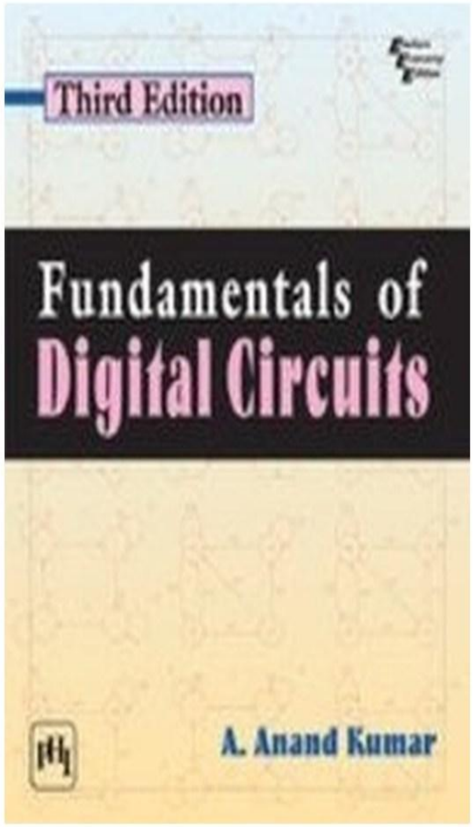
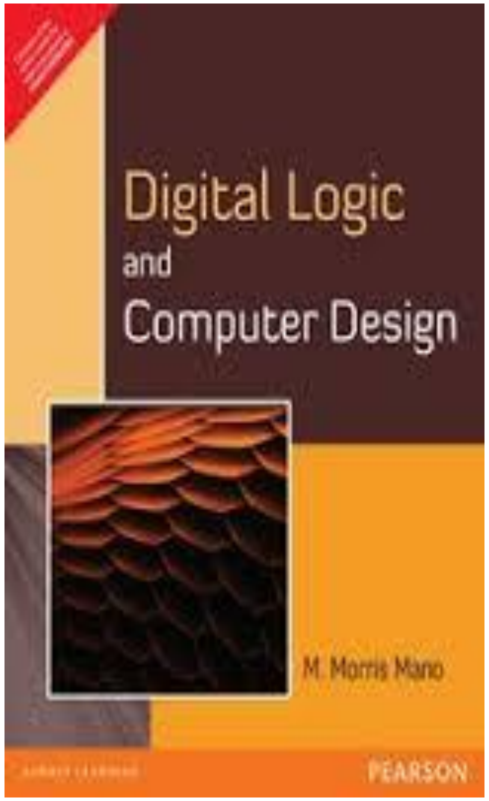
Digital Electronics

(19-204-0302)

Text Books

- 1. M. Morris Mano, *Digital Logic & Computer Design*, 4/e, Pearson Education, 2013.
- 2. FUNDAMENTALS OF DIGITAL CIRCUITS THIRD EDITION A. Anand Kumar





Module - 2

Syllabus - Module 2

- Combinational Logic Circuits: Design of Adders, Subtractors, Binary Parallel Adder, Carry look ahead Adder, BCD Adder, Multiplexer, Demutiplexer, Magnitude Comparator, Decoder, Encoder

<u>Combinational Logic</u>

- Logic Circuits (Digital Circuits) are of two types.

- 1. Combinational Circuits.
- 2. Sequential Circuits.

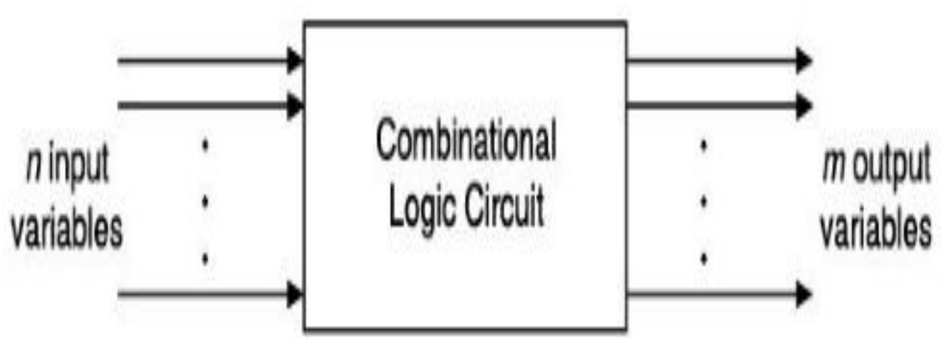
- <u>Combinational Circuits</u>

- The output of a combinational circuit depends on its present inputs only.

- <u>Sequential Circuits.</u>

- The output of a sequential circuit depends on not only its present inputs but also the past inputs.

Block diagram of a Combinational Circuit



- A combinational circuit consists of input variables, logic gates, and output variables.
- The logic gates accept signals from the inputs and generate signals to the outputs.
- The n input binary variables come from an external source and the m output variables go to an external destination.
- For n input variables, there are 2^n possible combinations of binary input values.

Design Procedure of Combinational Logic

The procedure involves the following steps:

- 1. The problem is stated.
- 2.The number of available input variables and required output variables is determined.
- 3.The input and output variables are assigned letter symbols.
- 4. The truth table that defines the required relationship between inputs and outputs is derived.
- 5.The simplified Boolean function for each output is obtained.
- 6.The logic diagram is drawn.

ADDER Circuits

- Arithmetic Circuits which perform addition operations are called Adder Circuits.

- The most basic arithmetic operation is the addition of two binary digits.

- This simple addition consists of 4 possible operations, namely,

- 0 + 0 = 0
- 0 + 1 = 1
- 1 + 0 = 1
- 1 + 1 = 10.

Types of Adder Circuits

- Adder Circuits are of two types.

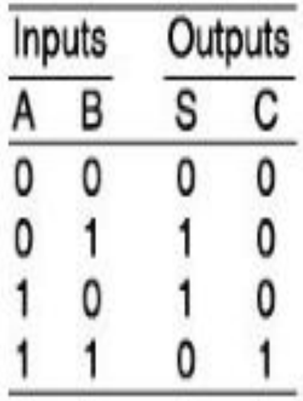
- 1. Half Adder
- 2. Full Adder

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

- A combinational circuit that performs the addition of three bits (two significant bits and a previous carry) is called a full-adder.

