGER range 7 downto 0;
begin
    EI <= not EI_L;  I <= not I_L;  EO <= '1';  GS <= '0'; A <= "000";
    if (EI)='0' then EO <= '0';
    else for j in 7 downto 0 loop
    elsif I(j)='1' then
            GS <= '1';  EO <= '0';
            A <= CONV_STD_LOGIC_VECTOR(j,3);
    end if;
    end loop;
    end if;
    EO_L <= not EO;          GS_L <= not GS;              A_L <= not A;
end process;
end arch_enc_74x148;
```

Three state devices

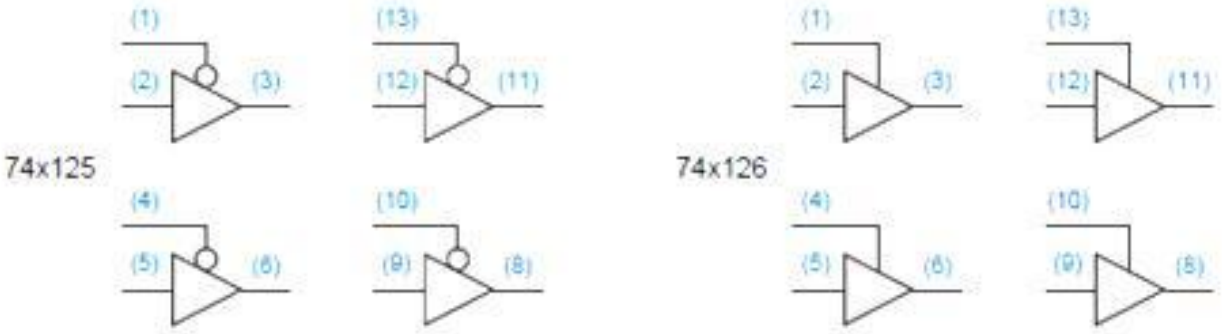
- The electrical design of CMOS and TTL devices whose outputs may be in one of three states—0, 1, or Hi-Z.
- The most basic three-state device is a *three-state buffer*, often called a *three-state driver*.
- The logic symbols for four physically different three-state buffers.



- (a) noninverting, active-high enable (b) noninverting, active-low enable (c) inverting, active-high enable (d) inverting, active-low enable

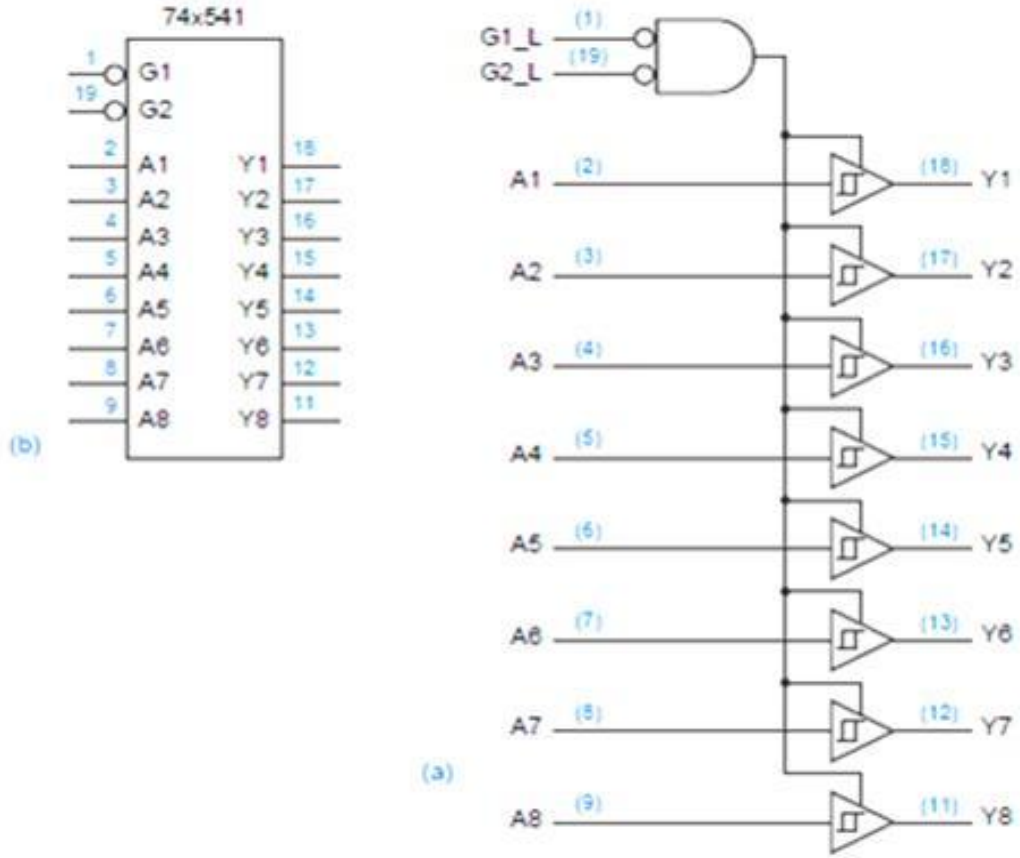
Standard SSI and MSI Three-State Buffers

- *74x125* and *74x126*, each of which contains four independent non-inverting three-state buffers in a 14-pin package.
- The three-state enable inputs in the 74x125 are active low, and in the 74x126 they are active high.



Standard SSI and MSI Three-State Buffers

- *74x541* octal non-inverting three-state buffer.



- The *74x540* is identical to the 74x541 except that it contains inverting buffers.

VHDL Code for 74x541

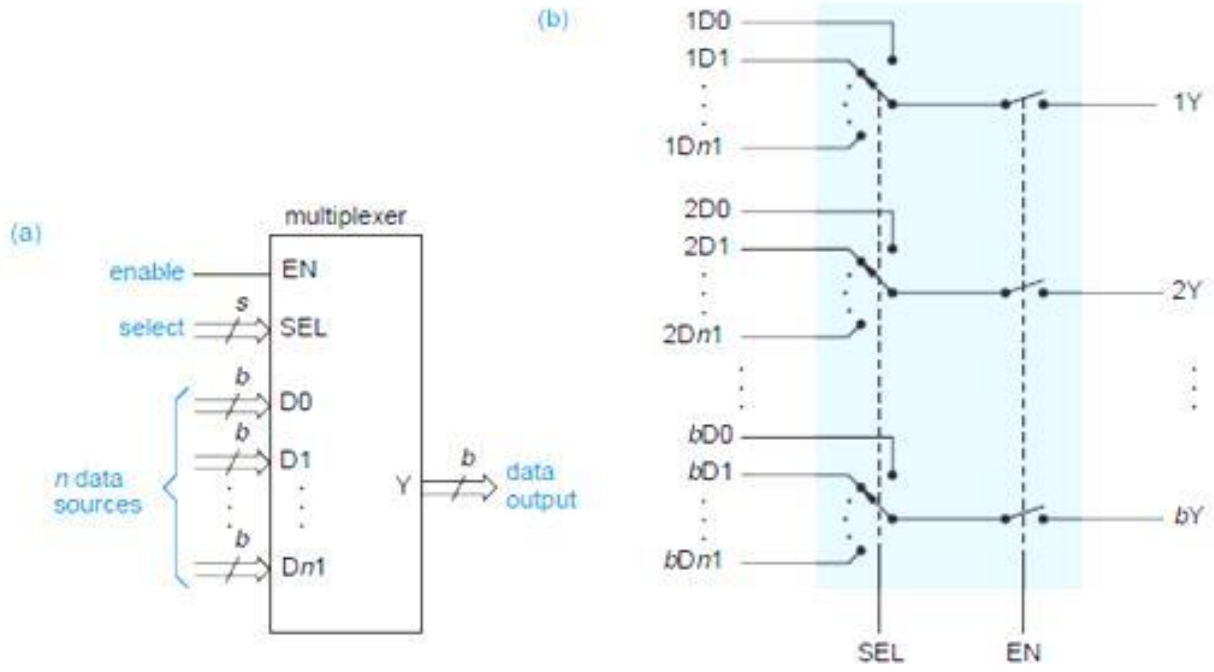
```
Library ieee;
Use ieee.std_logic_1164.all;

Entity IC74541 is
Port(    A:          in      std_logic_vector(7 downto 0);
         G1_L,G2_L:      in      std_logic;
         Y:          out   std_logic_vector(7 downto 0));
End IC74541;

Architecture behav of IC74541 is
Begin
Process(a,G1_L, G2_L)
Begin
    If(G1_L='0' and G2_L='0' )     then     Y<=A;
    else  Y<="ZZZZZZZZ";
    End if;
End process;
End behav;
```

VHDL Code for 74x151

```
Library ieee;
Use ieee.std_logic_1164.all;
Entity mux74x151 is
   Port( EN_L:    in std_logic;
         S:       in std_logic_vector(2 downto 0);
         D:       in std_logic_vector(7 downto 0);
         Y, Y_L:  out std_logic);
End mux74x151;

Architecture dataflow of mux74x151 is
Signal Y1: std_logic;
Begin
```

Multiplexers

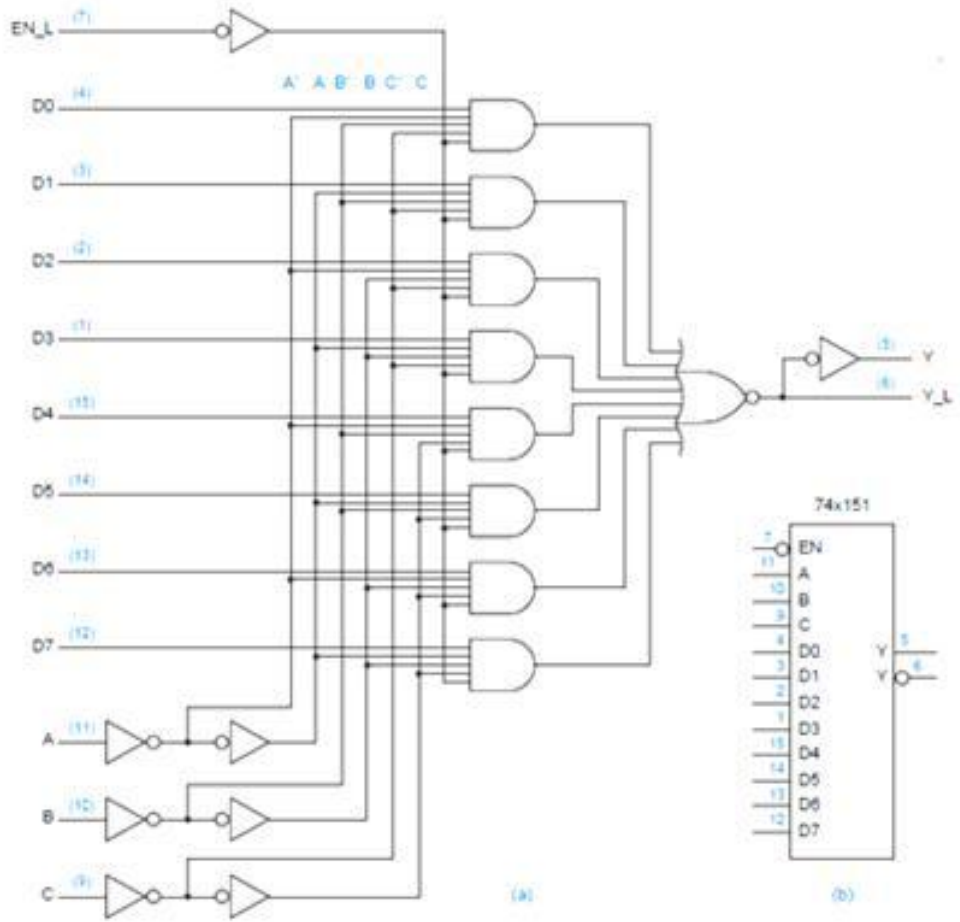
- A *multiplexer* is a digital switch—it connects data from one of *n* sources to its output.



The 74x151 8-input, 1-bit multiplexer

The *74x151* selects among eight 1-bit inputs. The enable input EN_L is active low; both active-high (Y) and active-low (Y_L) versions of the output are provided.

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′



VHDL Code for 74x151

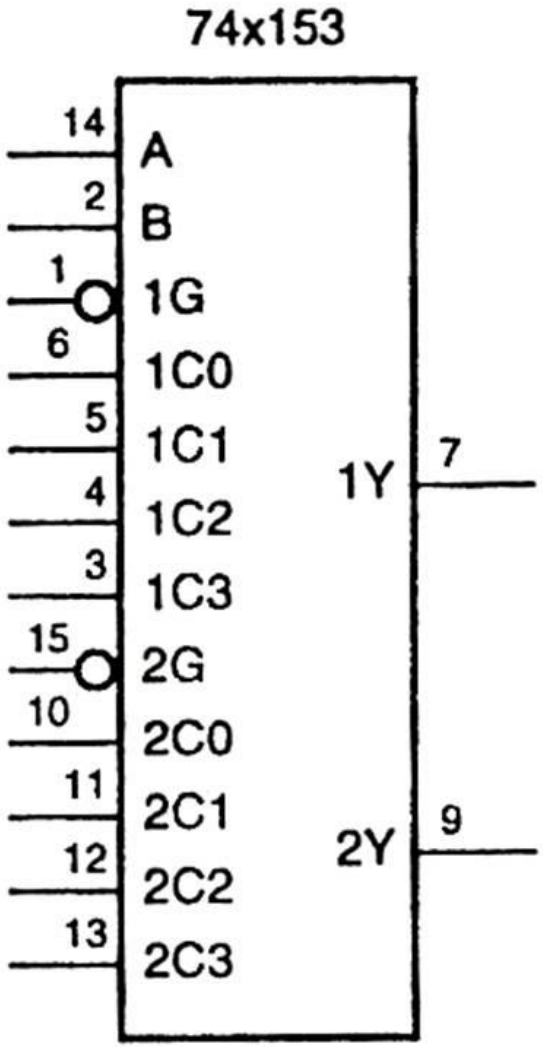
```
Library ieee;
Use ieee.std_logic_1164.all;
Entity mux74x151 is
   Port( EN_L:    in std_logic;
         S:       in std_logic_vector(2 downto 0);
         D:       in std_logic_vector(7 downto 0);
         Y, Y_L:  out std_logic);
End mux74x151;

Architecture dataflow of mux74x151 is
Signal Y1: std_logic;
Begin
```

VHDL Code for 74x151

```
with S select  Y1<=  D(7) when “111”,
                     D(6) when “110”,
                     D(5) when “101”,
                     D(4) when “100”,
                     D(3) when “011”,
                     D(2) when “010”,
                     D(1) when “001”,
                     D(0) when “000”,
                     ‘0’ when others;

    Y<=Y1 when EN_L=‘0’ else ‘0’;
    Y_L<=not y;
End dataflow;
```

74x153 4-input, 2-bit multiplexer

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



VHDL Code for 74x153

```
Library ieee;
Use ieee.std_logic_1164.all;

Entity mux74x153 is
   Port( G_L,S:   in      std_logic_vector(1 downto 0);
         C1,C2:   in      std_logic_vector(3 downto 0);
         Y:       out     std_logic_Vector(1 downto 0));
End mux74x153;

Architecture behavioral of mux74x153 is
    Signal y1: std_logic_vector(1 downto 0);
Begin

Process(G_L,S,C1,C2)
Begin
    If (G_L="00") then
          If (S="00") then      Y1(1)<= C1(0);   Y1(0)<= C2(0);
          Elsif (S="01" ) then  Y1(1)<= C1(1);   Y1(0)<= C2(1);
          Elsif (S="10" ) then  Y1(1)<= C1(2);   Y1(0)<= C2(2);
          Elsif (S="11" ) then  Y1(1)<= C1(3);   Y1(0)<= C2(3);
           End if;
```

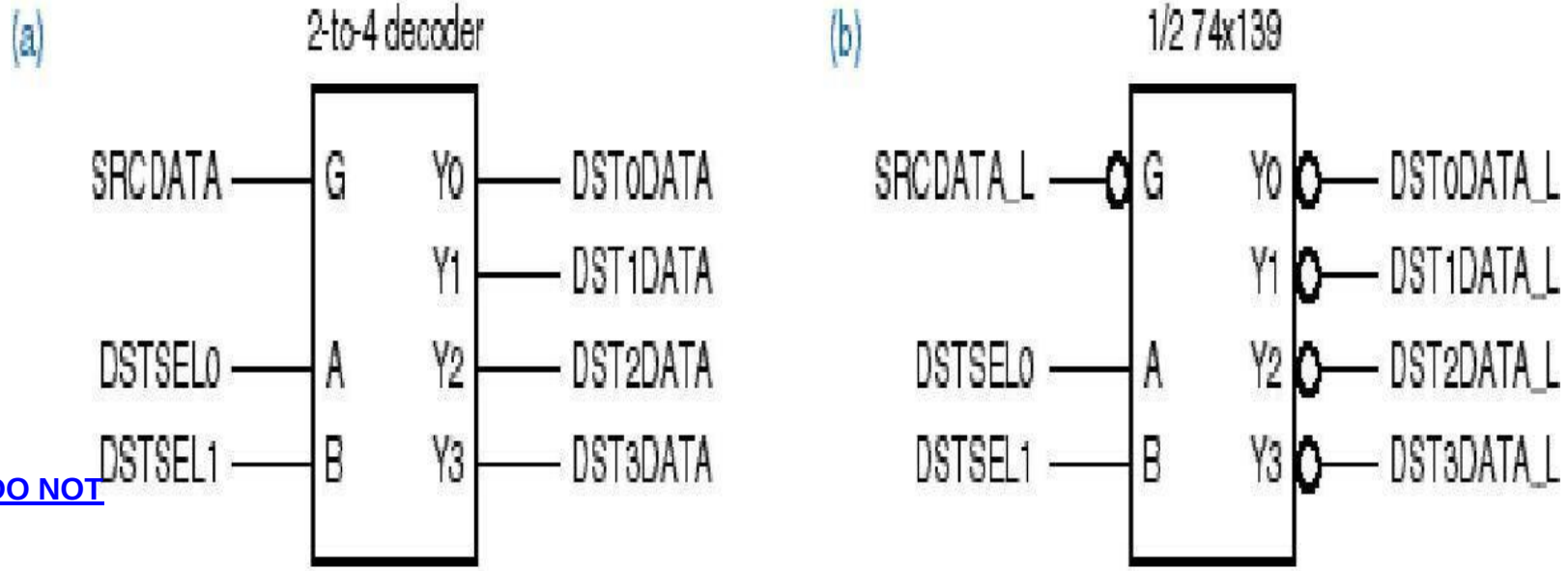
VHDL Code for 74x153

```
Els if(G_L="01" ) then
        If (S="00") then        Y1(1)<= C1(0);   Y1(0)<= '0';
        Elsif (S="01" ) then    Y1(1)<= C1(1);   Y1(0)<= '0';
        Elsif (S="10" ) then    Y1(1)<= C1(2);   Y1(0)<= '0';
        Elsif (S="11" ) then    Y1(1)<= C1(3);   Y1(0)<= '0';
        End if;

Els if(G_L="10" ) then
        If (S="00") then        Y1(1)<= '0';     Y1(0)<= C2(0);
        Elsif (S="01" ) then    Y1(1)<= '0';     Y1(0)<= C2(1);
        Elsif (S="10" ) then    Y1(1)<= '0';     Y1(0)<= C2(2);
        Elsif (S="11" ) then    Y1(1)<= '0';     Y1(0)<= C2(3);
        End if;

Elsif (G_L="11") then           Y1="00";

end if;
End process;
End behavioral;
```

Demultiplexers

- A *demultiplexer* can be used to route the bus data to one of *m* destinations.
- The function of a demultiplexer is just the inverse of a multiplexer's. For example, a 1-bit, *n*-output demultiplexer has one data input and *s* inputs to select one of *n* 2*s* data outputs.

Demultiplexers



DO NOT

The decoder's enable input is connected to the data line, and its select inputs determine which of its output lines is driven with the data bit. The remaining output lines are negated.

Binary to BCD converter

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

K-map simplification

For B_0

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

$B_0 = A$

For B_1

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

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

For B_2

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

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

For B_3

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

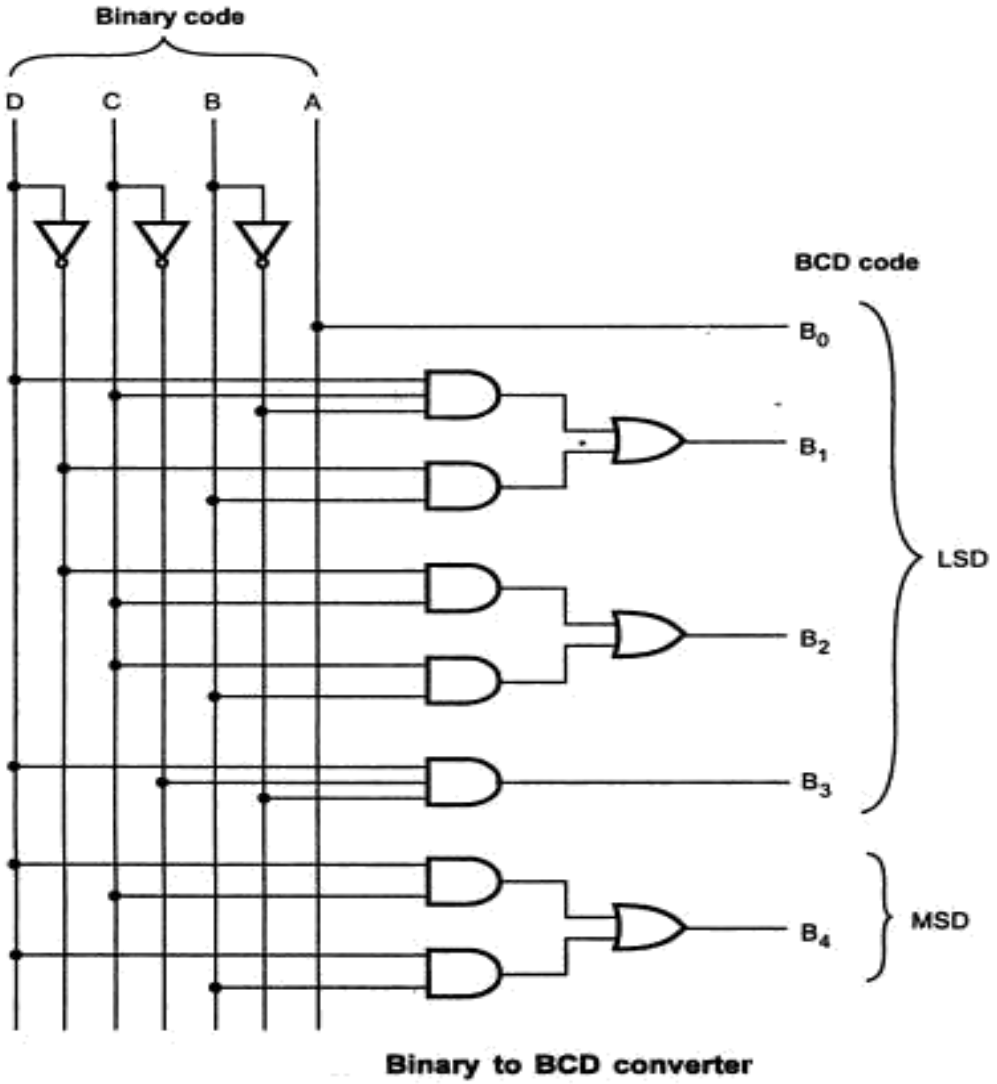
$B_3 = D\overline{C}\,\overline{B}$

For B_4

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

$B_4 = DC + DB$

Binary to BCD converter

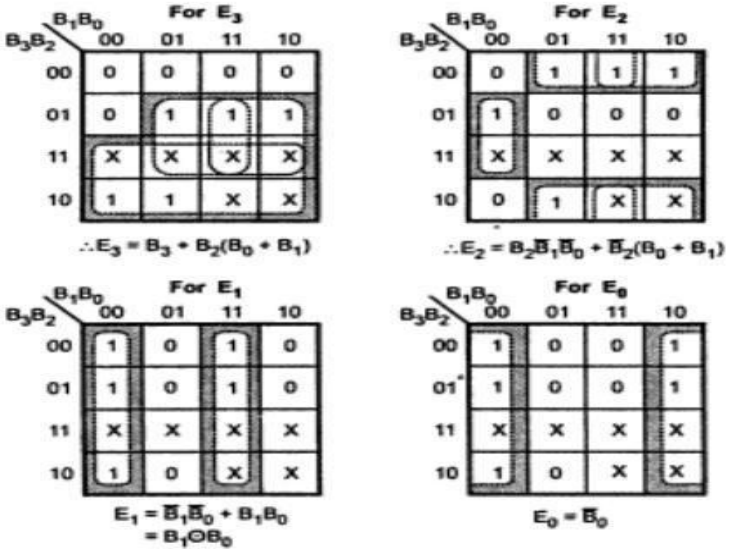


Binary to BCD converter

BCD to Excess-3 code converter

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

K-map simplification



BCD to Excess-3 code converter

Logic diagram

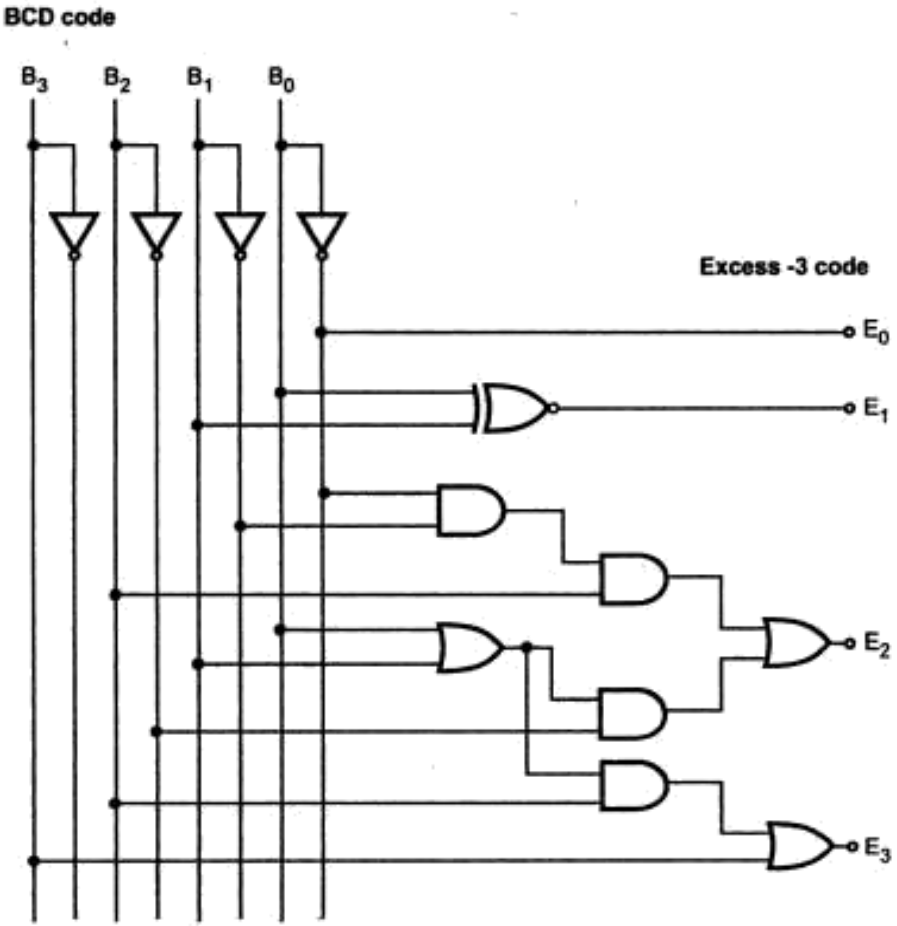


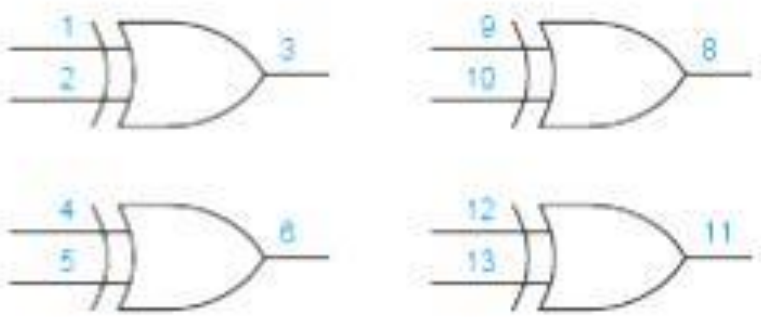
Fig. BCD to Excess-3 code converter

Exclusive OR Gates

Truth Table for XOR and XNOR functions

X	Y	$X \oplus Y$ (XOR)	$(X \oplus Y)'$ (XNOR)
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1

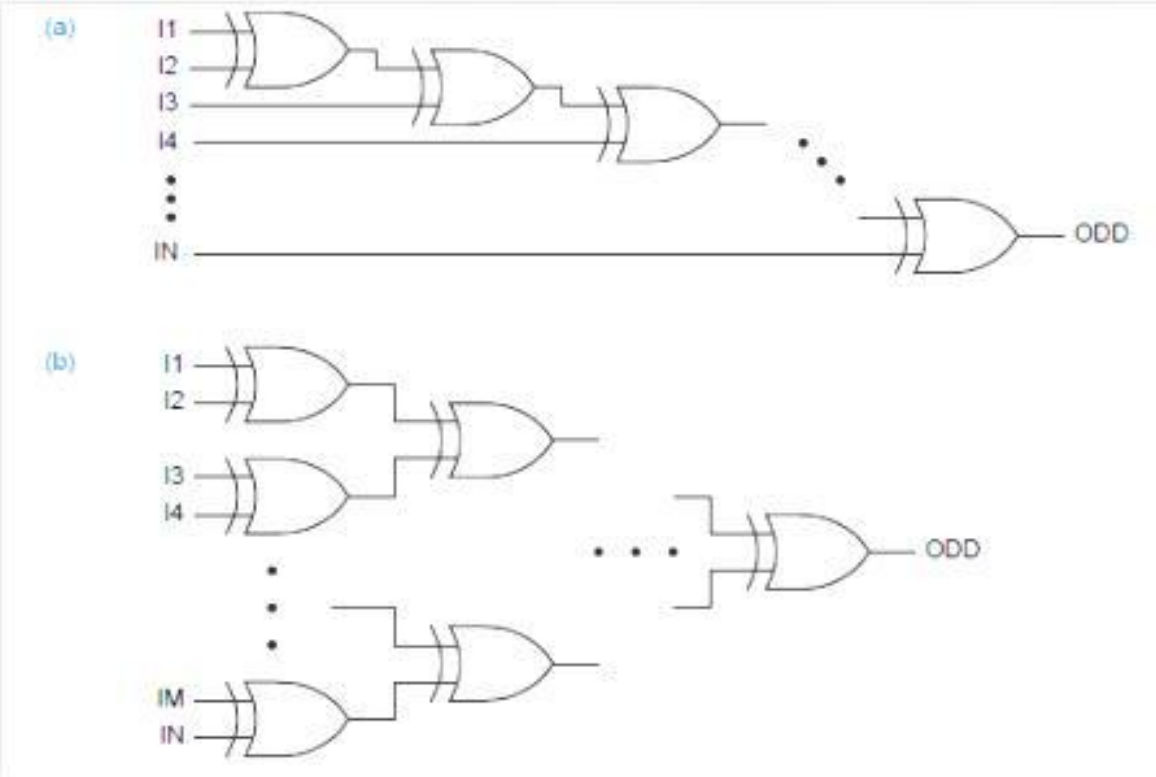
Pin outs of the 74x86 quadruple 2-input Exclusive OR Gate



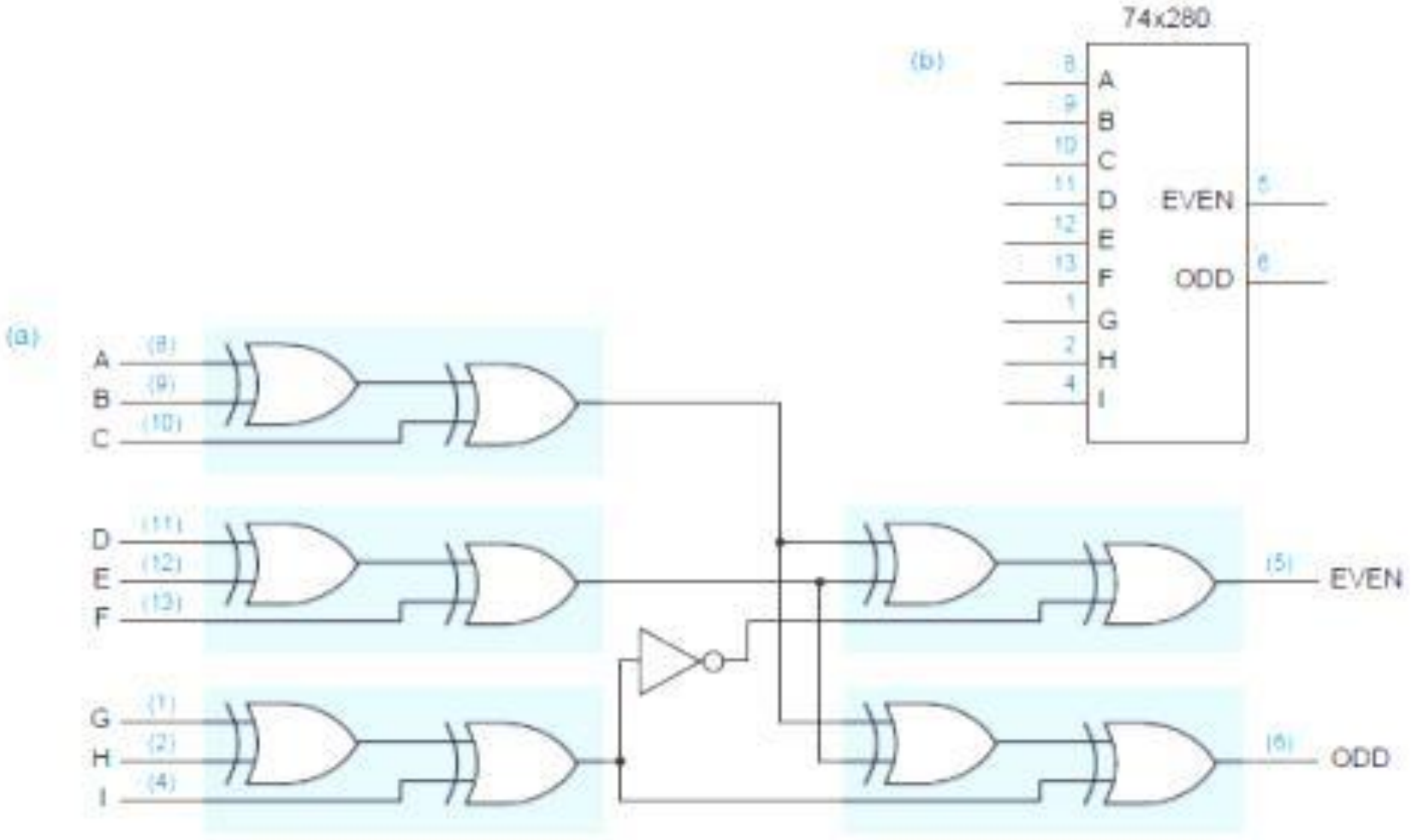
Parity Circuits

n XOR gates may be cascaded to form a circuit with *n*+1 inputs and a single output. This is called an *odd-parity circuit*, because its output is if an odd number of its inputs are 1.

The circuit in (b) is also an odd parity circuit, but it's faster because its gates are arranged in a tree-like structure



74x280 9-Bit Parity Generator



VHDL Code for 74x280

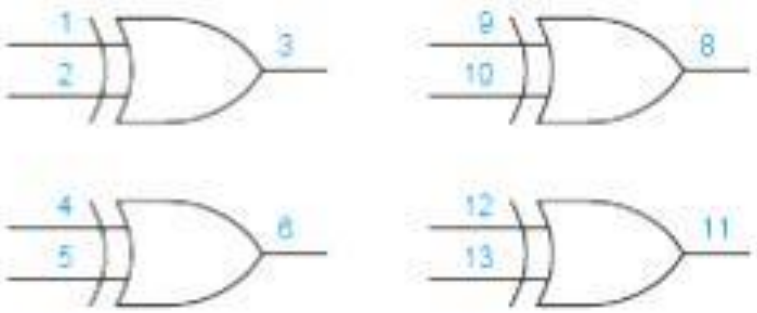
```
library IEEE;
use IEEE.std_logic_1164.all;

entity parity74x280 is
        port (  I:              in   STD_LOGIC_VECTOR (1 to 9);
                EVEN, ODD:      out  STD_LOGIC);
end parity74x280;

architecture structural of parity74x280 is
    component vxor3
    port (A, B, C: in STD_LOGIC; Y: out STD_LOGIC);
    end component;
    signal Y1, Y2, Y3, Y3N: STD_LOGIC;
begin

    U1: vxor3 port map (I(1), I(2), I(3), Y1);
    U2: vxor3 port map (I(4), I(5), I(6), Y2);
    U3: vxor3 port map (I(7), I(8), I(9), Y3);
    Y3N <= not Y3;
    U4: vxor3 port map (Y1, Y2, Y3, ODD);
    U5: vxor3 port map (Y1, Y2, Y3N, EVEN);
end Structural;
```

Exclusive OR Gates

Truth Table for XOR and XNOR functions

X	Y	$X \oplus Y$ (XOR)	$(X \oplus Y)'$ (XNOR)
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1

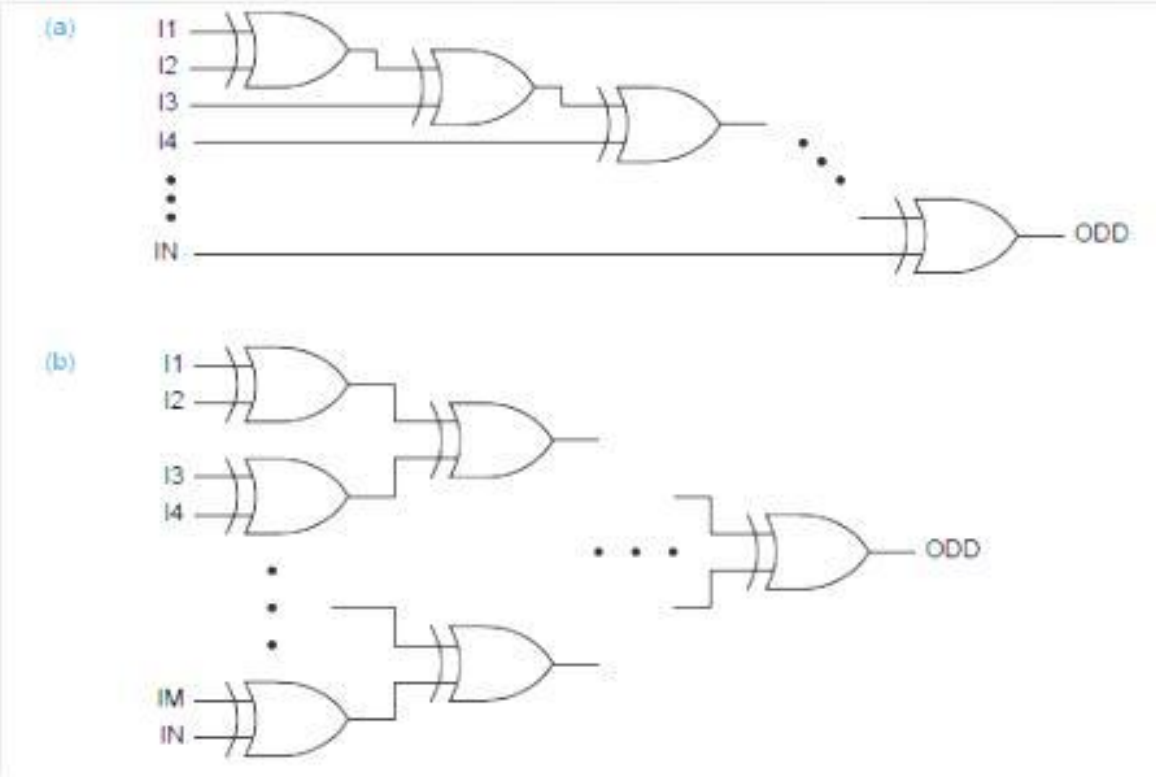
Pin outs of the 74x86 quadruple 2-input Exclusive OR Gate



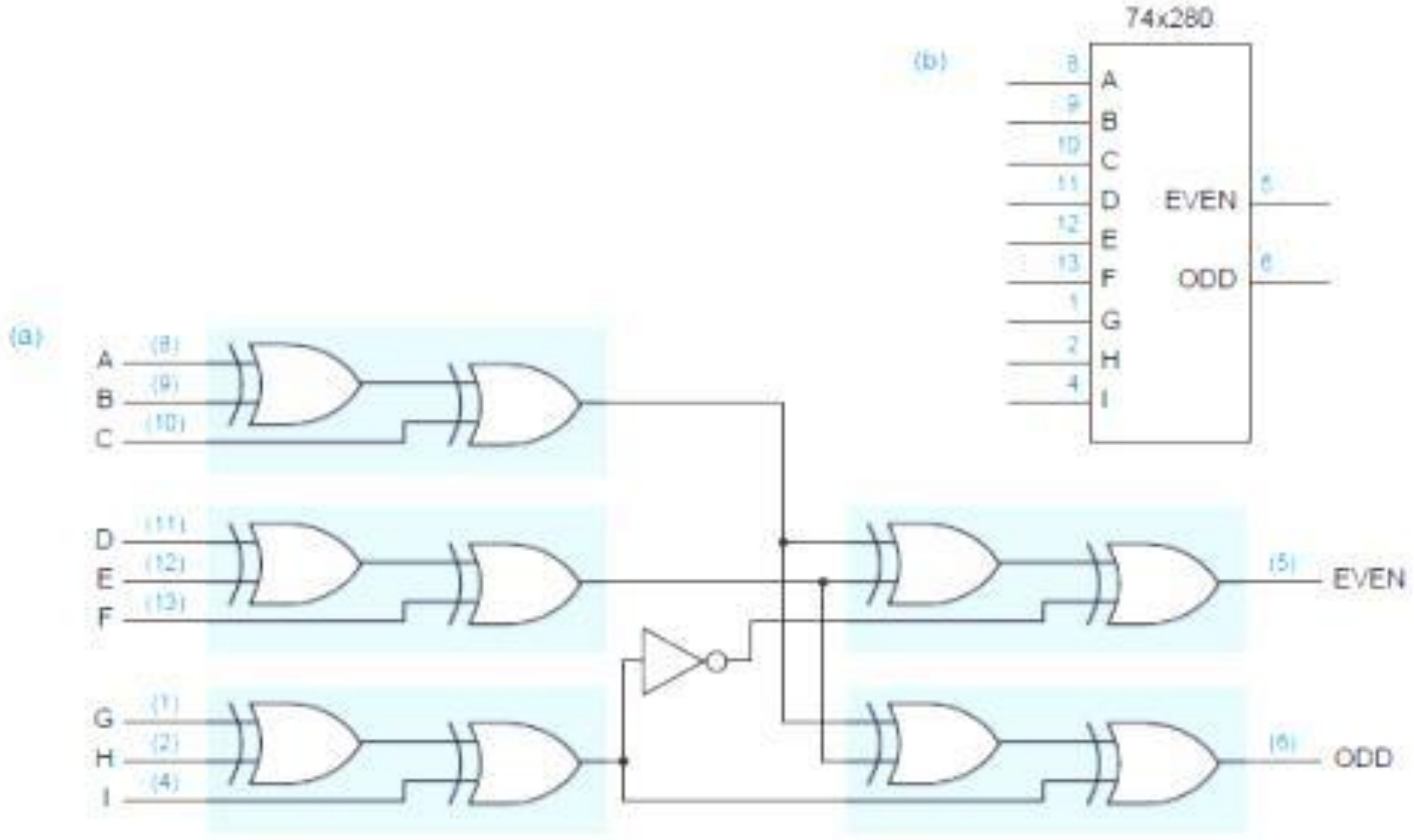
Parity Circuits

n XOR gates may be cascaded to form a circuit with *n*+1 inputs and a single output. This is called an *odd-parity circuit*, because its output is if an odd number of its inputs are 1.

The circuit in (b) is also an odd parity circuit, but it's faster because its gates are arranged in a tree-like structure



74x280 9-Bit Parity Generator



VHDL Code for 74x280

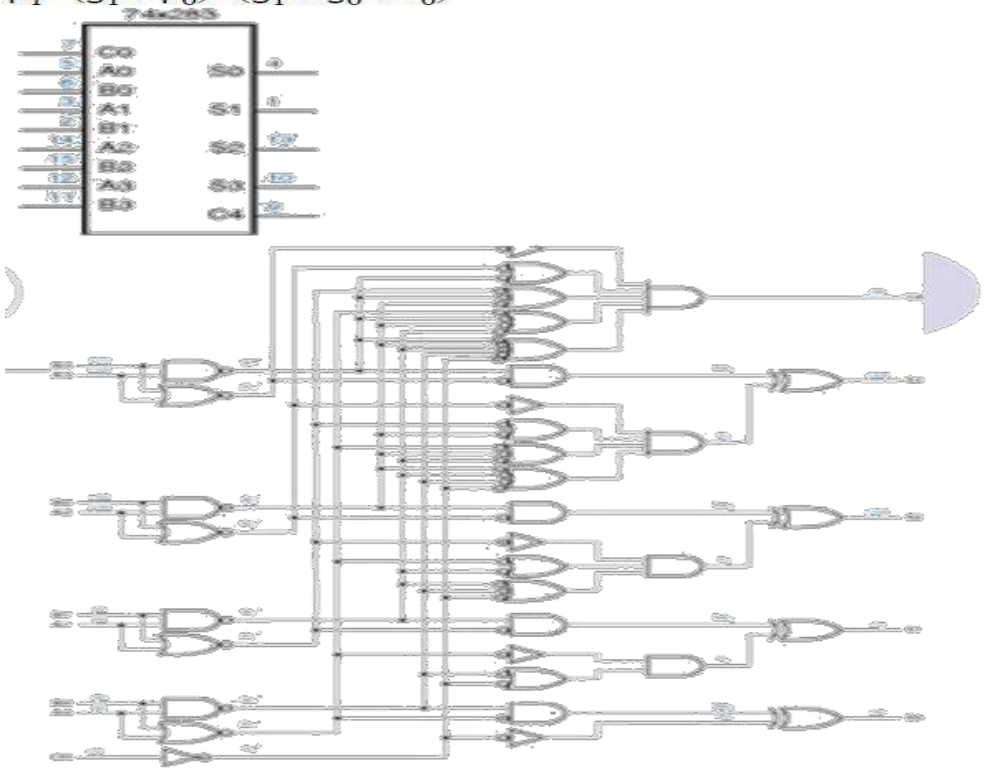
```
library IEEE;
use IEEE.std_logic_1164.all;