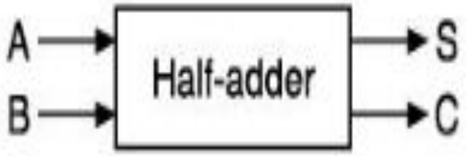
The Half-Adder

- A half-adder is a combinational circuit with two binary inputs (augend and addend bits) and two binary outputs (sum and carry bits).
- It adds the two inputs (A and B) and produces the sum (S) and the carry (C) bits.
- The truth table and block diagram of a half-adder are shown below.



Inputs		Outputs	
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

(a) Truth table



(b) Block diagram

- The sum (S) is the X-OR of A and B .

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

- The carry (C) is the AND of A and B. Therefore, **C = AB**

A half-adder can, therefore, be realized by using one X-OR gate and one AND gate as shown in Figure 7.3a. Realization using AOI logic is shown in Figure 7.3b.

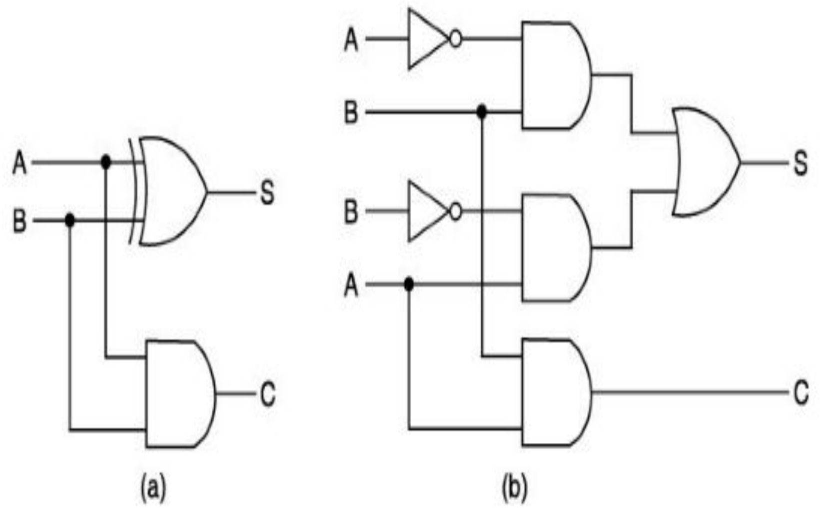


Figure 7.3 Logic diagrams of half-adder.

Realization of Half Adder using NAND Gates only

NAND logic

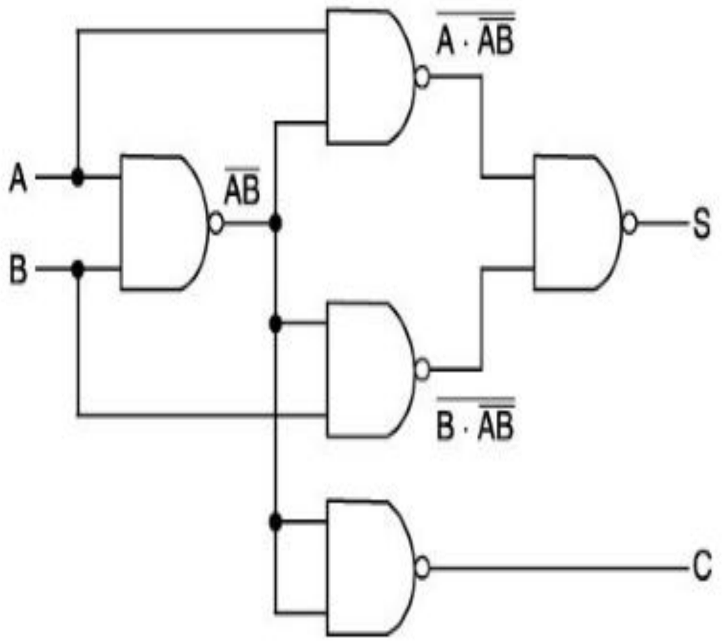
$$S = A\bar{B} + \bar{A}B = A\bar{B} + A\bar{A} + \bar{A}B + B\bar{B}$$

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

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

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

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



A
B
$\overline{AB}$
$\overline{A \cdot \overline{AB}}$
$\overline{B \cdot \overline{AB}}$
S
C

Realization of Half Adder using NOR Gates only

NOR logic

$$S = A_{\bar{B}} + {}_{\bar{A}}B = A_{\bar{B}} + A_{\bar{A}} + {}_{\bar{A}}B + B_{\bar{B}}$$

$$= A({}_{\bar{A}} + {}_{\bar{B}}) + B({}_{\bar{A}} + {}_{\bar{B}})$$

$$= (A + B)({}_{\bar{A}} + {}_{\bar{B}})$$

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

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

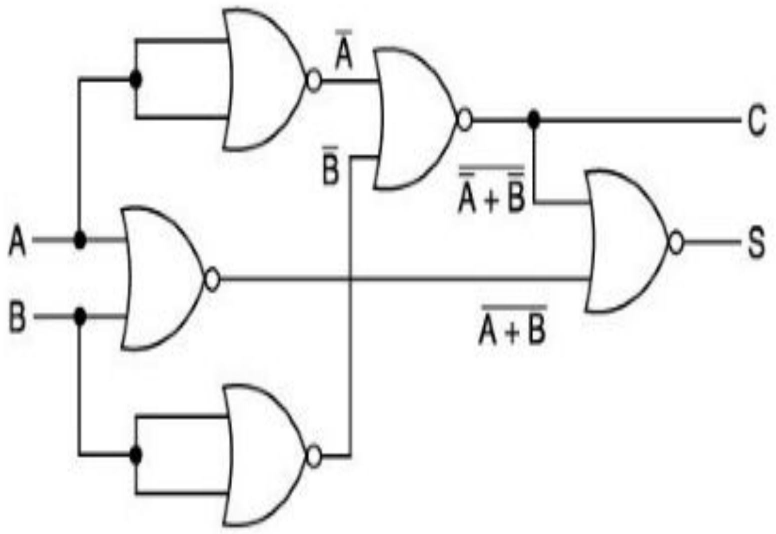
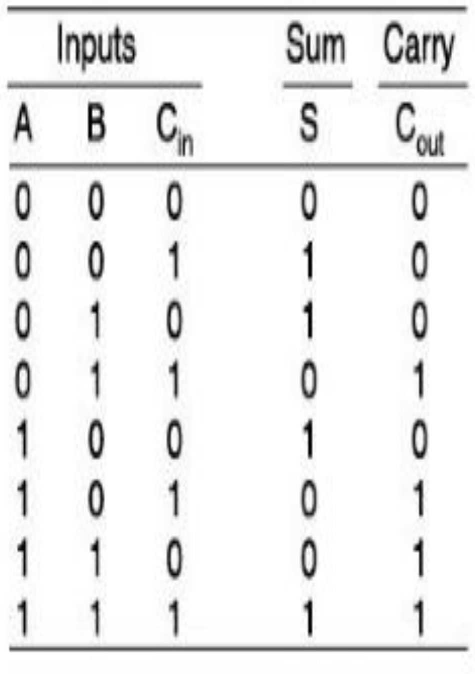


Figure 7.5 Logic diagram of a half-adder using only 2-input NOR gates.

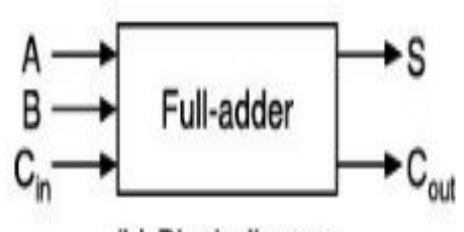
The Full-Adder

- A full-adder is a combinational circuit that adds two bits and a carry and outputs a sum bit and a carry bit.
- The full-adder adds the bits A and B and the carry from the previous column called the carry-in **C$_{\text{in}}$** and outputs the sum bit **S** and the carry bit called the



Inputs			Sum	Carry
A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

(a) Truth table

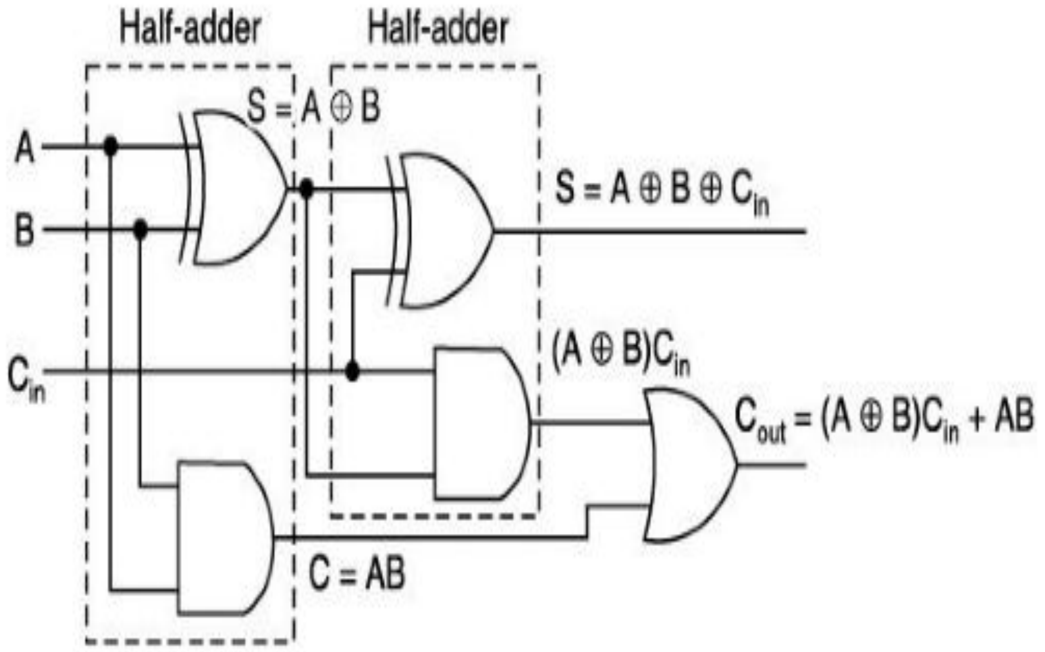


(b) Block diagram

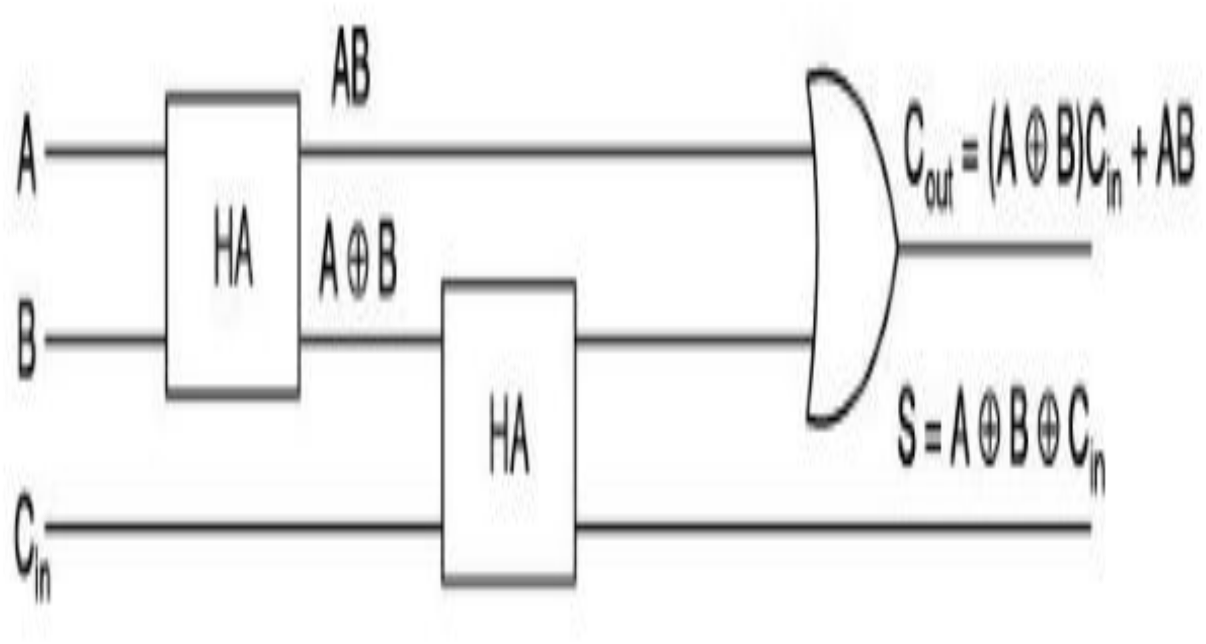
$$S = \bar{A}\bar{B}C_{in} + \bar{A}B\bar{C}_{in} + A\bar{B}\bar{C}_{in} + ABC_{in}$$