entity parity74x280 is
        port (  I:              in   STD_LOGIC_VECTOR (1 to 9);
                EVEN, ODD:      out  STD_LOGIC);
end parity74x280;

architecture structural of parity74x280 is
    component vxor3
    port (A, B, C: in STD_LOGIC; Y: out STD_LOGIC);
    end component;
    signal Y1, Y2, Y3, Y3N: STD_LOGIC;
begin

    U1: vxor3 port map (I(1), I(2), I(3), Y1);
    U2: vxor3 port map (I(4), I(5), I(6), Y2);
    U3: vxor3 port map (I(7), I(8), I(9), Y3);
    Y3N <= not Y3;
    U4: vxor3 port map (Y1, Y2, Y3, ODD);
    U5: vxor3 port map (Y1, Y2, Y3N, EVEN);
end Structural;
```

Carry look ahead technique

- The *74x283* is a 4-bit binary adder that forms its sum and carry outputs with just a few levels of logic, using the carry lookahead technique.
- The older *74x83* is identical except for its pinout, which has nonstandard locations for power and ground.

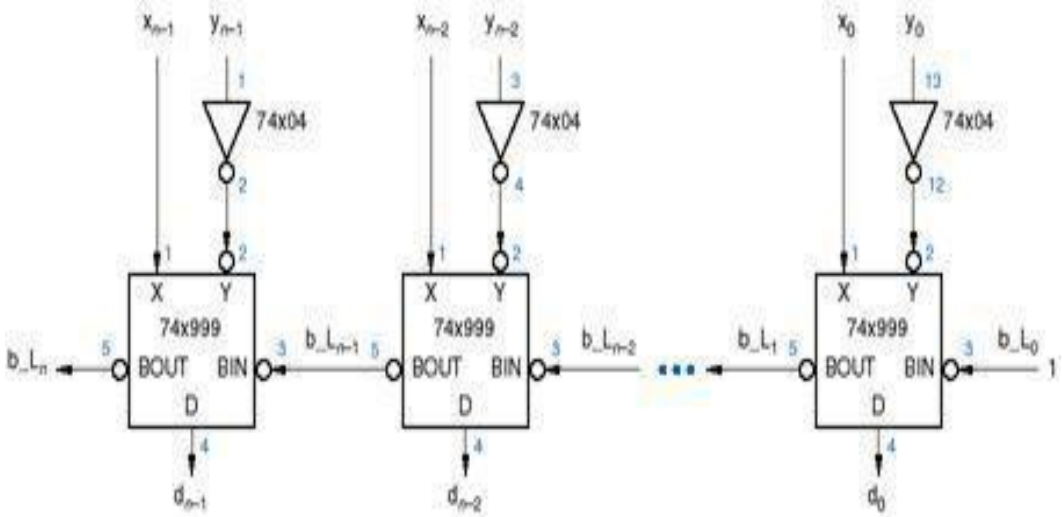
Carry lookahead technique

$$c_1 = p_0 \cdot (g_0 + c_0)$$
$$c_2 = p_1 \cdot (g_1 + c_1)$$
$$= p_1 \cdot (g_1 + p_0 \cdot (g_0 + c_0))$$
$$= p_1 \cdot (g_1 + p_0) \cdot (g_1 + g_0 + c_0)$$

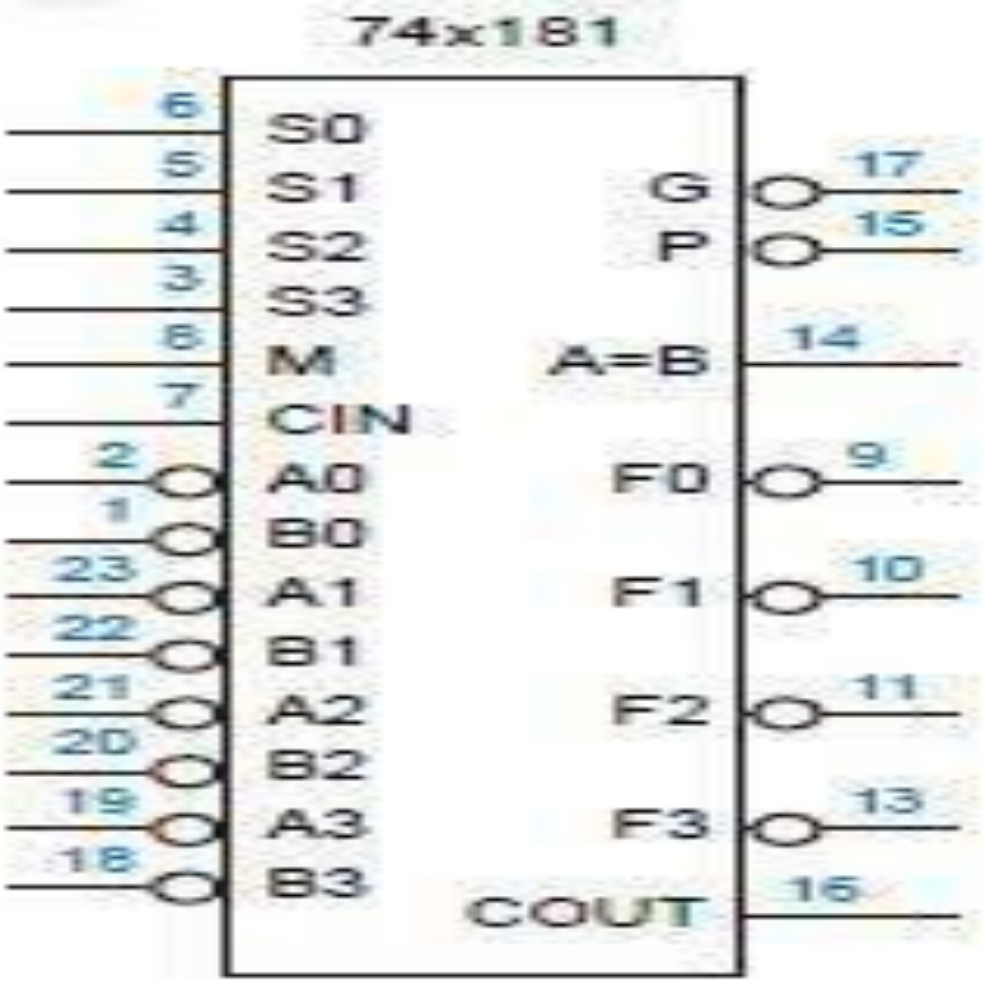


Subtractors

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



Arithmetic and Logic Units

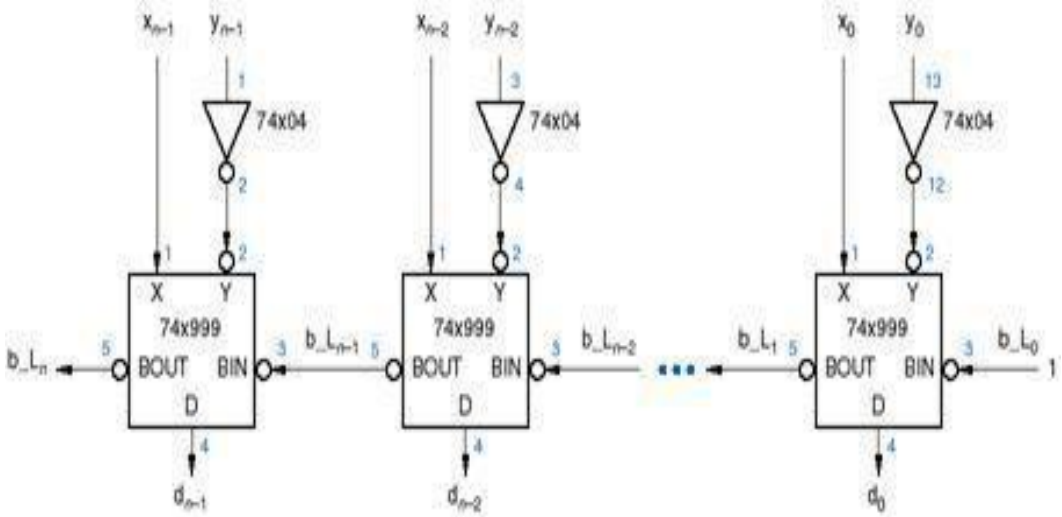


Arithmetic and Logic Unit

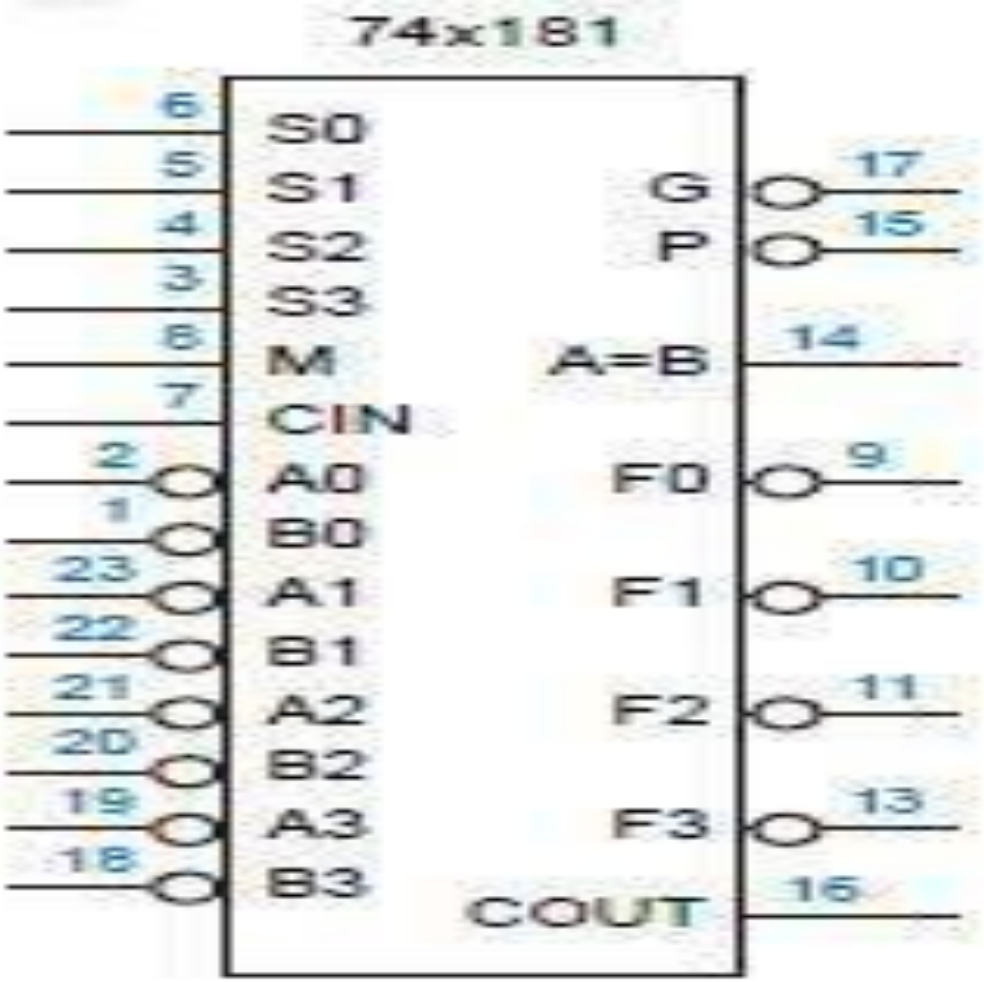
Inputs				Function	
S3	S2	S1	S0	M = 0 (arithmetic)	M = 1 (logic)
0	0	0	0	F = A minus 1 plus CIN	F = A′
0	0	0	1	F = A · B minus 1 plus CIN	F = A′ + B′
0	0	1	0	F = A · B′ minus 1 plus CIN	F = A′ + B
0	0	1	1	F = 1111 plus CIN	F = 1111
0	1	0	0	F = A plus (A + B′) plus CIN	F = A′ · B′
0	1	0	1	F = A · B plus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

Subtractors

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



Arithmetic and Logic Units



Arithmetic and Logic Unit

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

Combinational Multipliers

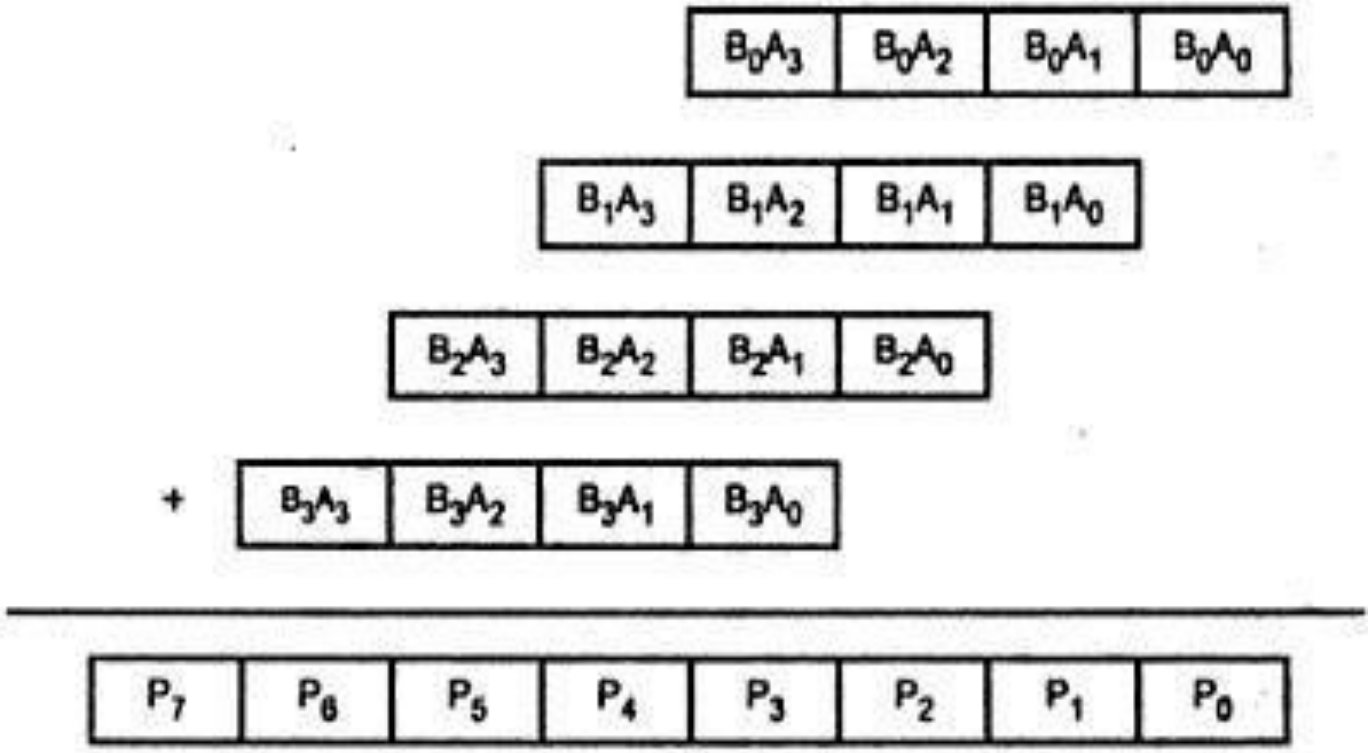
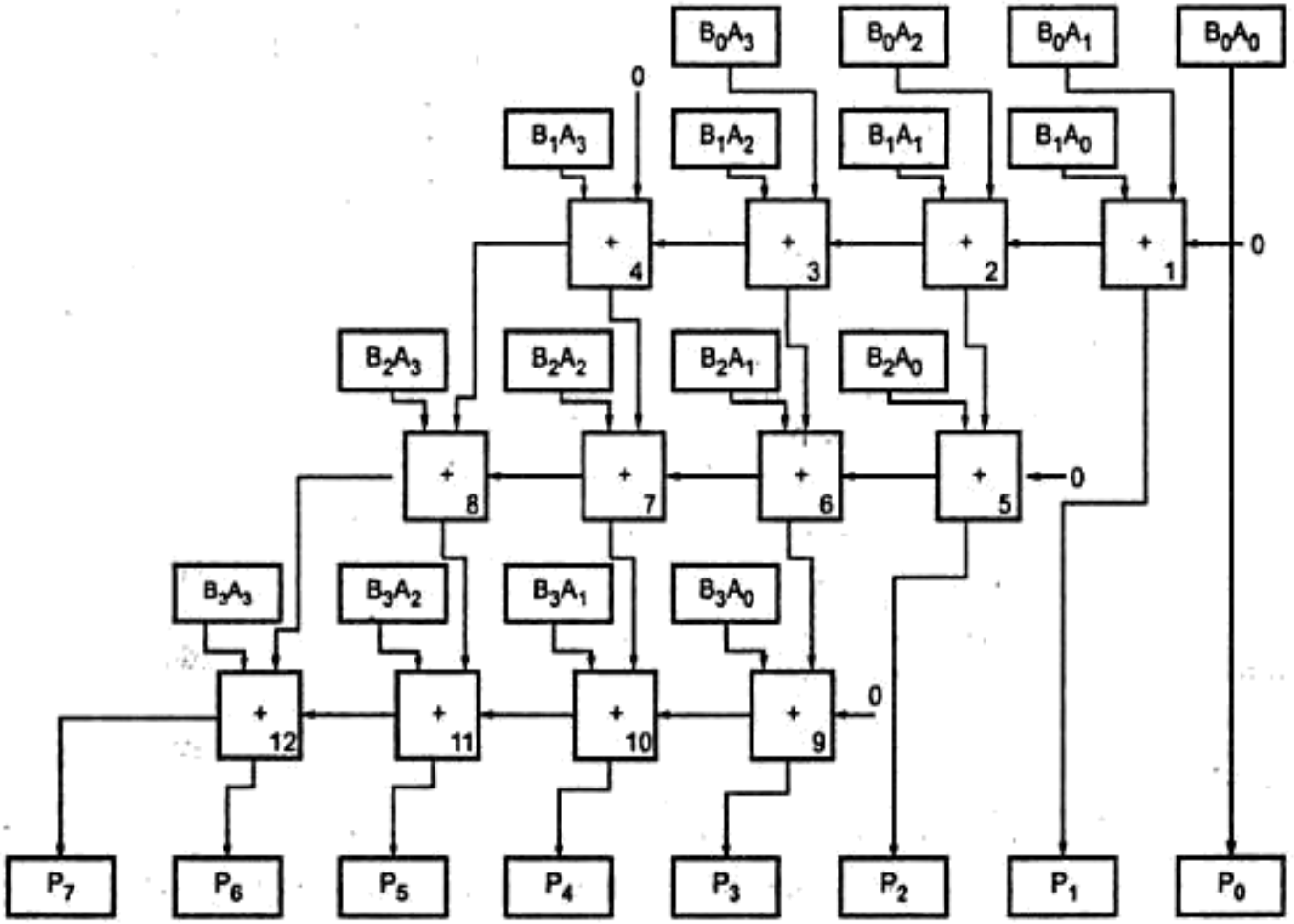


Fig. Multiplication process of 4 × 4 multiplier

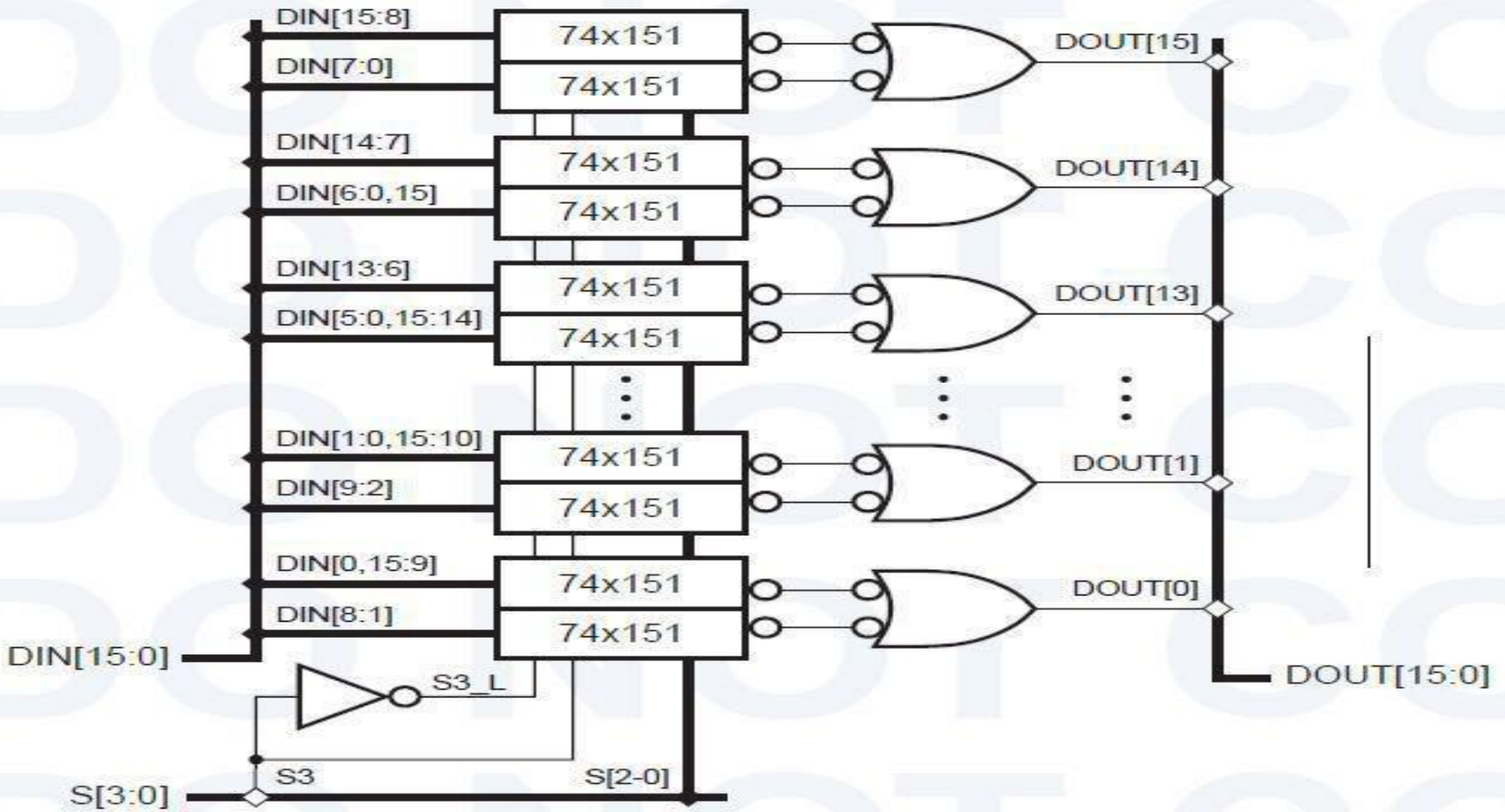
4X4 combinational multiplier with carry save



Barrel shifter

- A *barrel shifter* is a combinational logic circuit with *n* data inputs, *n* data outputs, and a set of control inputs that specify how to shift the data between input and output. A barrel shifter

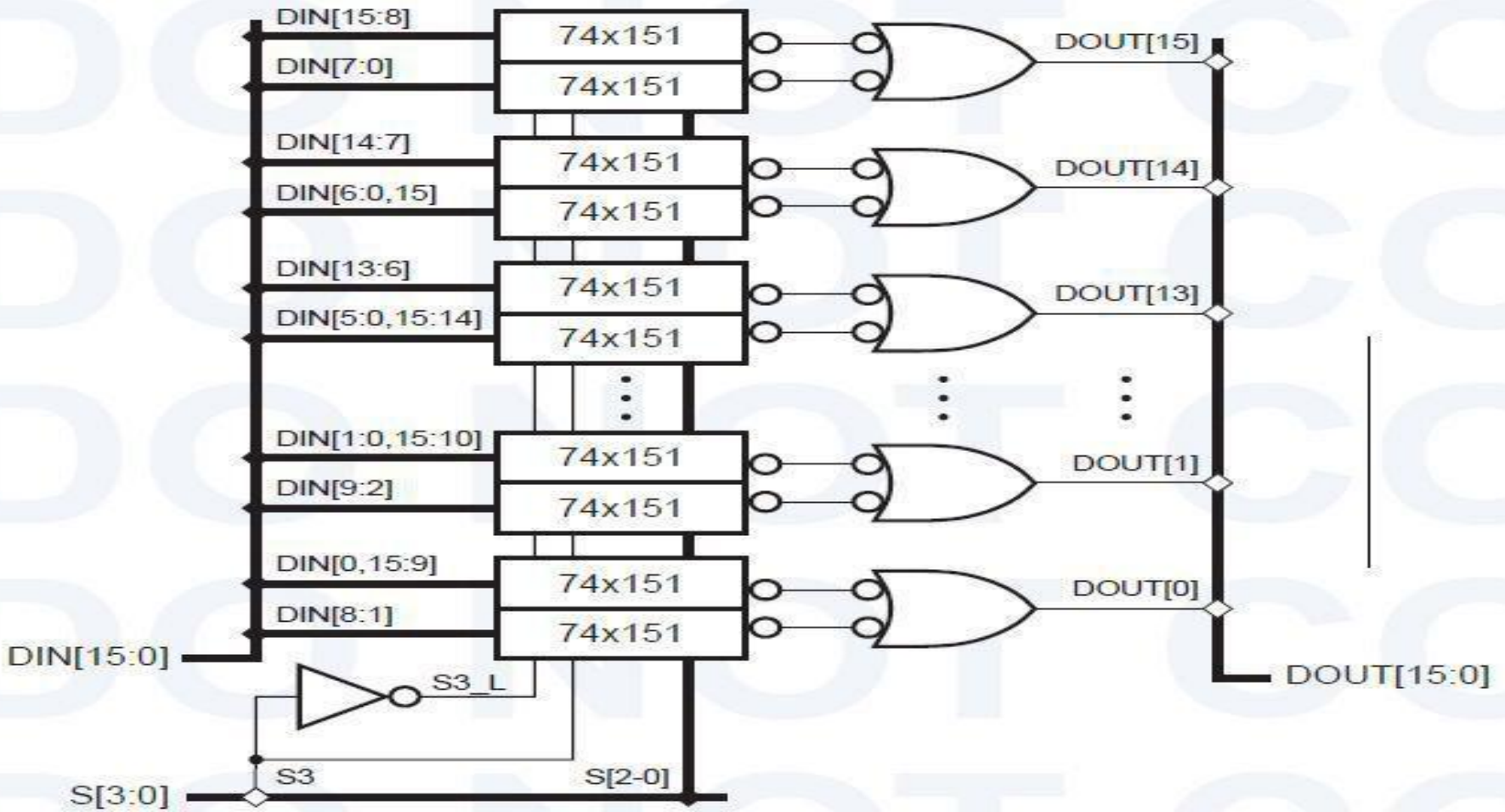
Barrel shifter



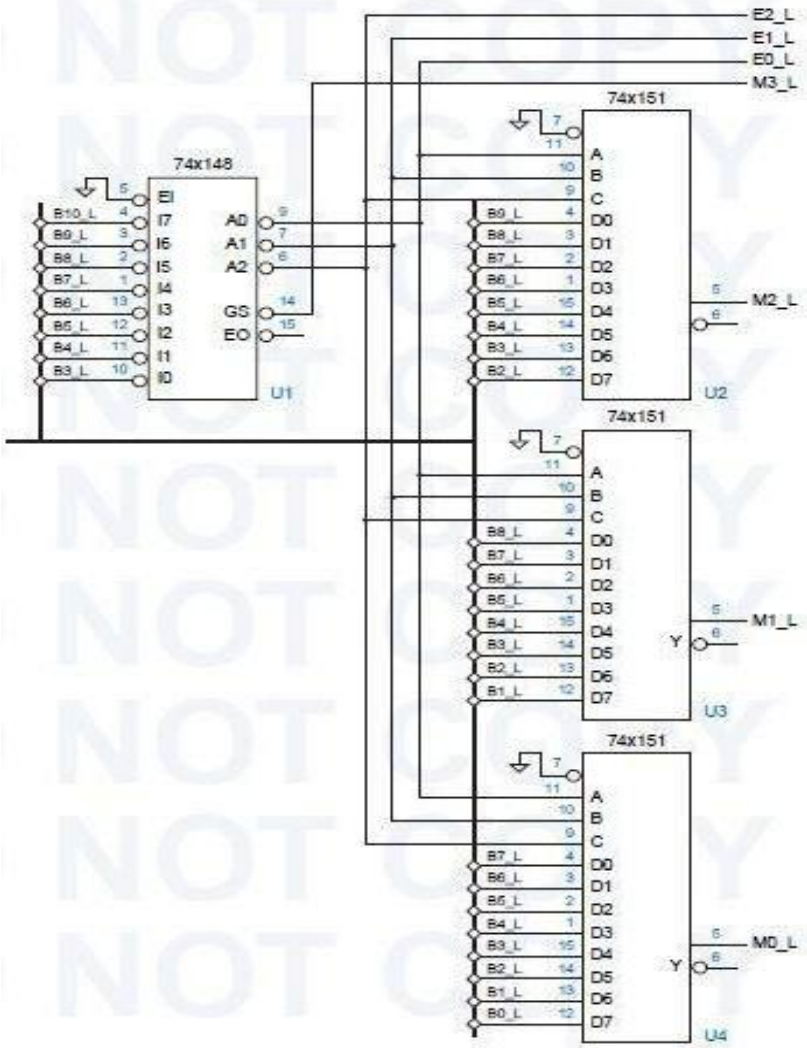
Barrel shifter

- A *barrel shifter* is a combinational logic circuit with *n* data inputs, *n* data outputs, and a set of control inputs that specify how to shift the data between input and output. A barrel shifter

Barrel shifter



Floating-point encoder



Cascading Comparators

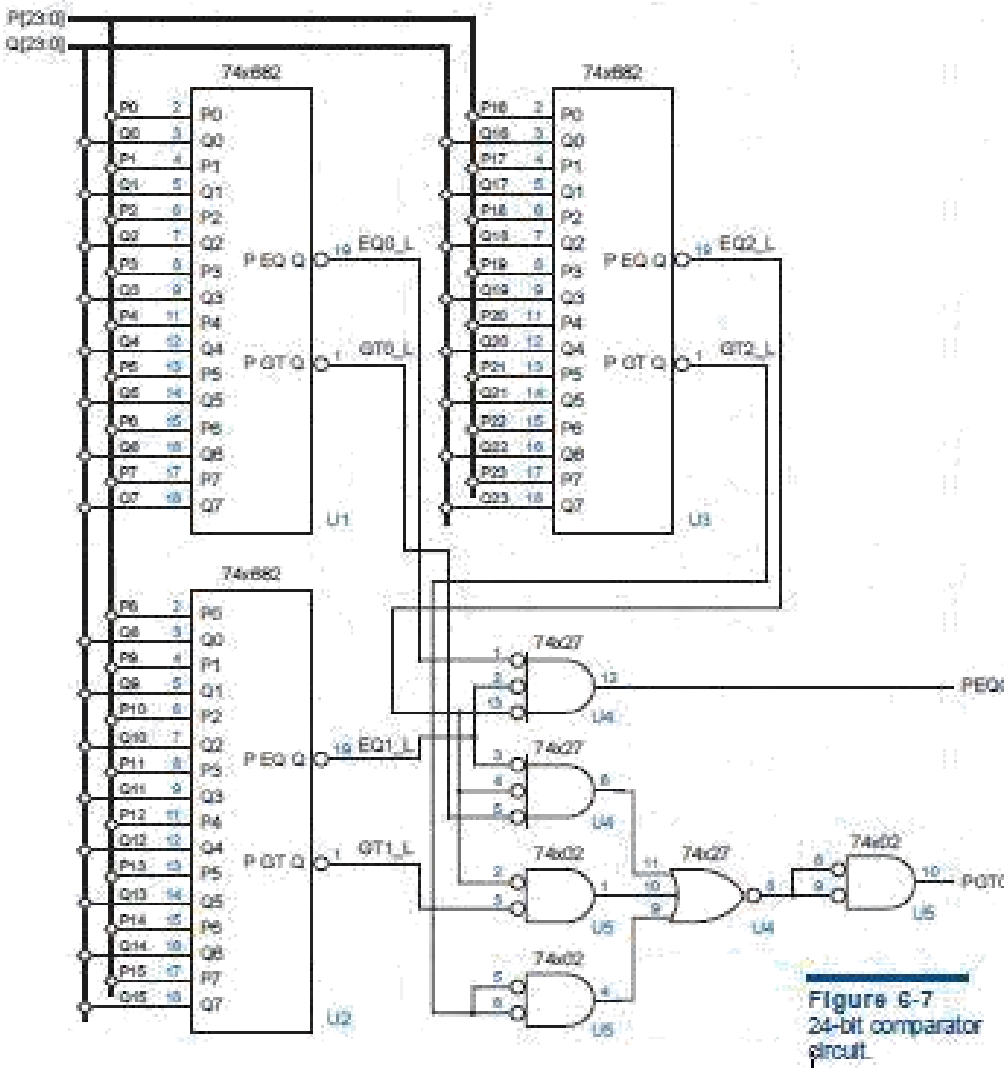
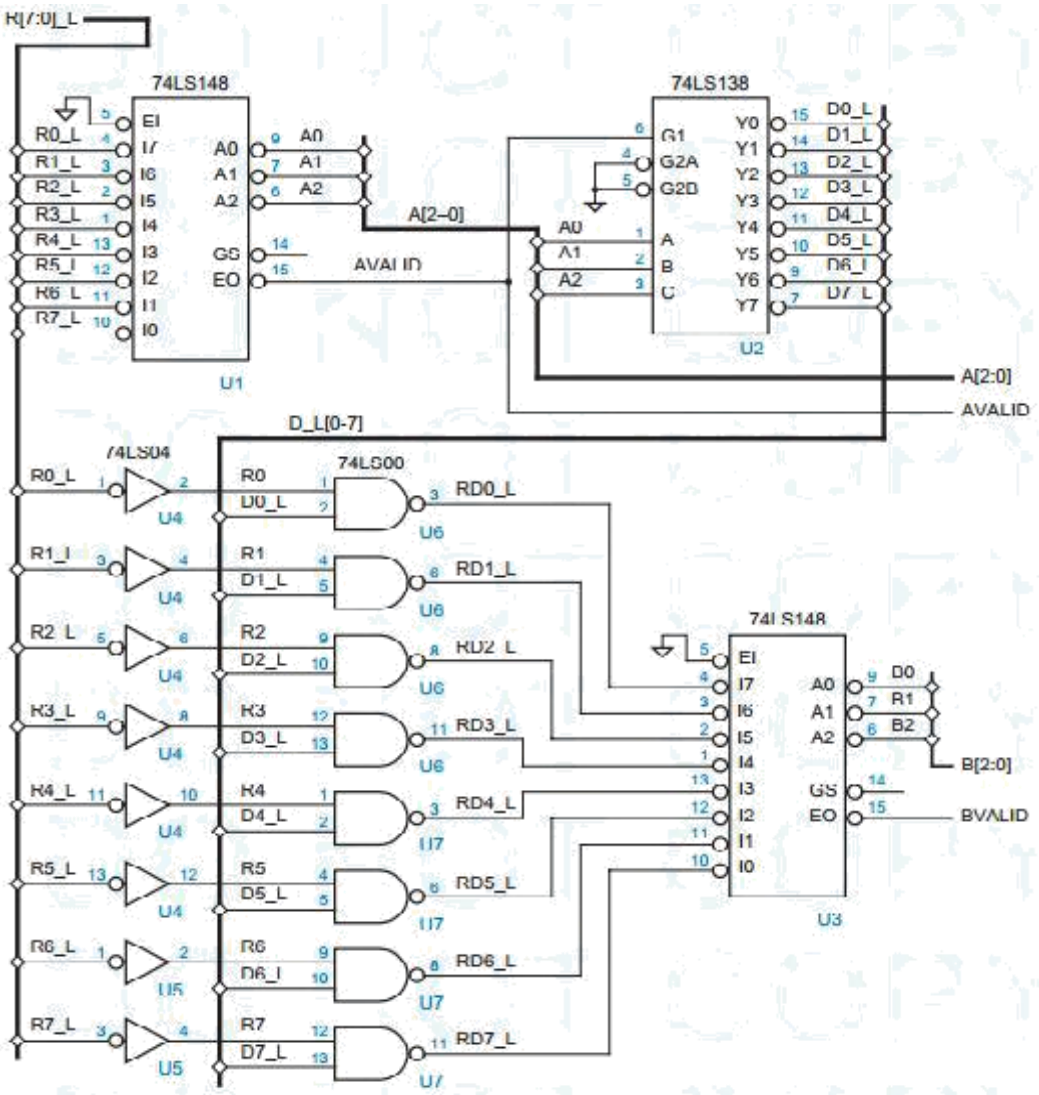


Figure 6-7
24-bit comparator circuit

Dual parity encoder

- A priority encoder that identifies not only the highest but also the second-highest priority asserted signal among a set of eight request inputs.

Dual parity encoder



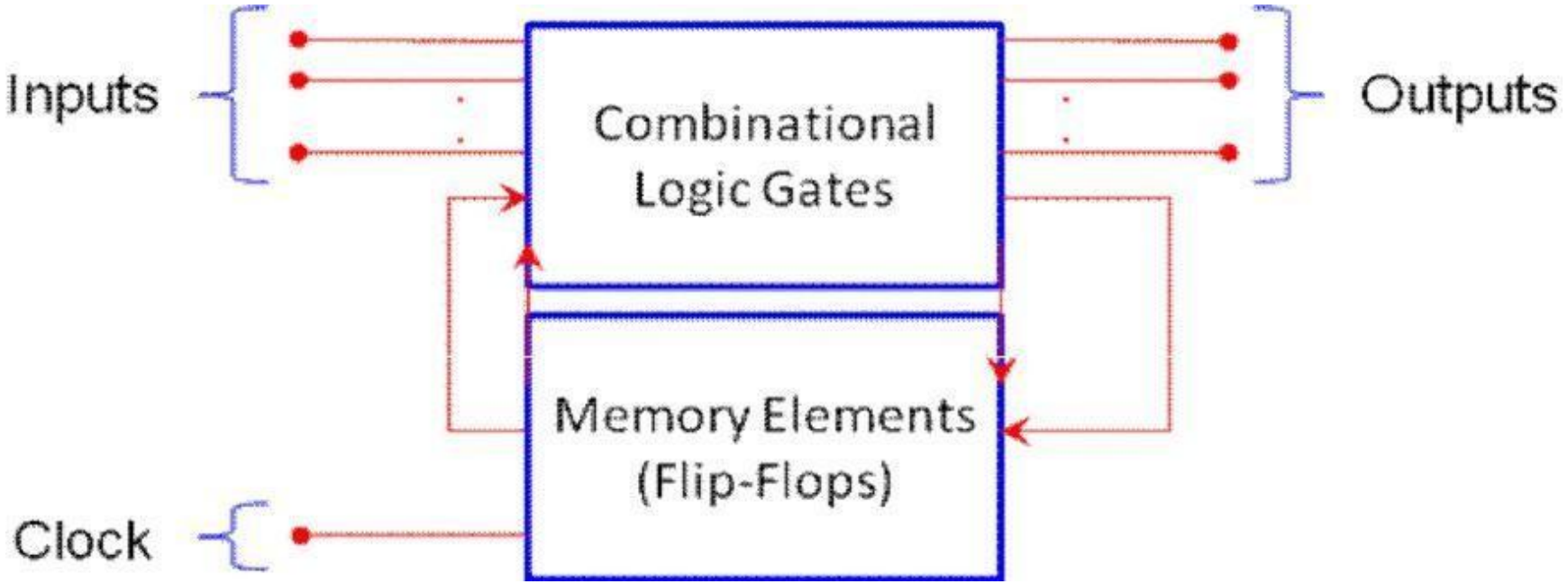
UNIT – IV

SEQUENTIAL LOGIC DESIGN

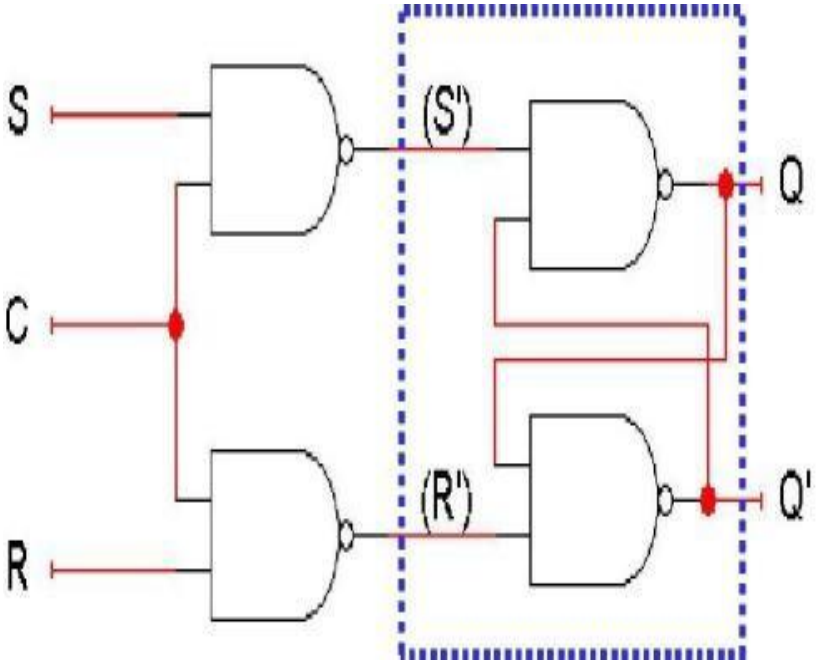
Sequential Circuits

- *Latches and flip-flops* (FFs) are the basic building blocks of sequential circuits.
  - latch: bistable memory device with level sensitive triggering (no clock), watches all of its inputs continuously and changes its outputs at any time, independent of a clocking signal.
  - flip-flop: bistable memory device with edge-triggering (with clock), samples its inputs, and changes its output only at times determined by a clocking signal.

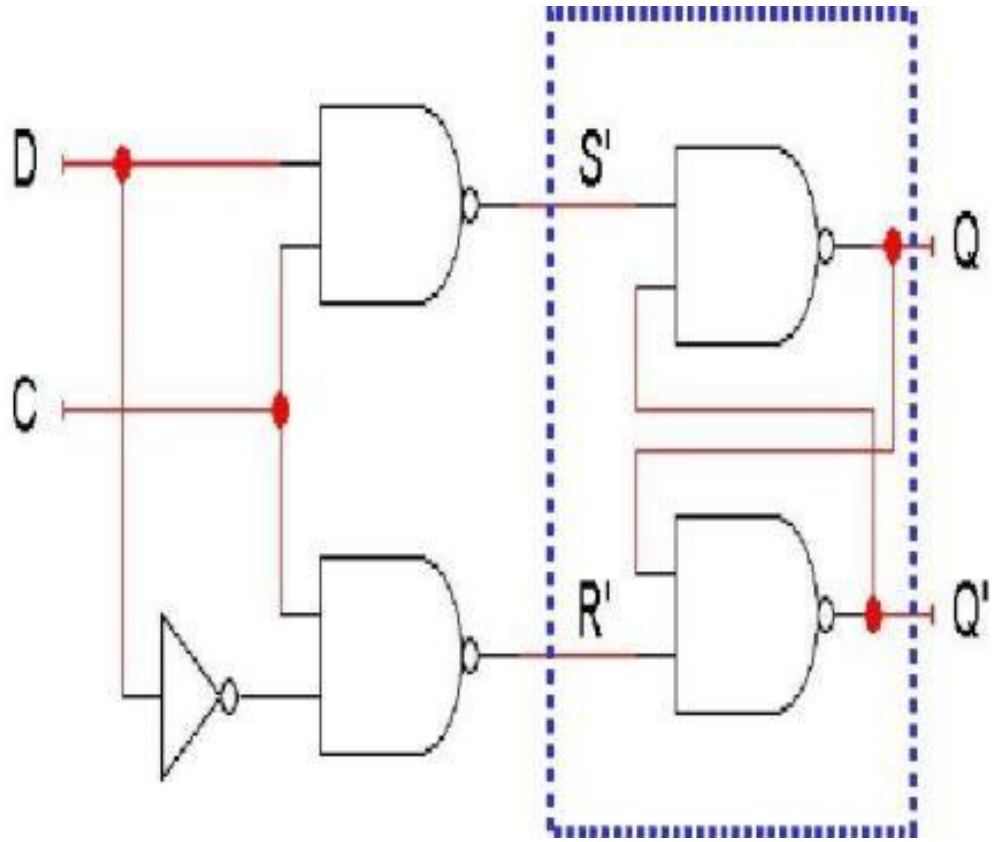
Sequential Circuit



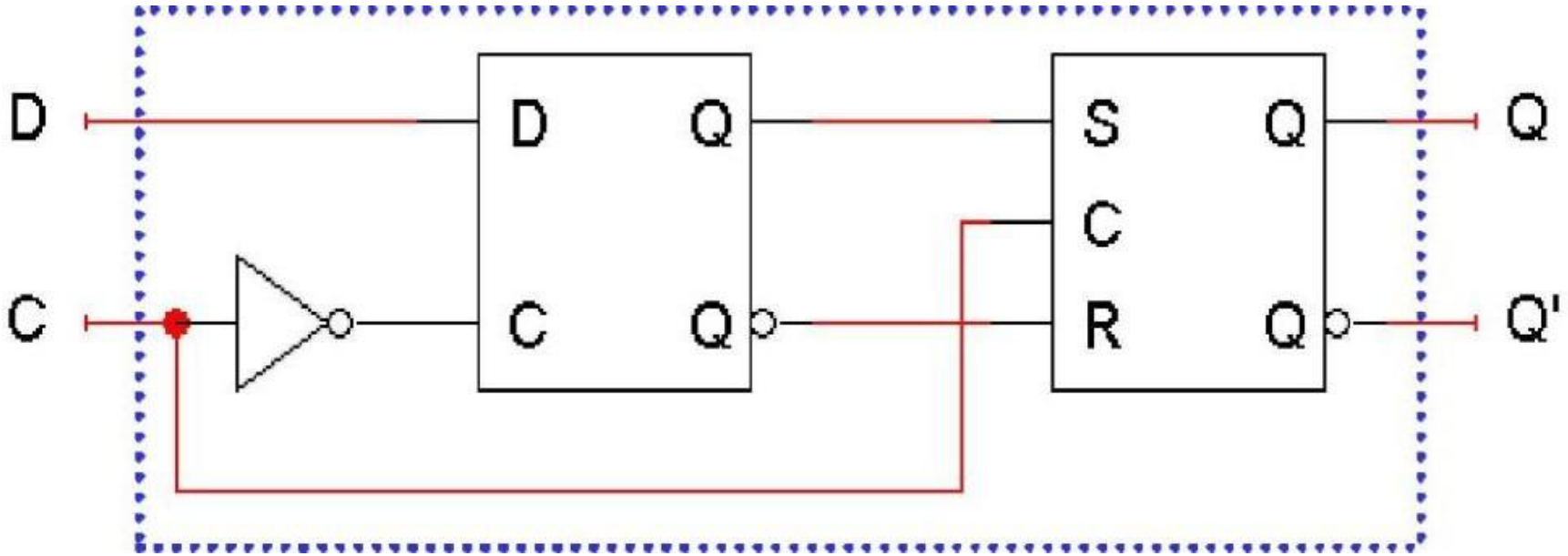
SR Latch



D latch



D flip-flop



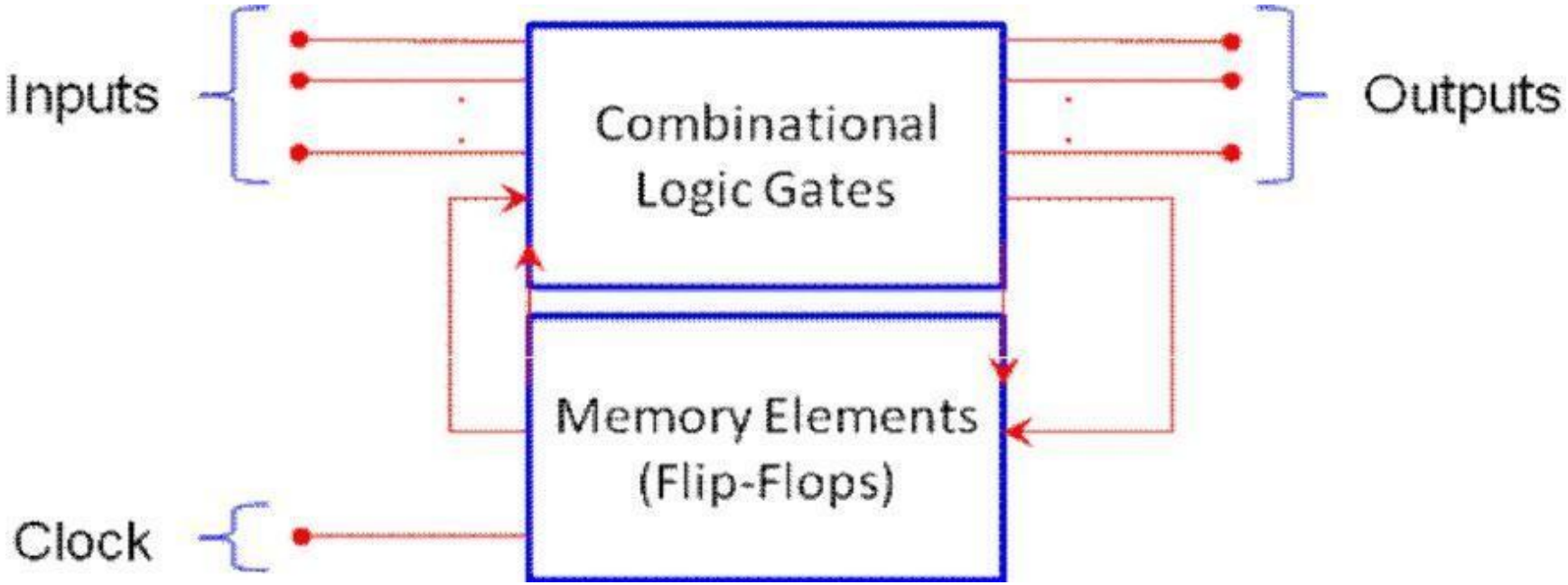