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

$$C_{out} = \bar{A}BC_{in} + A\bar{B}C_{in} + AB\bar{C}_{in} + ABC_{in} = AB + (A \oplus B)C_{in} = AB + AC_{in} + BC_{in}$$



Full Adder Using 2 Half Adders


A
B
C_{in}
HA
HA
AB
$A \oplus B$
$C_{out} = (A \oplus B)C_{in} + AB$
$S = A \oplus B \oplus C_{in}$

Realization of Full Adder Using AOI Gates

$$A \oplus B = \overline{\overline{A \cdot \overline{AB}} \cdot \overline{B \cdot \overline{AB}}}$$

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

$$C_{out} = C_{in}(A \oplus B) + AB = \overline{\overline{C_{in}(A \oplus B)} \cdot \overline{AB}}$$

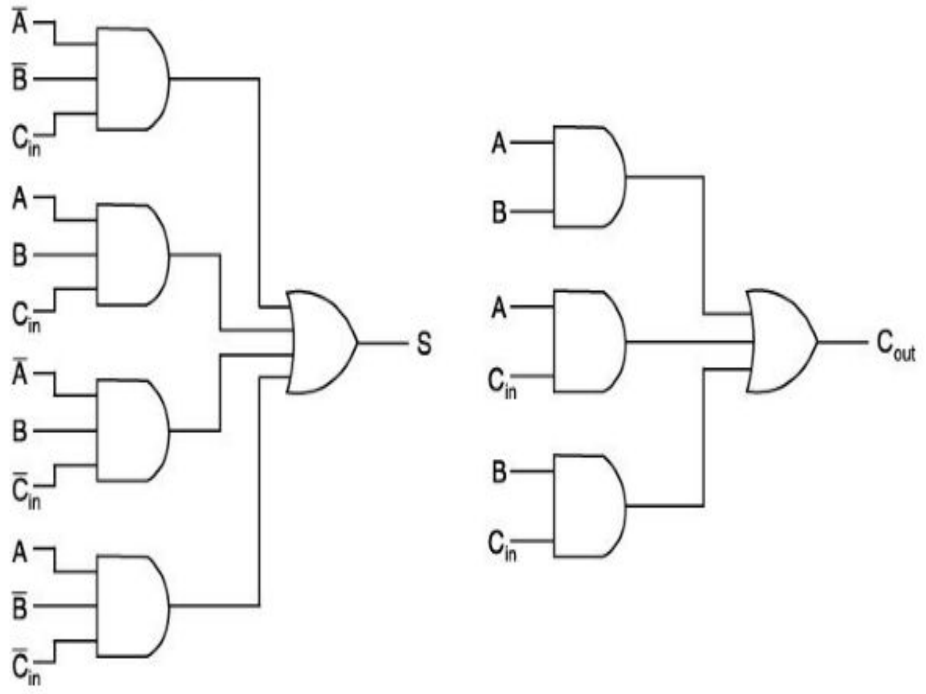


Figure 7.9 Sum and carry bits of a full-adder using AOI logic.

Realization of Full Adder Using NAND Gates

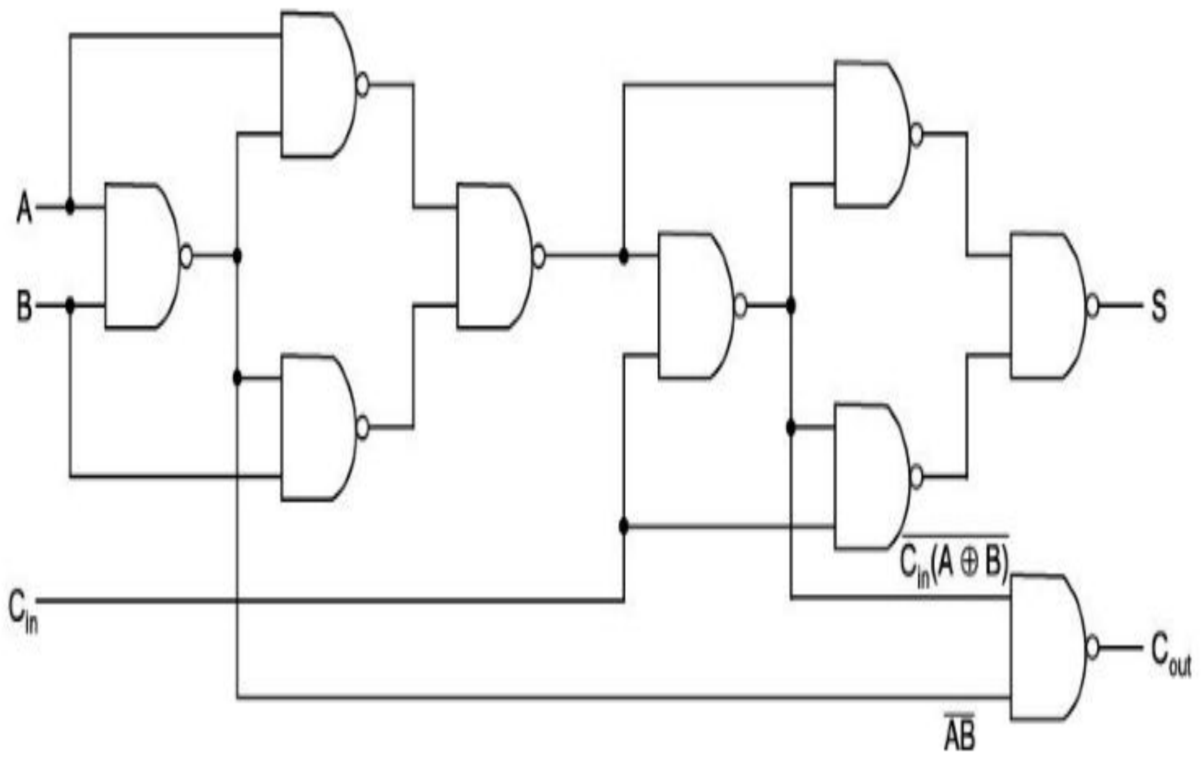


Figure 7.10 Logic diagram of a full-adder using only 2-input NAND gates.

Realization of Full Adder Using NOR Gates

NOR logic

We know that

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

Then

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

$$C_{out} = AB + C_{in}(A \oplus B) = \overline{\overline{\overline{A} + \overline{B}} + \overline{\overline{C_{in}} + \overline{A \oplus B}}}$$

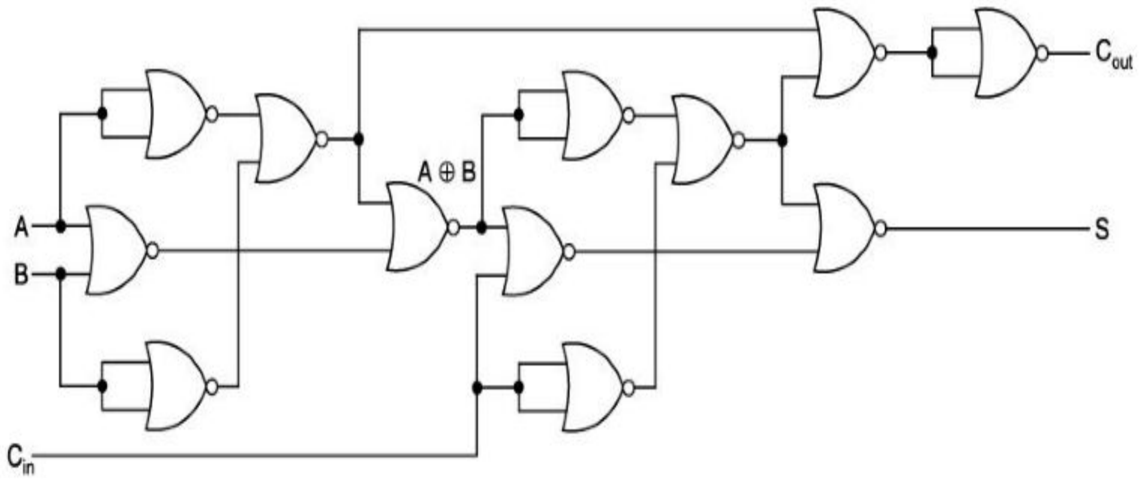


Figure 7.11 Logic diagram of a full-adder using only 2-input NOR gates.

SUBTRACTORS

- Arithmetic Circuits which perform subtraction operations are called Subtractor Circuits.

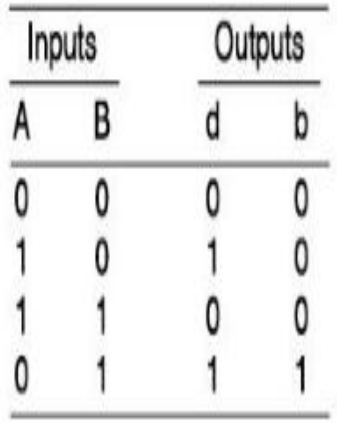
- The Half-Subtractor

- A half-subtractor is a combinational circuit that subtracts one bit from the other and produces the difference.

- It also has an output to specify if a 1 has been borrowed.

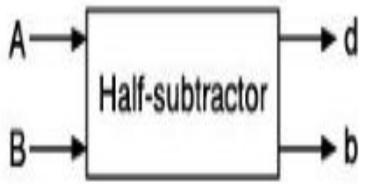
- It is used to subtract the LSB of the subtrahend from the LSB of the minuend when one binary number is subtracted from the other.

- A half-subtractor is a combinational circuit with two inputs A and B and two outputs d and b.



Inputs		Outputs	
A	B	d	b
0	0	0	0
1	0	1	0
1	1	0	0
0	1	1	1

(a) Truth table



(b) Block diagram

$$d = A\bar{B} + \bar{A}B = A \oplus B \text{ and } b = \bar{A}B$$

Logic Diagram of Half Subtractor

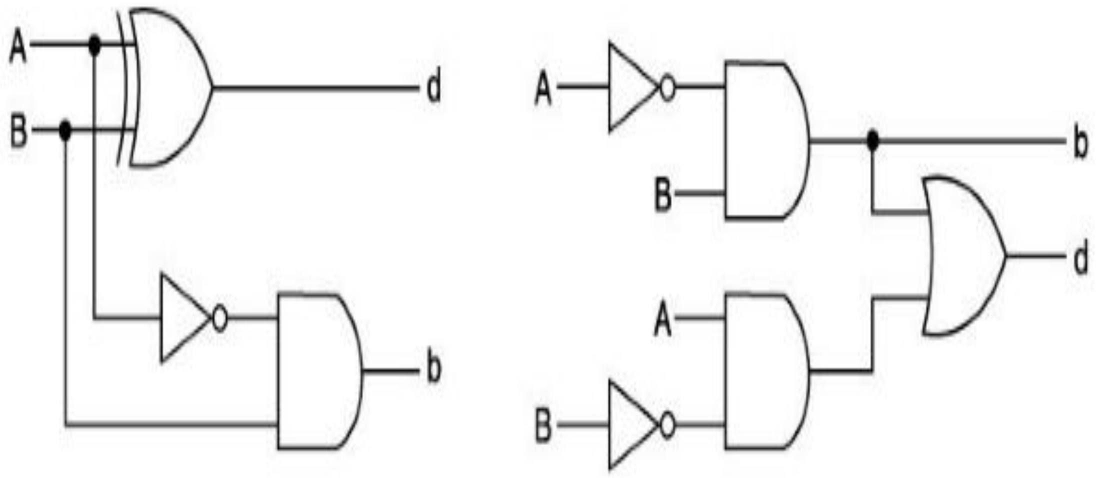


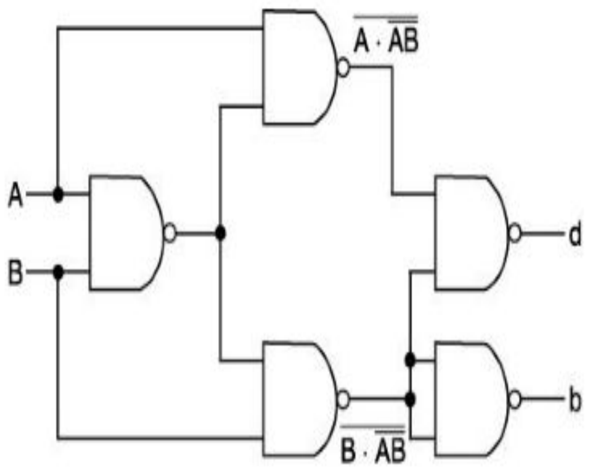
Figure 7.13 Logic diagrams of a half-subtractor.