Flip-Flop Vs. Latch

- The primary difference between a D flip-flop and D latch is the EN/CLOCK input.
- The flip-flop's CLOCK input is edge sensitive, meaning the flip-flop's output changes on the edge (rising or falling) of the CLOCK input.
- The latch's EN input is level sensitive, meaning the latch's output changes on the level (high or low) of the EN input.

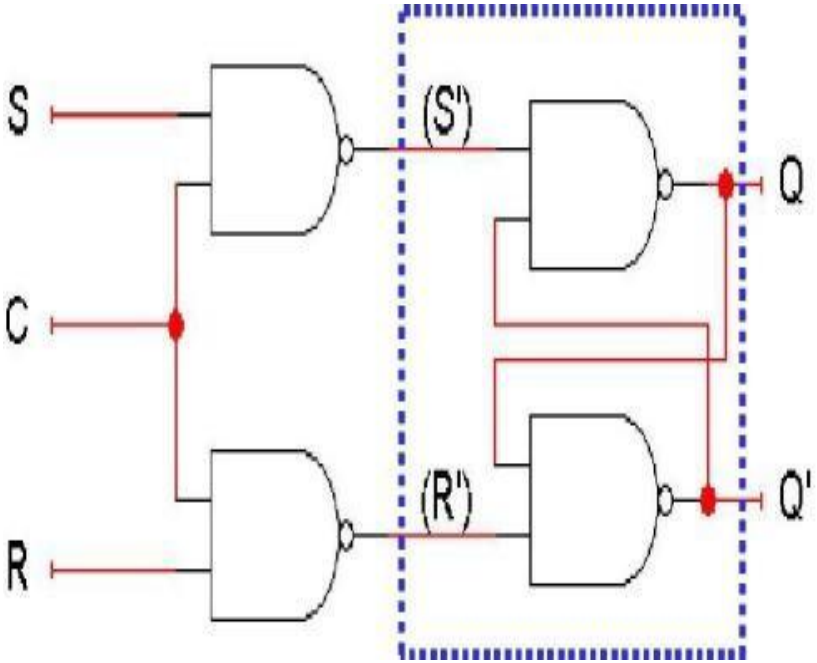
Sequential Circuits

- *Latches and flip-flops* (FFs) are the basic building blocks of sequential circuits.
  - latch: bistable memory device with level sensitive triggering (no clock), watches all of its inputs continuously and changes its outputs at any time, independent of a clocking signal.
  - flip-flop: bistable memory device with edge-triggering (with clock), samples its inputs, and changes its output only at times determined by a clocking signal.

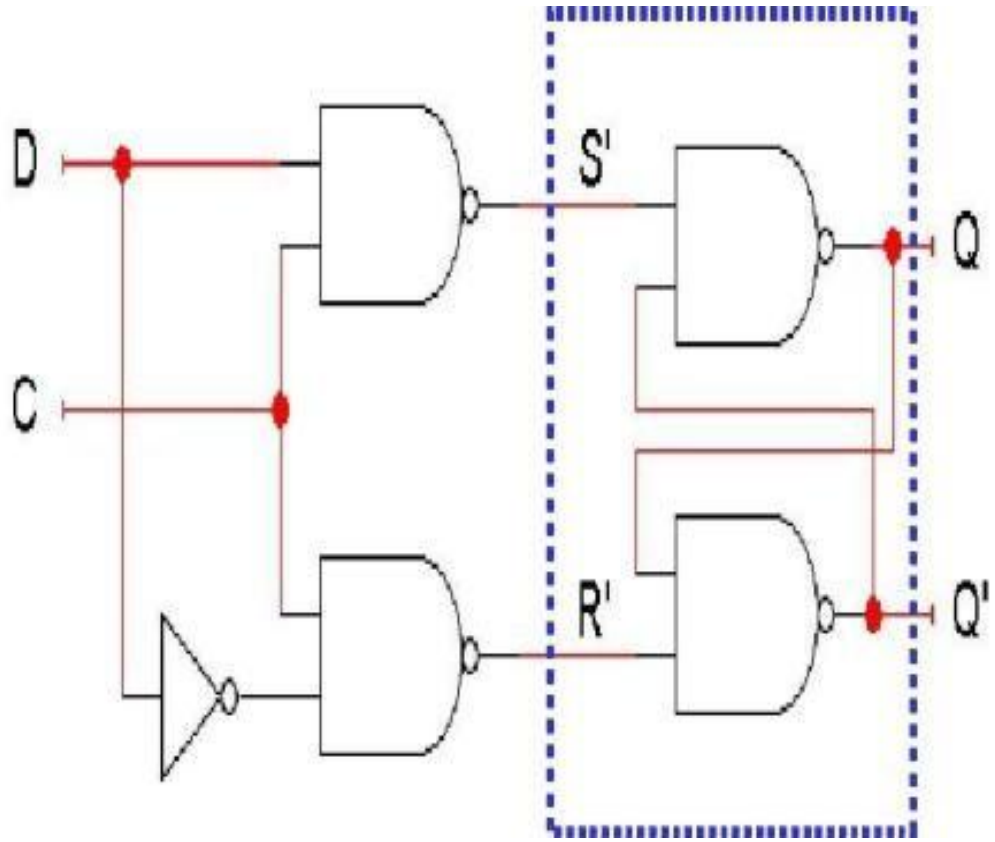
Sequential Circuit



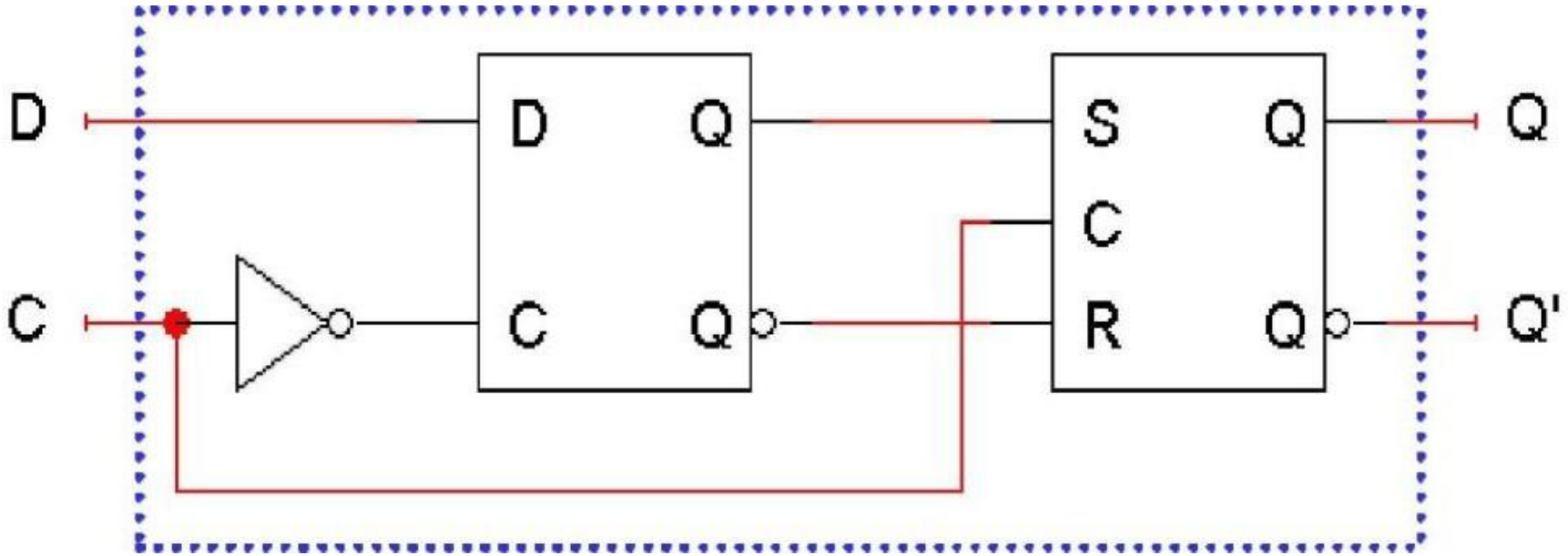
SR Latch



D latch



D flip-flop

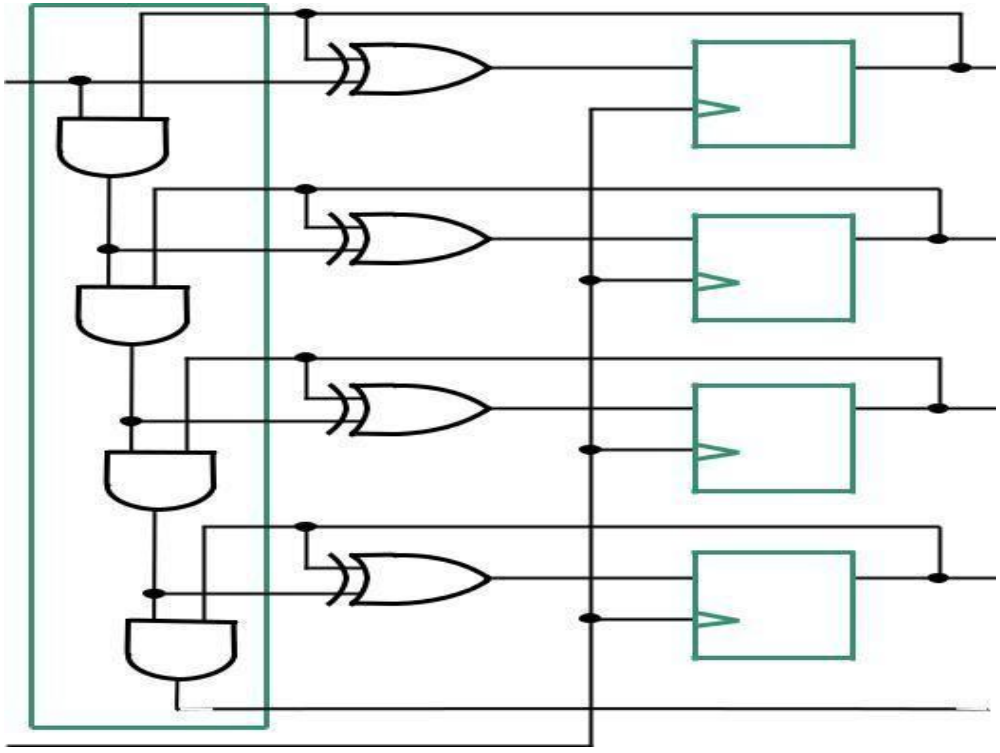


Flip-Flop Vs. Latch

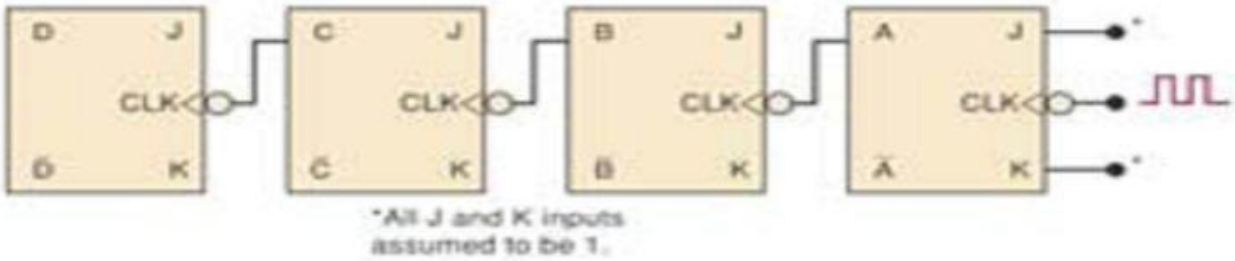
- The primary difference between a D flip-flop and D latch is the EN/CLOCK input.
- The flip-flop's CLOCK input is edge sensitive, meaning the flip-flop's output changes on the edge (rising or falling) of the CLOCK input.
- The latch's EN input is level sensitive, meaning the latch's output changes on the level (high or low) of the EN input.

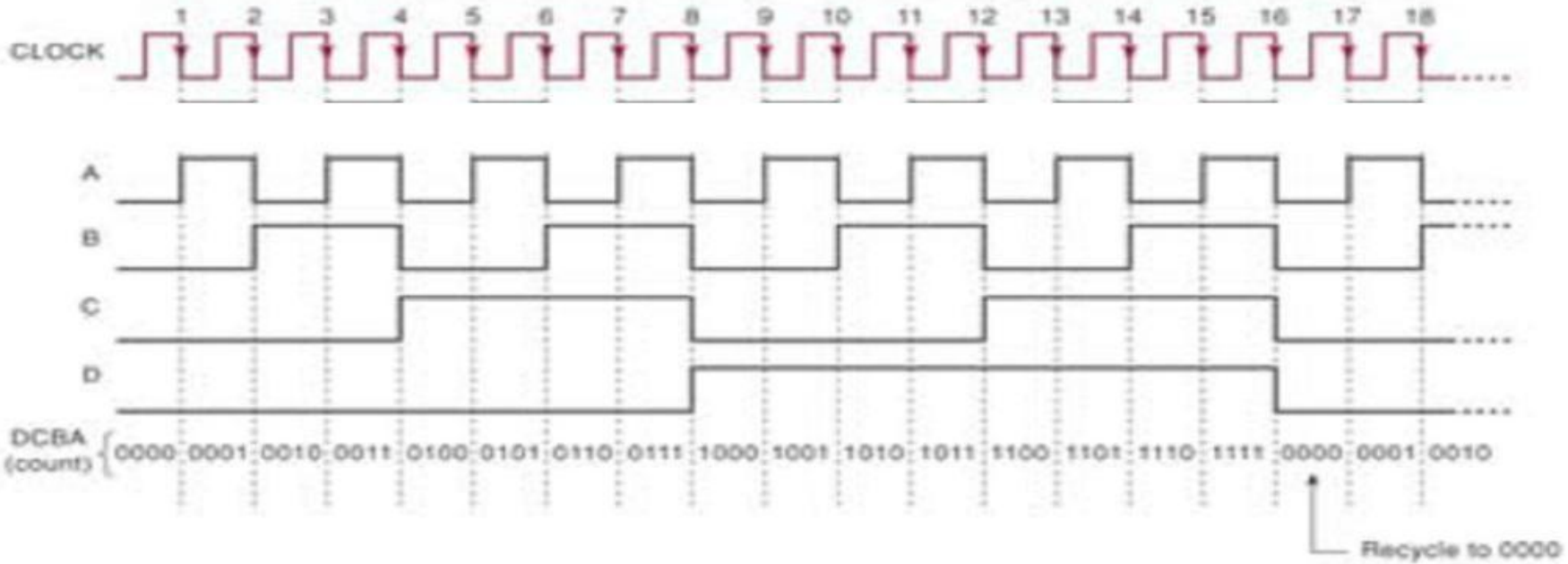
Synchronous Counters

- To eliminate the "ripple" effects, use a common clock for each flip-flop and a combinational circuit to generate the next state.



ASynchronous Counters

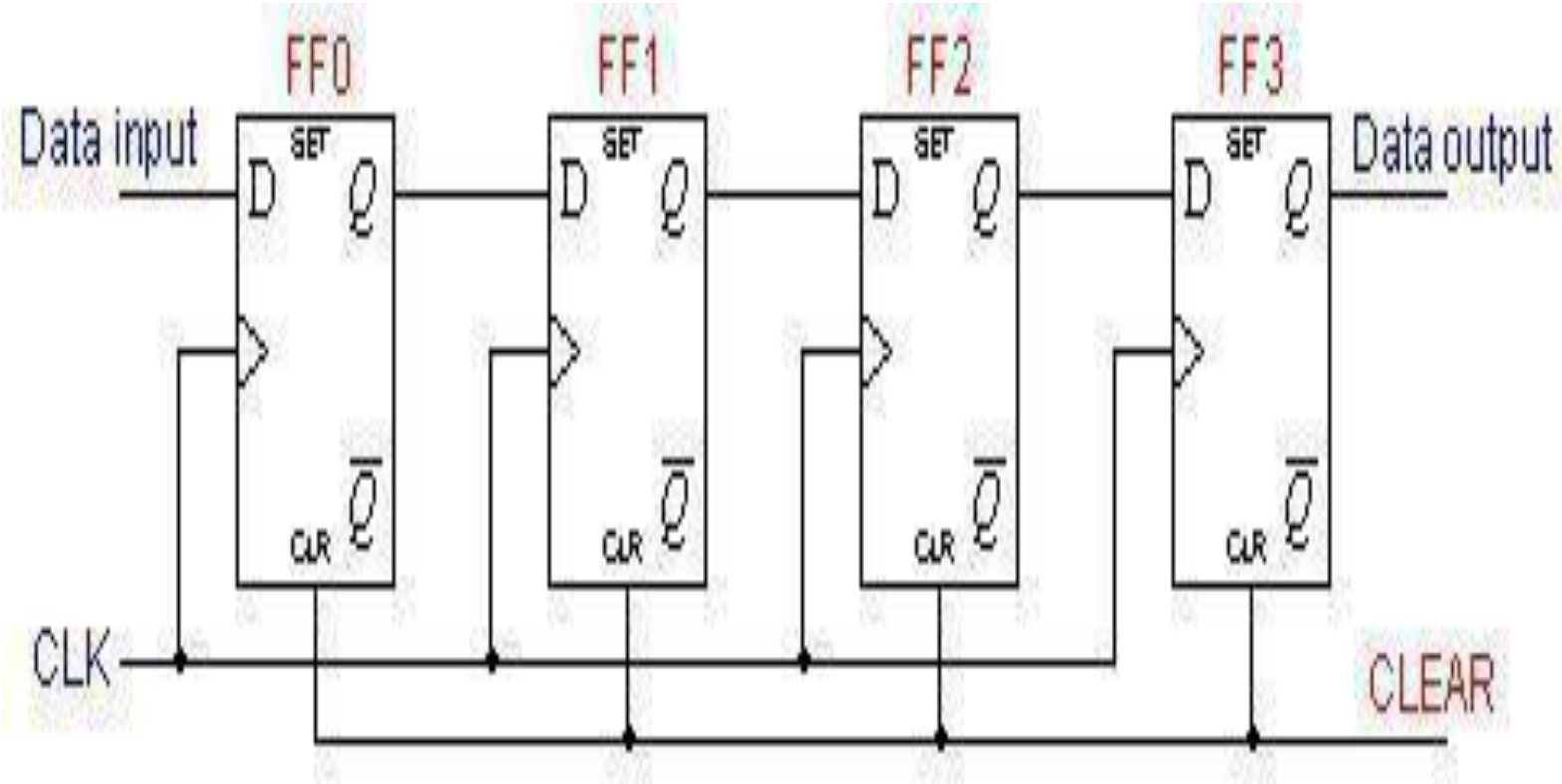




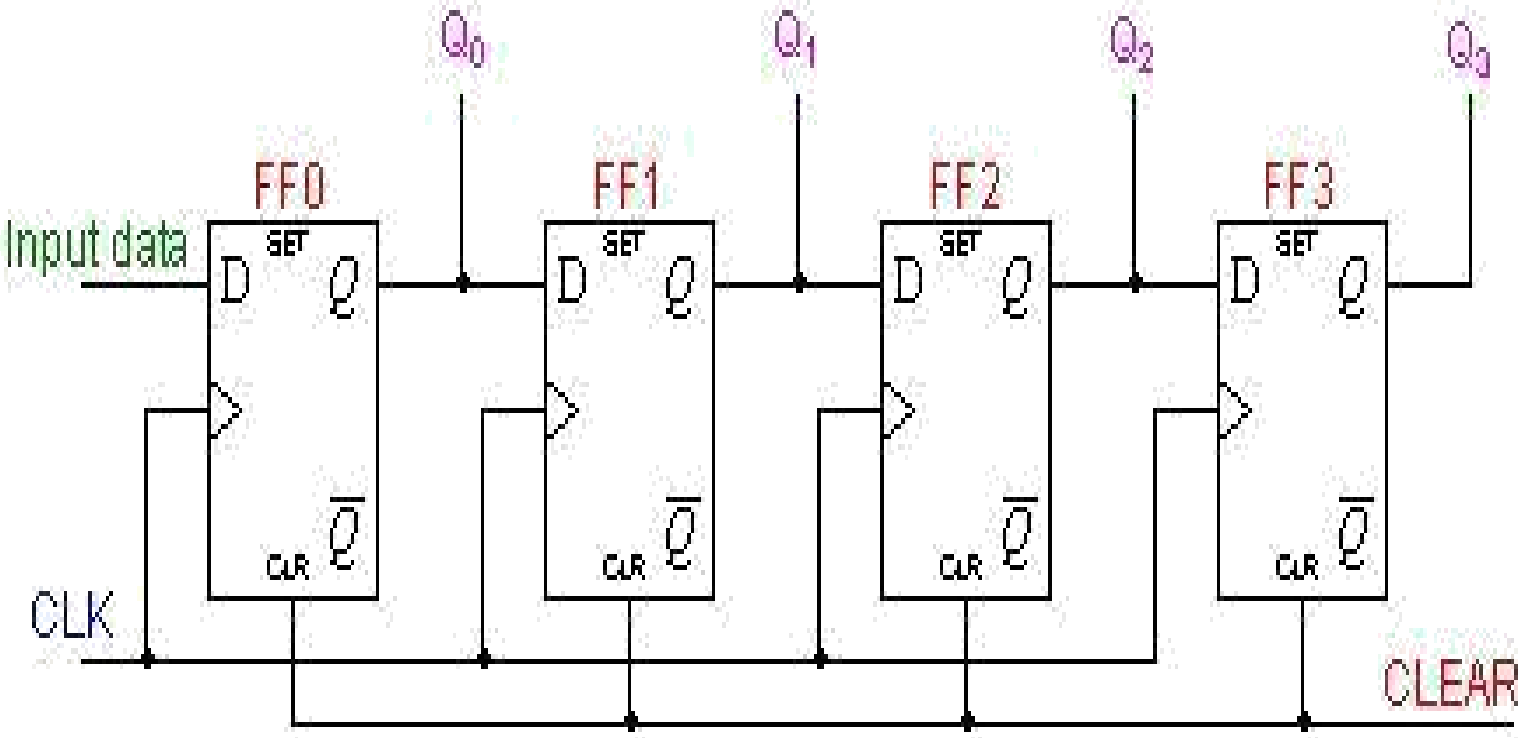
ASynchronous Counters

- Clock is applied only to FF A. J and K are high in all FFs to toggle on every clock pulse. Output of FF A is CLK of FF B and so forth.
- FF outputs D, C, B, and A are a 4 bit binary number with D as the MSB.
- After the negative transition of the 15th clock pulse the counter recycles to 0000.
- This is an asynchronous counter because state is not changed in exact synchronism with the clock.

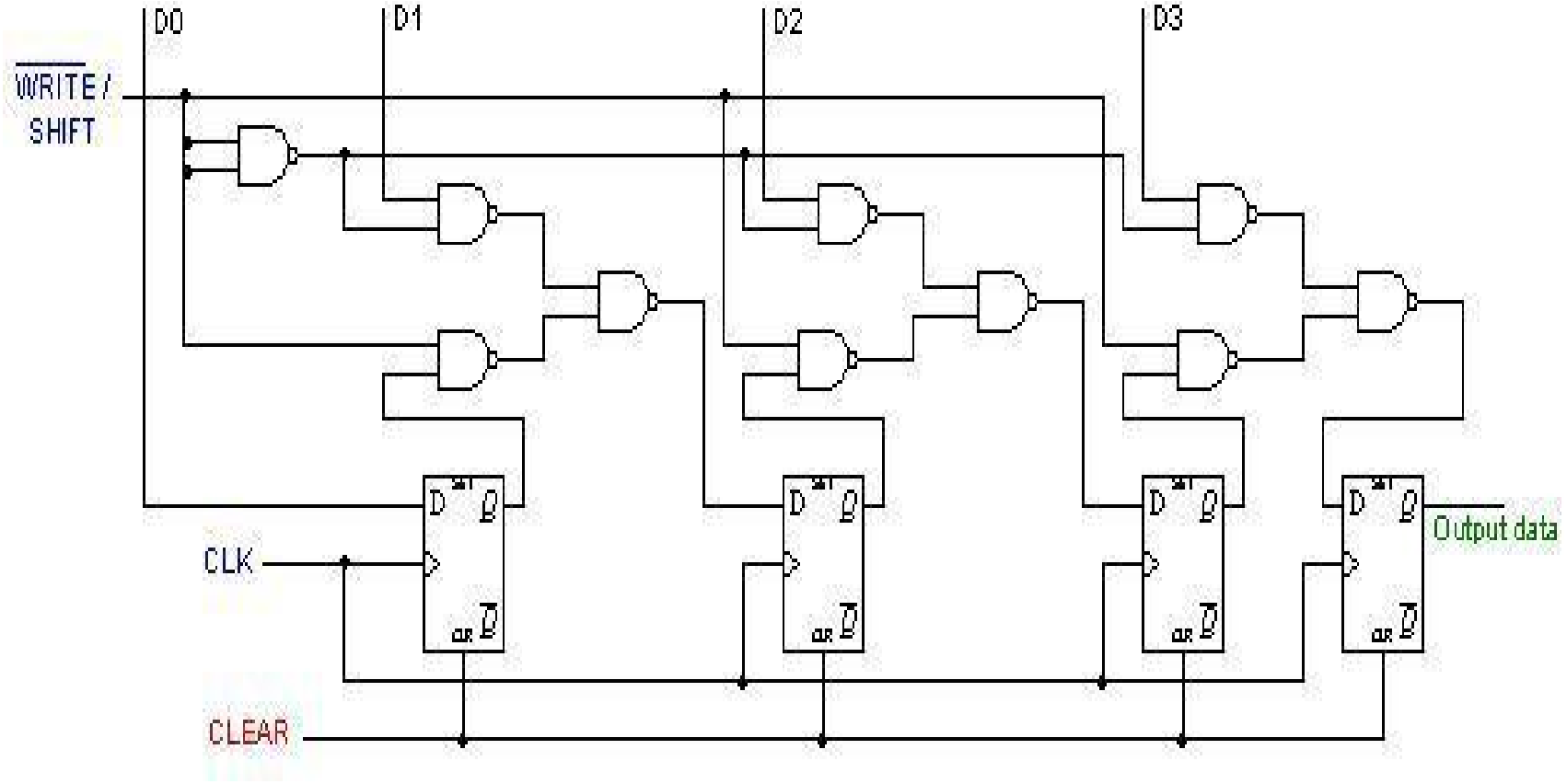
Serial In - Serial Out Shift Registers



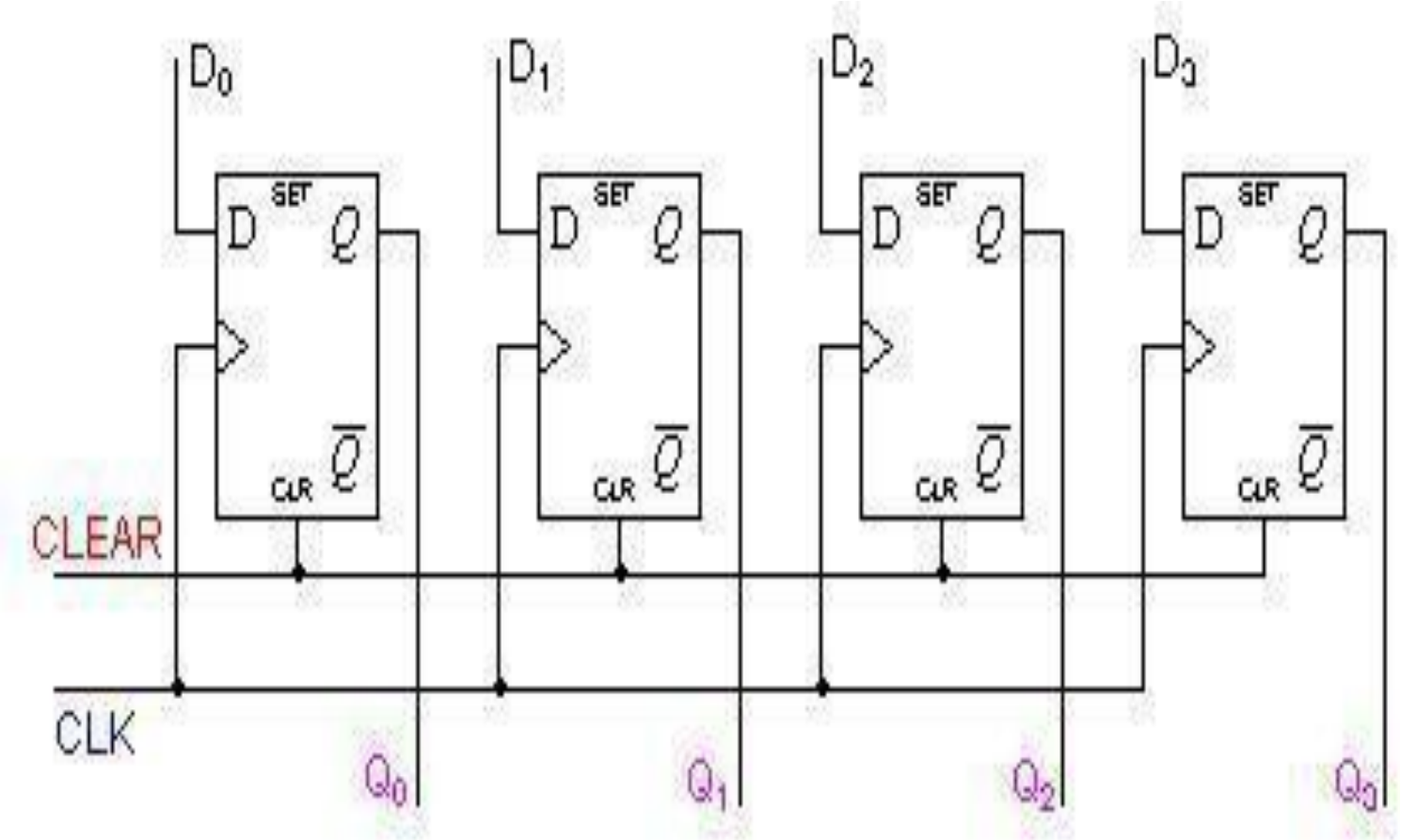
Serial In - Parallel Out Shift Registers



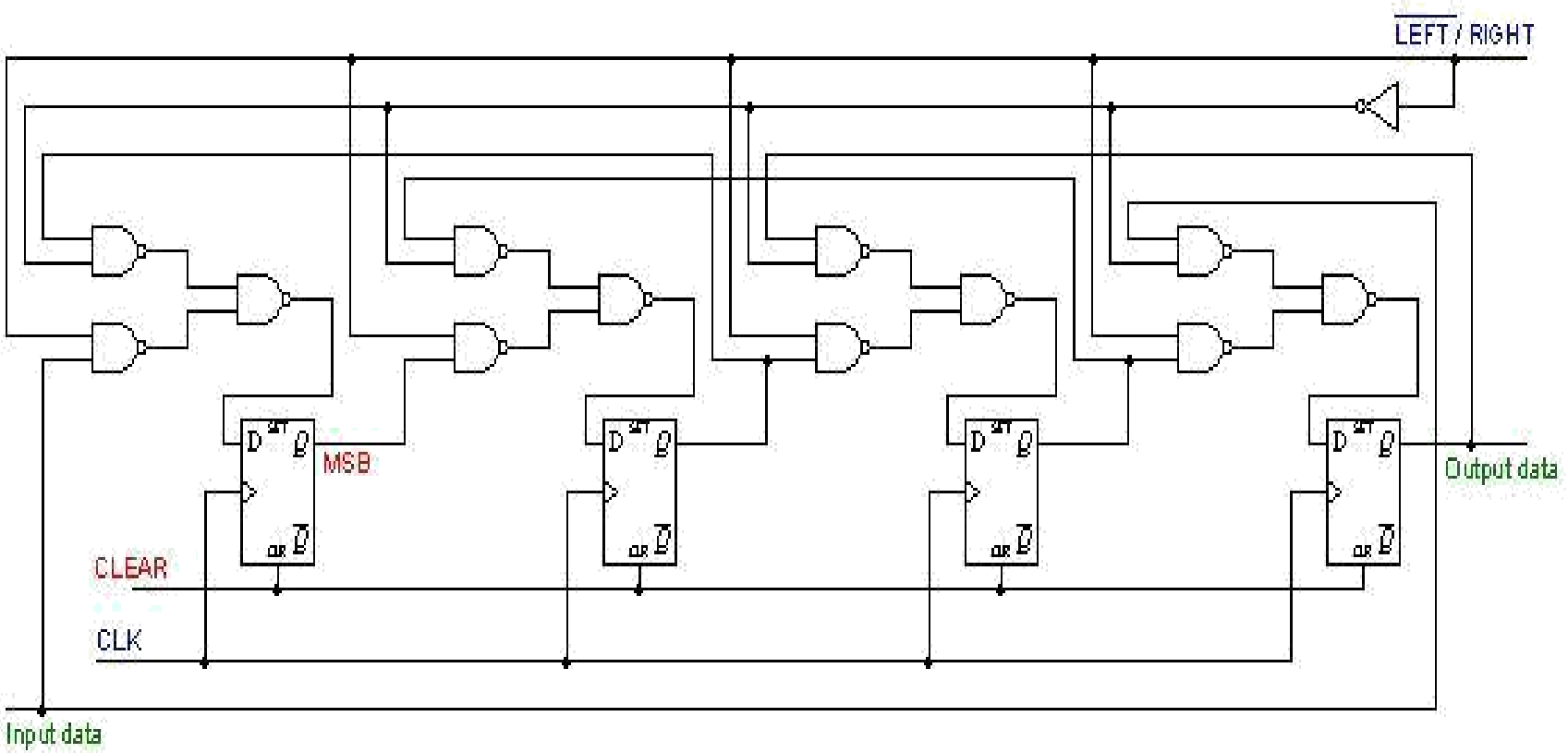
Parallel In - Serial Out Shift Registers



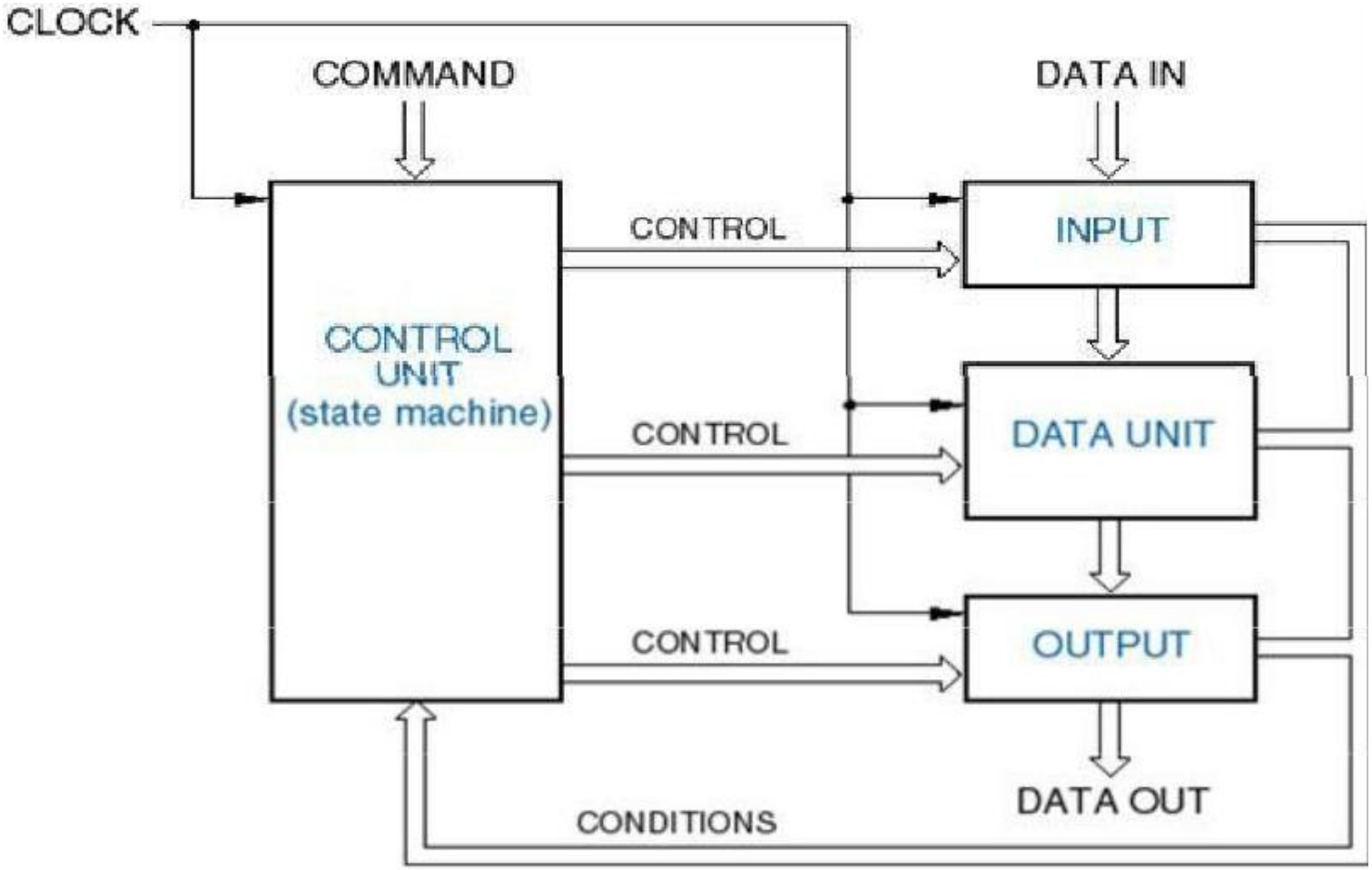
Parallel In - Parallel Out Shift Register



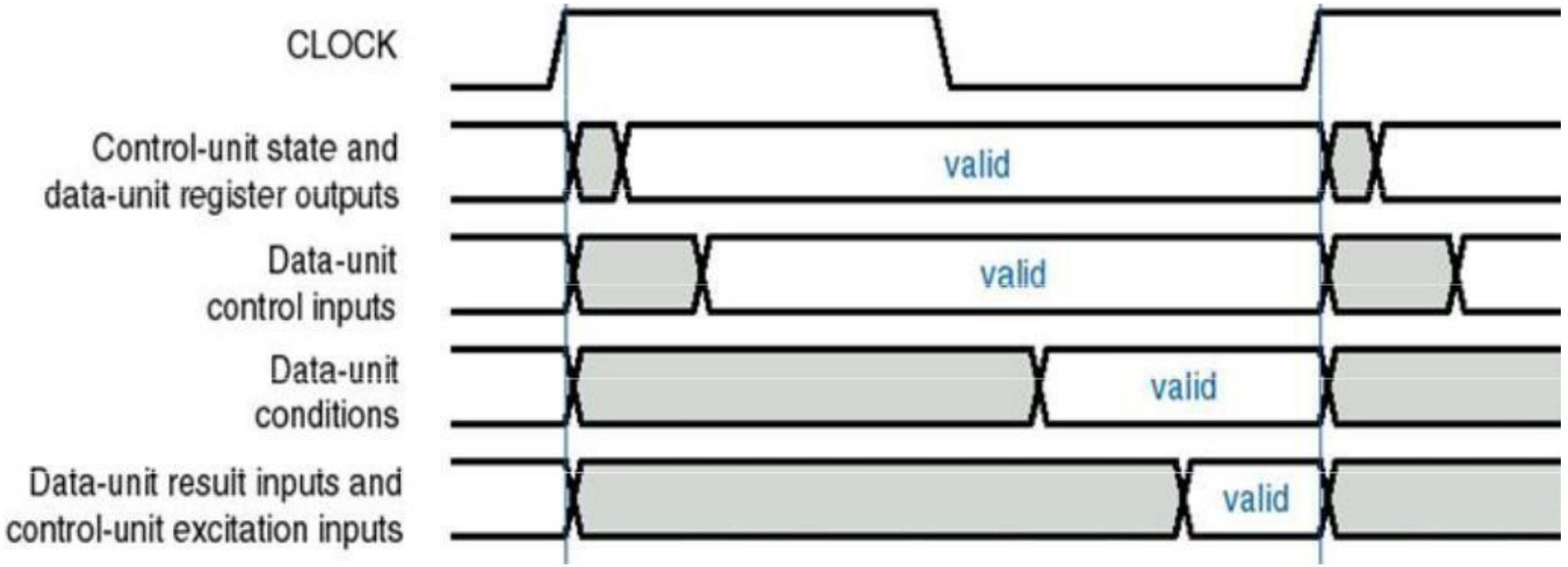
Bidirectional Shift Registers



Synchronous System Structure



Serial In - Parallel Out Shift Registers



Everything is clocked by the same, common clock.

Typical synchronous-system timing

Outputs have one complete clock period to propagate to inputs.

Must take into account flip-flop setup times at next clock period.

Impediments to synchronous design

Clock skew

- The difference between arrival times of the clock at different memory devices.

Influence of clock skew

- Reduce the setup and hold time margins.
- For proper operation tffpd(min) + tcomb(min) - thold - tskew(max) > 0
- tsetup -tclk -tffpd(max) - tcomb(max) - tskew(max) > 0
- Reducing clock skew proper buffering the clock Better clock distribution .

Gating clock

- It is for Asynchronous inputs
- Problem with asynchronous inputs is Meta-stable

UNIT – V

MEMORIES

Types of Memories

There are two types of memories that are used in digital systems:

Random-access memory (RAM): perform both the write and read operations.

Read-only memory (ROM): perform only the read operation.

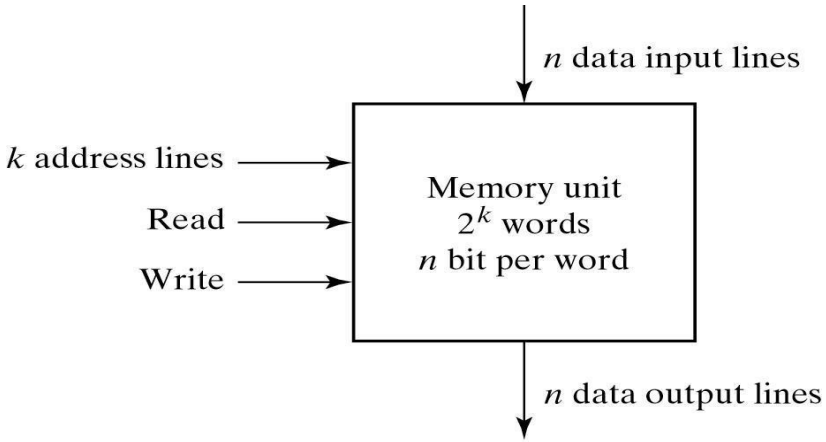


Fig. 7-2 Block Diagram of a Memory Unit

Write and Read operations

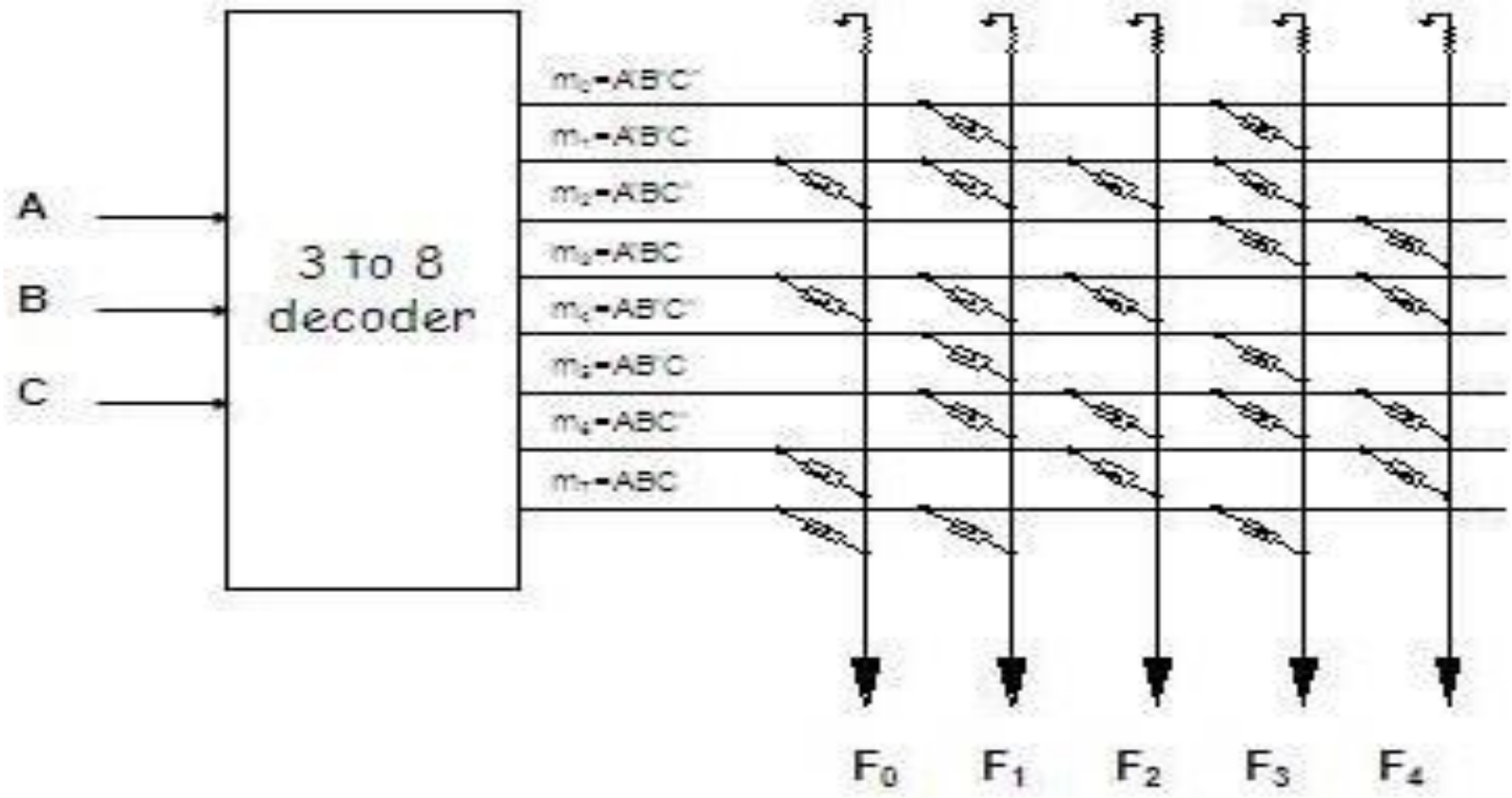
Transferring a new word to be stored into memory:

- Apply the binary address of the desired word to the address lines.
- Apply the data bits that must be stored in memory to the data input lines.
- Activate the write input.

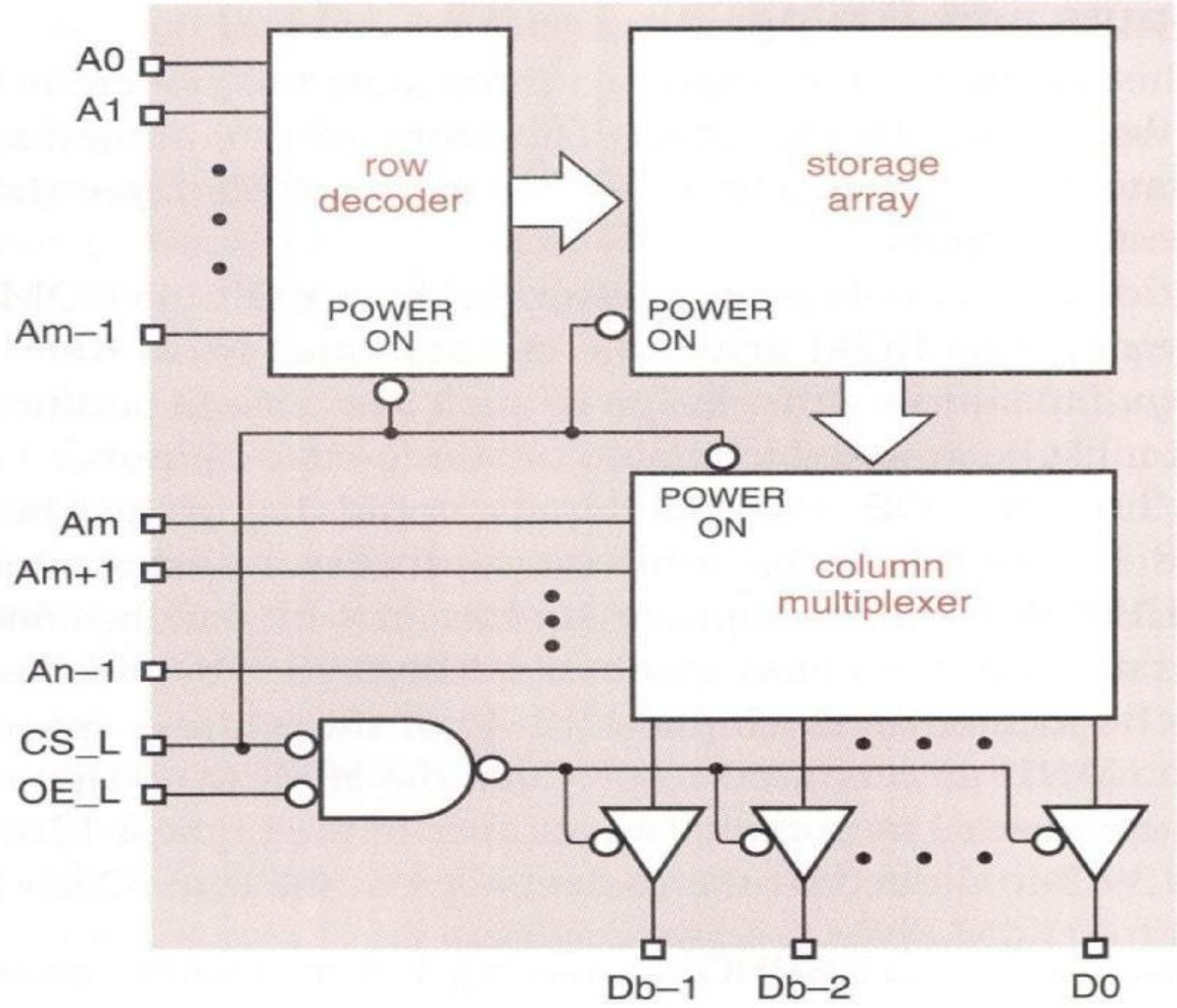
Transferring a stored word out of memory:

- Apply the binary address of the desired word to the address lines.
- Activate the read input.

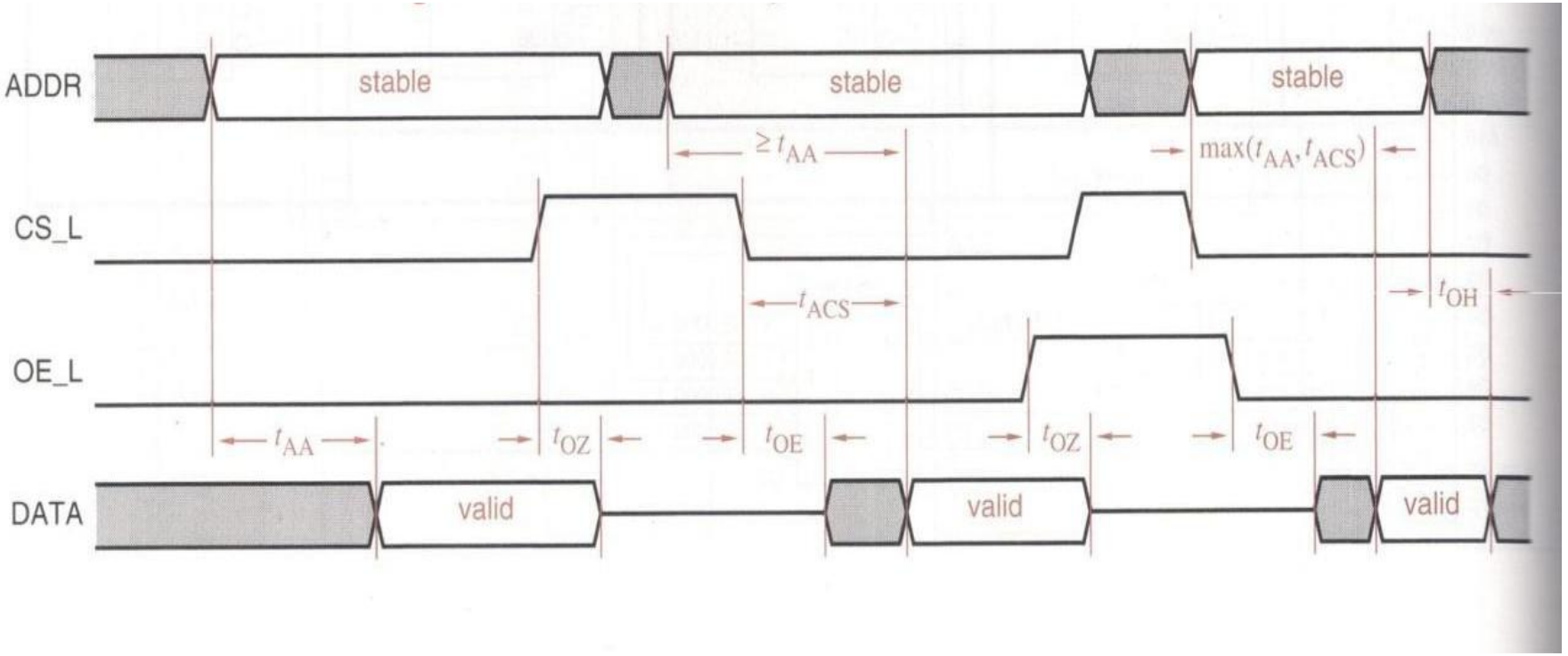
Internal Structure of ROM



General Block Diagram of ROM



Timing Diagram of ROM



Types of Memories

There are two types of memories that are used in digital systems:

Random-access memory (RAM): perform both the write and read operations.

Read-only memory (ROM): perform only the read operation.

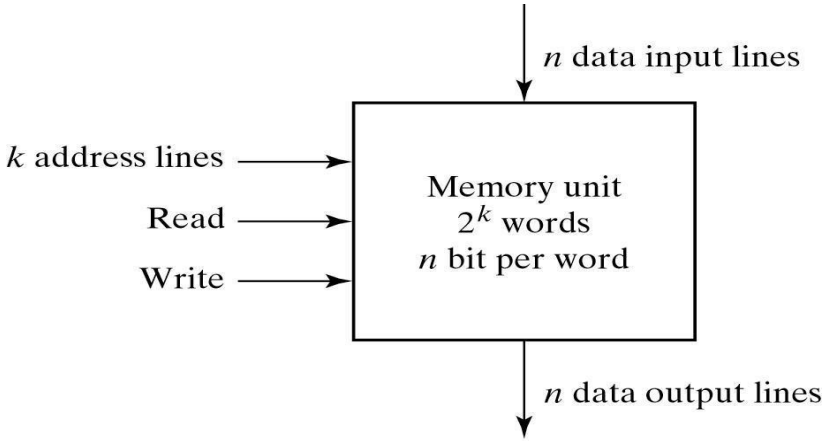


Fig. 7-2 Block Diagram of a Memory Unit

Write and Read operations

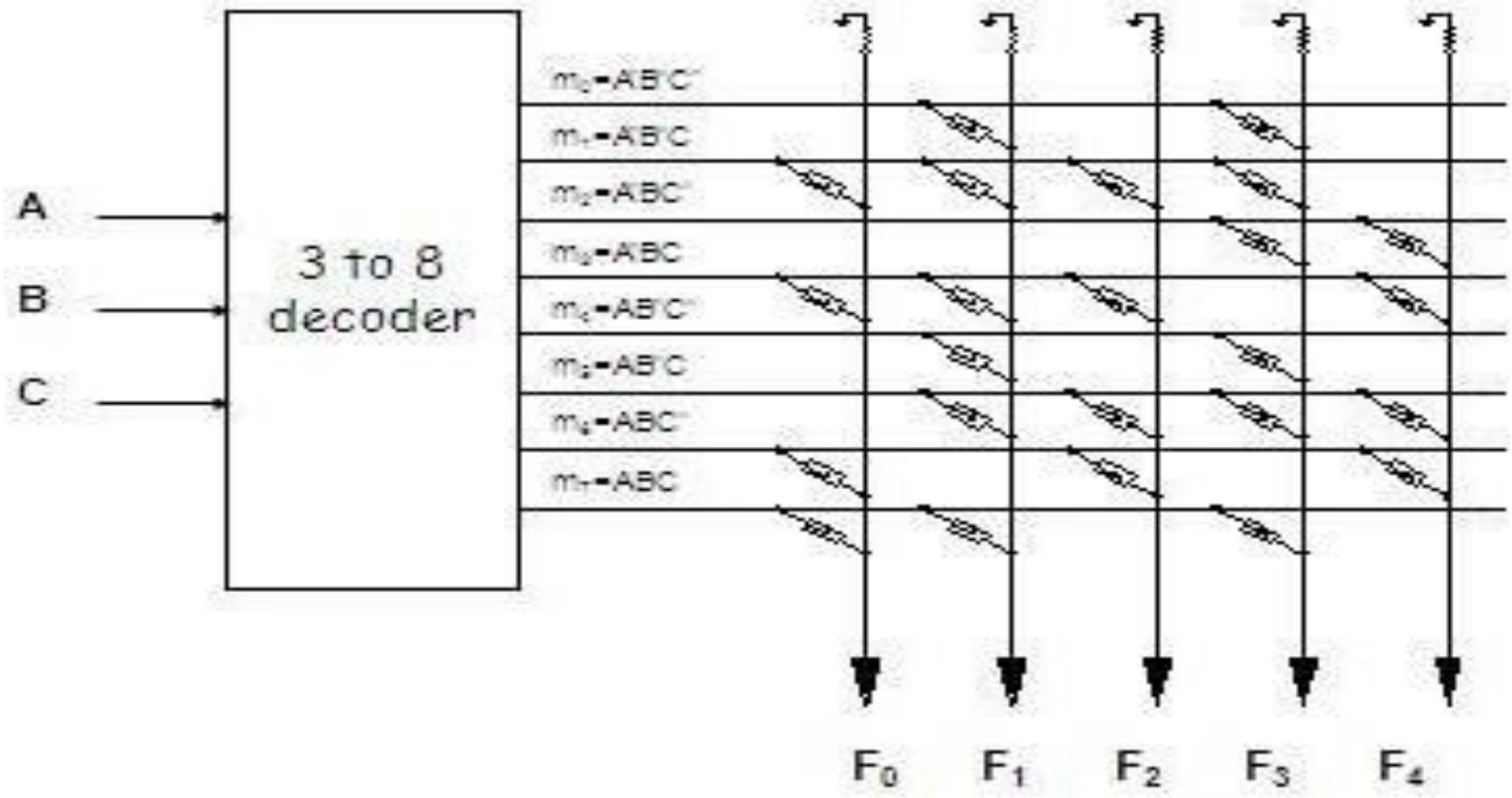
Transferring a new word to be stored into memory:

- Apply the binary address of the desired word to the address lines.
- Apply the data bits that must be stored in memory to the data input lines.
- Activate the write input.

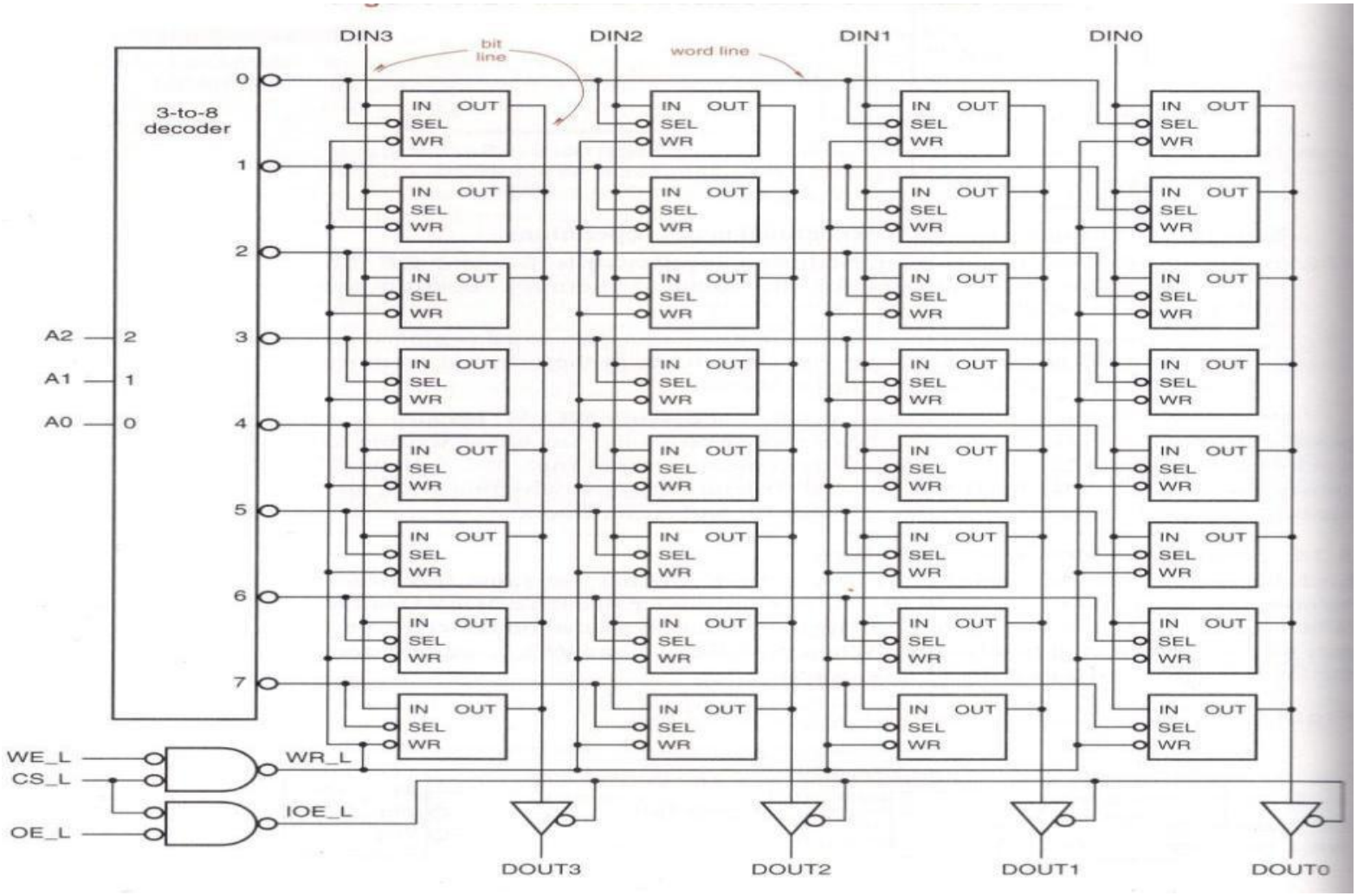
Transferring a stored word out of memory:

- Apply the binary address of the desired word to the address lines.
- Activate the read input.

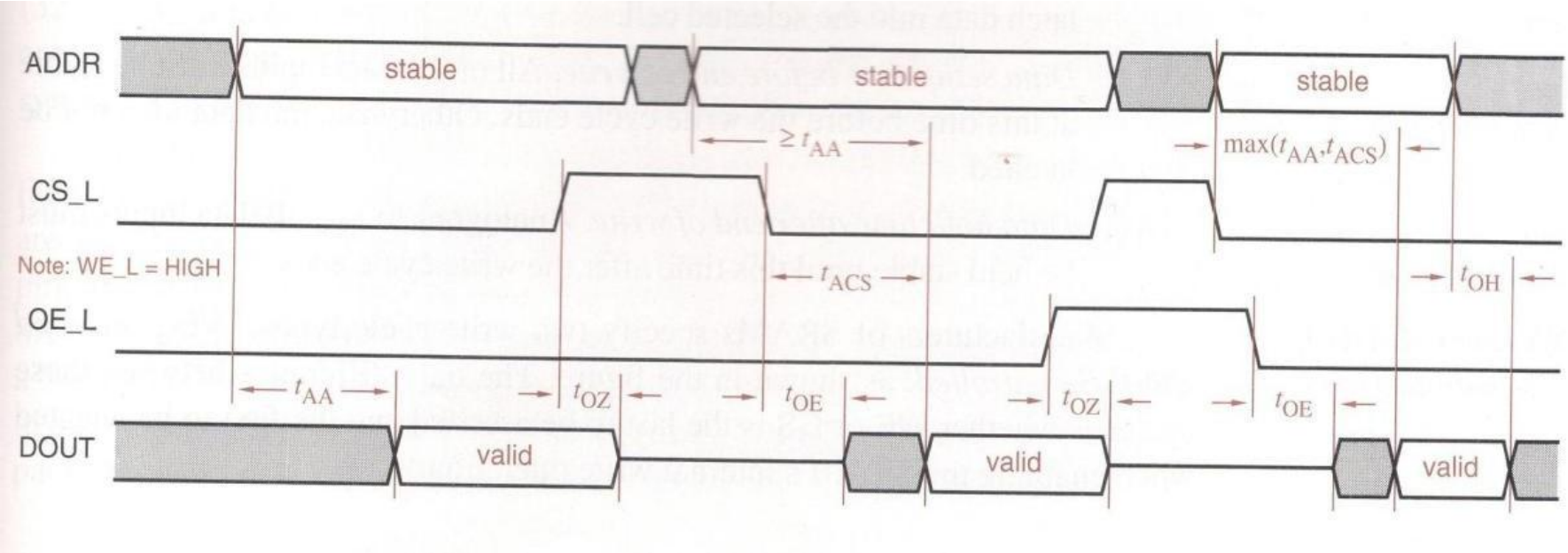
Internal Structure of ROM



Internal arrangement of storage structures

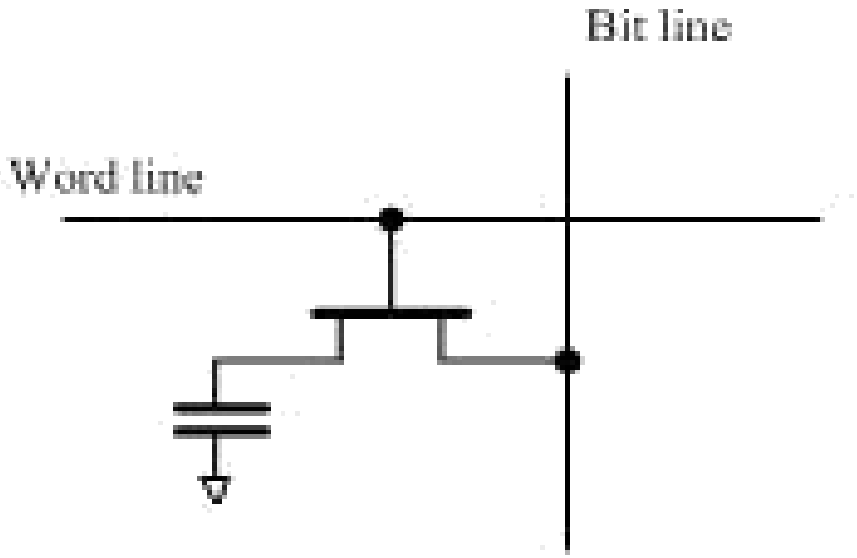


Timing for read



Internal arrangement of DRAM

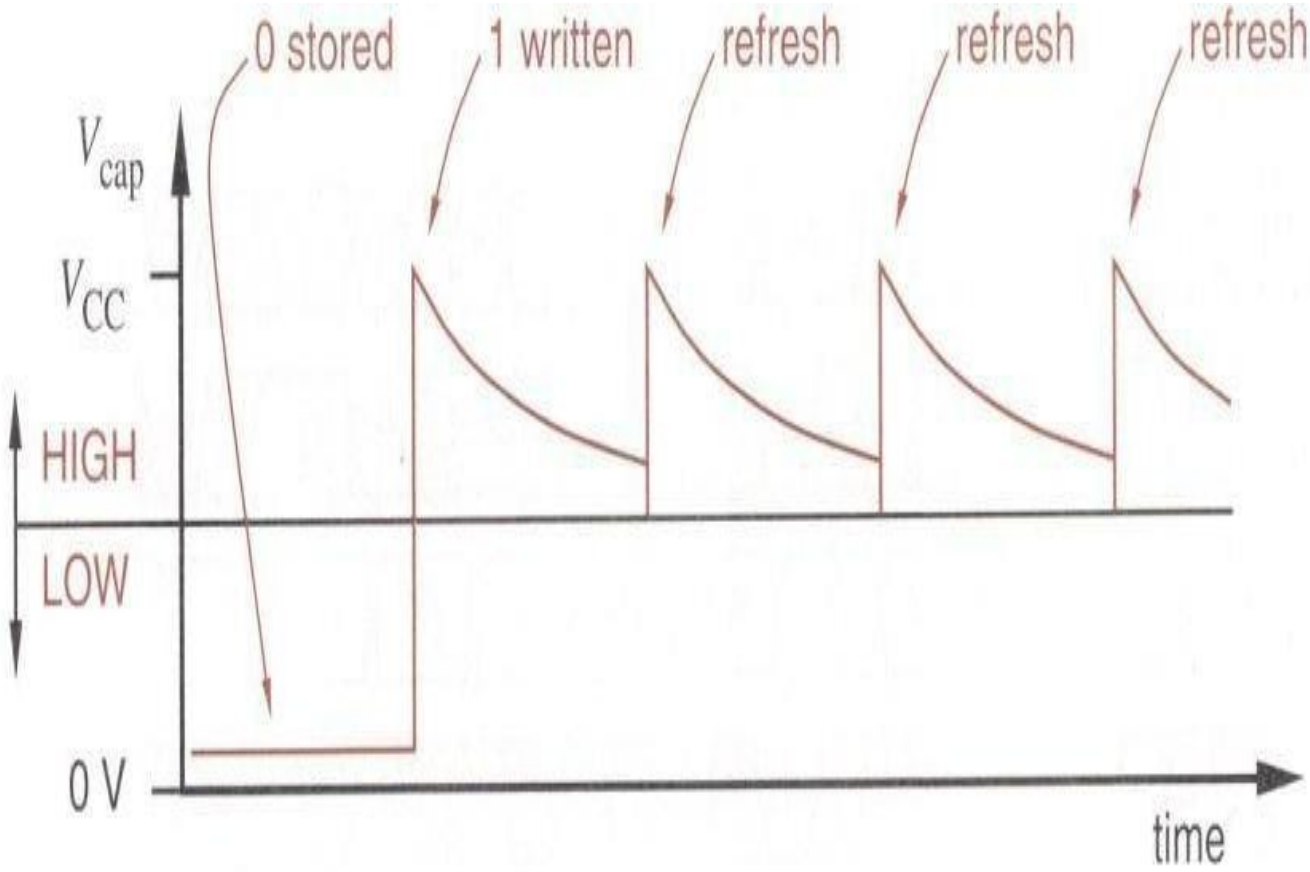
In DRAM data is stored in a semiconductor capacitor.



Internal arrangement of DRAM

- A read sees the bit line pre-charged to high. The word line is then activated.
- If cell stores a 0 then there is a small drop on the voltage on the bit line.
- This is monitored by a sense amp which provides the value stored.
- Value must be written back after the read.

DRAM Refresh



Operation of DRAMs

- Charge stored leaks off over time.