Realization of Half Subtractor using NAND Gates

NAND logic

$$d = A \oplus B = \overline{\overline{A \cdot \overline{AB}} \cdot \overline{B \cdot \overline{AB}}}$$

$$b = \bar{A}B = B(\bar{A} + \bar{B}) = B(\overline{AB}) = \overline{\overline{B \cdot \overline{AB}}}$$



$\overline{A \cdot \overline{AB}}$
A
B
d
b
$\overline{B \cdot \overline{AB}}$

Figure 7.14 Logic diagram of a half-subtractor using only 2-input NAND gates.

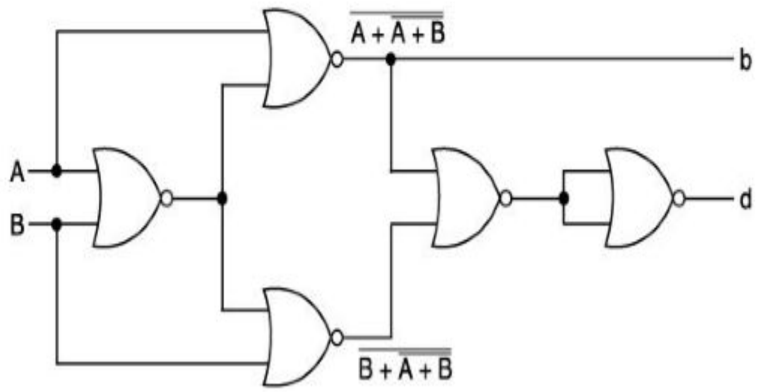
Realization of Half Subtractor using NOR Gates

NOR logic

$$d = A \oplus B = A_{\bar{B}} + {}_{\bar{A}}B = A_{\bar{B}} + B_{\bar{B}} + {}_{\bar{A}}B + A_{\bar{A}}$$

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

$$d = {}_{\bar{A}}B = {}_{\bar{A}}(A + B) = \overline{\overline{\bar{A}(A + B)}} = \overline{A + \overline{(A + B)}}$$



$\overline{A + \overline{A + B}}$
b
A
B
d
$\overline{B + \overline{A + B}}$

Figure 7.15 Logic diagram of a half-subtractor using only 2-input NOR gates.

The Full-Subtractor

- It subtracts one bit (B) from another bit (A), when already there is a borrow b_i from this column for the subtraction in the preceding column, and outputs the difference bit (d) and the borrow bit (b) required from the next column.

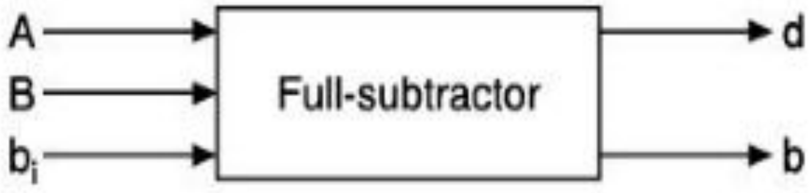
- So a full-subtractor is a combinational circuit with three inputs (A, B, b_i) and two outputs d and b.

- The two outputs present the difference and output borrow.

- The 1s and 0s for the output variables are determined from the subtraction of $A – B – b_i$.

Inputs			Difference	Borrow
A	B	b_i	d	b
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

(a) Truth table



(b) Block diagram

Figure 7.16 Full-subtractor.

$$d = \bar{A}\bar{B}b_i + \bar{A}B\bar{b}_i + A\bar{B}\bar{b}_i + ABb_i$$

$$= b_i(AB + \bar{A}\bar{B}) + \bar{b}_i(A\bar{B} + \bar{A}B)$$

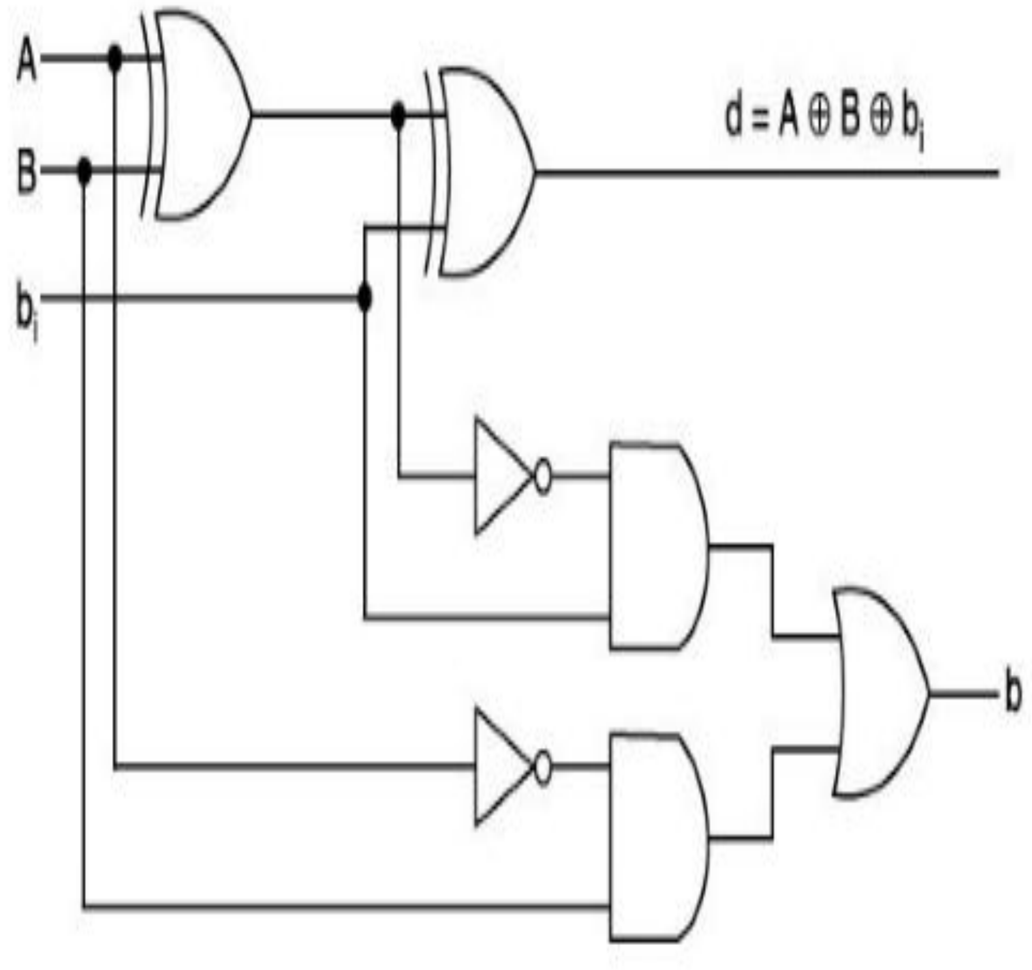
$$= b_i(\overline{A \oplus B}) + \bar{b}_i(A \oplus B) = A \oplus B \oplus b_i$$

and

$$b = \bar{A}\bar{B}b_i + \bar{A}B\bar{b}_i + \bar{A}Bb_i + ABb_i = \bar{A}B(b_i + \bar{b}_i) + (AB + \bar{A}\bar{B})b_i$$

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

A full-subtractor can, therefore, be realized using X-OR gates and AOI gates as shown in Figure 7.17.



Realization of FULL Subtractor using NAND Gates

NAND logic

$$d = A \oplus B \oplus b_i = \overline{\overline{(A \oplus B) \oplus b_i}} = \overline{\overline{(A \oplus B)\overline{(A \oplus B)b_i}} \cdot \overline{b_i\overline{(A \oplus B)b_i}}}$$

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

$$= \overline{\overline{\bar{A}B} \cdot \overline{b_i(\overline{A \oplus B})}} = \overline{\overline{B(\bar{A} + \bar{B})} \cdot \overline{b_i(\bar{b}_i + (\overline{A \oplus B})]}}$$

$$= \overline{\overline{B \cdot \overline{AB}} \cdot \overline{b_i[\overline{b_i \cdot (A \oplus B)}]}}$$

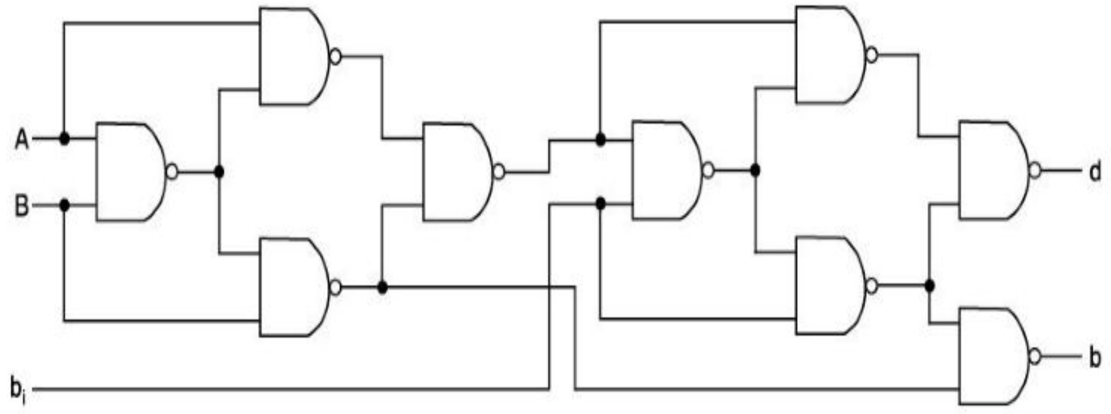


Figure 7.18 Logic diagram of a full-subtractor using only 2-input NAND gates.

Realization of FULL Subtractor using NOR Gates

NOR logic

$$d = A \oplus B \oplus b_i = \overline{\overline{(A \oplus B) \oplus b_i}}$$

$$= \overline{(A \oplus B)b_i + \overline{(A \oplus B)}\,\overline{b}_i}$$

$$= \overline{[(A \oplus B) + \overline{(A \oplus B)}\,\overline{b}_i][b_i + \overline{(A \oplus B)}\,\overline{b}_i]}$$

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

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

$$b = {}_{A}B + b_i(\overline{A \oplus B})$$

$$= {}_{A}(A + B) + (\overline{A \oplus B})[(A \oplus B) + b_i]$$

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

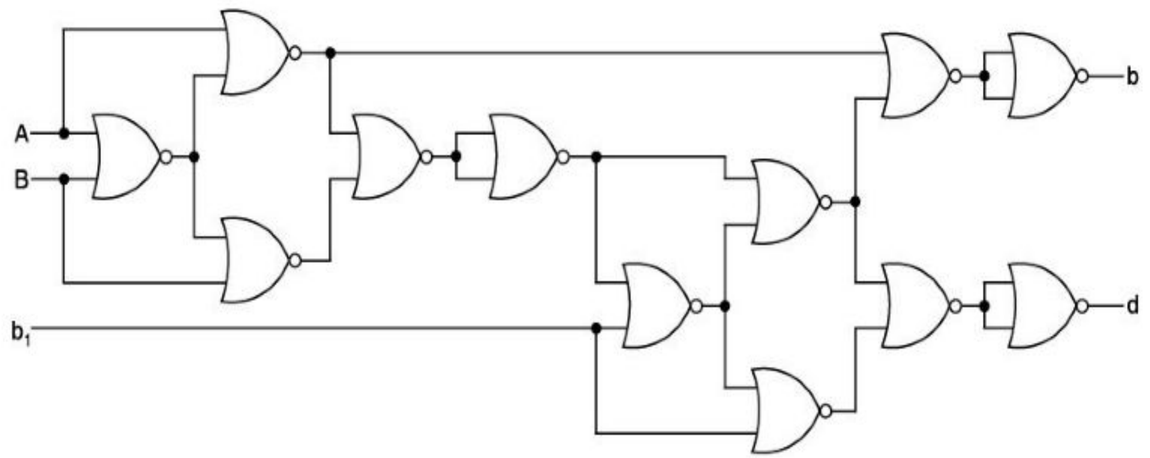


Figure 7.19 Logic diagram of a full subtractor using only 2-input NOR gates.

PARALLEL ADDER

- A binary parallel adder is a digital circuit that adds two binary numbers in parallel form and produces the arithmetic sum of those numbers in parallel form.
- It consists of full adders connected in a chain, with the output carry from each full-adder connected to the input carry of the next full-adder in the chain.
- An n-bit parallel adder requires n-full adders.