- Must restore the values stored a 4096 row
- DRAM it refresh every 64ms and thus each row every 15.6 usec.
- Larger DRAMs are banks of smaller.

Write and Read Operations

Write operation

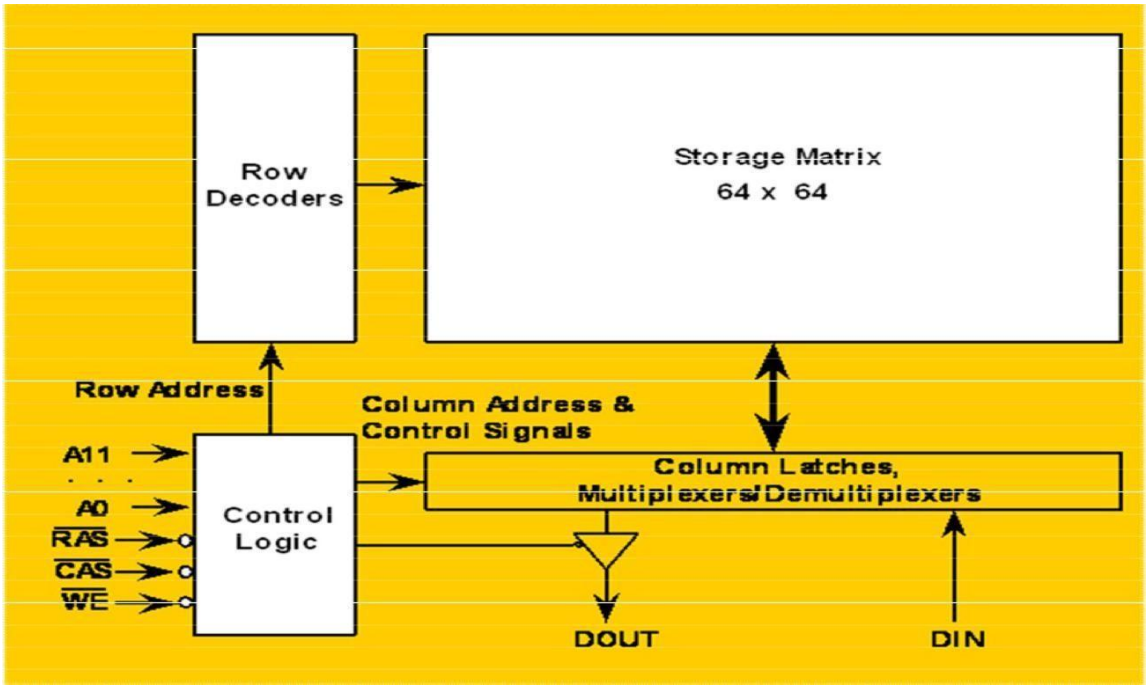
- Setting the word line to 1.To store a 1, a HIGH voltage is placed on the bit line, which charges the capacitor through the —on transistor.
- To store a 0, a LOW voltage is placed on the bit line, which discharges the capacitor through the —on transistor.

Read operation

- The bit line is first precharged to a voltage halfway between HIGH and LOW.
- The word line is set HIGH so that the precharged bit line is pulled slightly higher or slightly lower.
- A sense amplifier detects this small change and recovers a 1 or 0 accordingly. Reading a DRAM cell destroy the original voltage stored on the capacitor, the DRAM cell must be written back the original data after reading.

SYNCHRONOUS DYNAMIC RAM

In a synchronous DRAM, the control signals are synchronized with the system bus clock and therefore with the microprocessor. It allows *pipelined* read/write operations



SYNCHRONOUS DYNAMIC RAM

- Tied to the system clock Burst mode
  - System timing : 5-1-1-1
  - Internal interleaving New memory

. standard for modern PCs Speed

  - Access time: 10ns, 12ns,…
  - MHz rating: 100 MHz, 133MHz

.

Latency

- SDRAMs are still DRAMs
- 5-1-1-1 (10ns means the second, third and fourth access times) 2-clock and 4-

.

clock Circuitry

- 2-clock: 2 different DRAM chips on the module

Comparison of semiconductor memories

	Bit org.	Cell size (mm^2)	Chip size (mm^2)	Cell 효율 (%)	Real Access/ cycle	Write cycle	Erase time	Write 횟수	Power consumption (Act./Stdby)
64M DRAM	8Mx8b 8k ref	1.7	211	0.52	38/100 (ns)	100 (ns)	-		85/ (mA)
16M SRAM	2Mx8b	8.4	226	0.59	14/33 (ns)	14 (ns)	-		70/ (mA)
64M Flash	4Mx16b	1.7	257	0.42	50/100 (ns)	6.4 (μs)	0.8s	10^4~ 10^5	30/0.1 (mA)

* 0.4 mm design rule