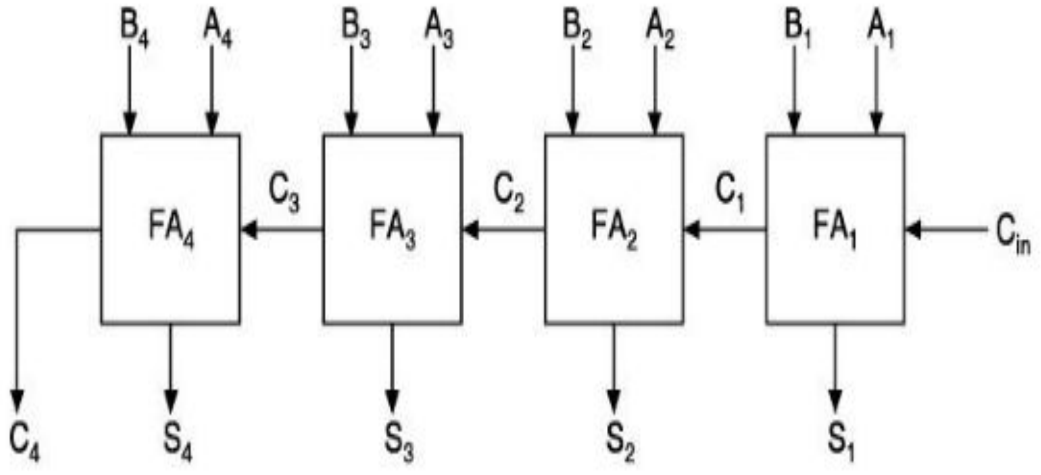
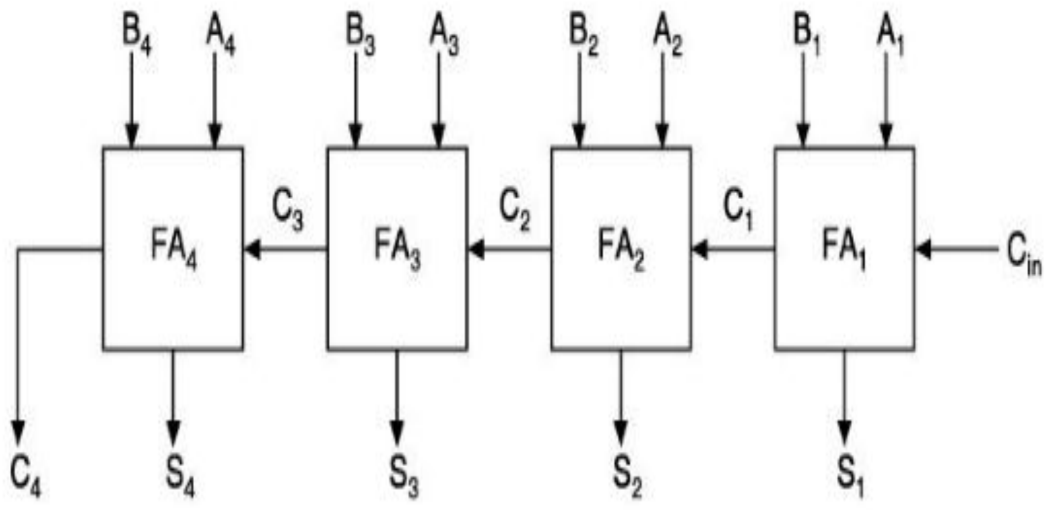


Figure 7.20 Logic diagram of a 4-bit binary parallel adder.

The Ripple Carry Adder

- The parallel adder in which the carry-out of each full-adder is the carry-in to the next most significant adder is called a ripple carry adder.



- The carry-out of each stage of ripple carry adder is connected to the carry-in of the next stage.
- The sum and carry-out bits of any stage cannot be produced, until some time after the carry-in of that stage occurs.
- This is due to the propagation delays in the logic circuitry, which lead to a time delay in the addition process.
- The carry propagation delay for each full-adder is the time between the application of the carry-in and the occurrence of the carry-out.

- That the sum (S_1) and carry-out (C_1) bits given by FA_1 are not valid, until after the propagation delay of FA_1.

- Similarly, the sum S_2 and carry-out (C_2) bits given by FA_2 are not valid until after the cumulative propagation delay of two full adders (FA_1 and FA_2), and so on.

- At each stage, the sum bit is not valid until after the carry bits in all the preceding stages are valid. In effect, carry bits must propagate or ripple through all stages before the most significant sum bit is valid.

- Thus, the total sum (the parallel output) is not valid until after the cumulative delay of all the adders.

4-BIT PARALLEL SUBTRACTOR

- The subtraction A – B can be done by taking the 2' s complement of B and adding it to A.
- The 2' s complement can be obtained by taking the 1' s complement and adding 1 to the least significant pair of bits.
- The 1' s complement can be implemented with inverters

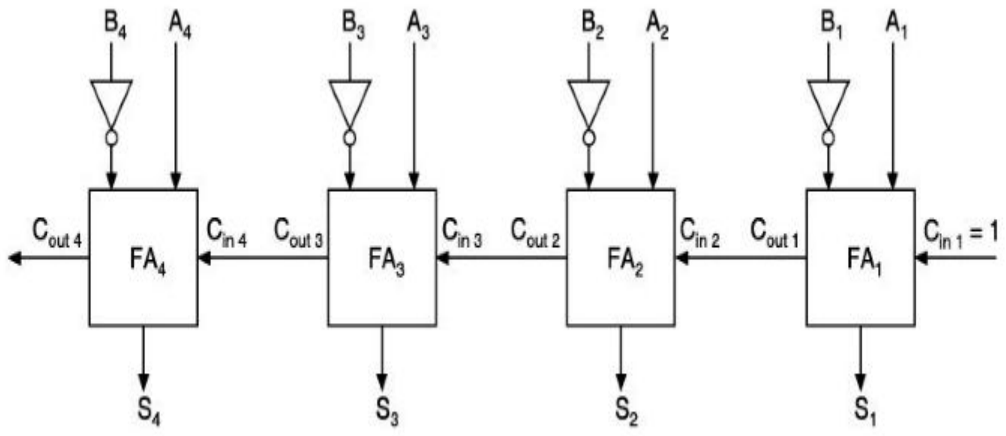


Figure 7.21 Logic diagram of a 4-bit parallel subtractor.

BINARY ADDER-SUBTRACTOR CIRCUIT

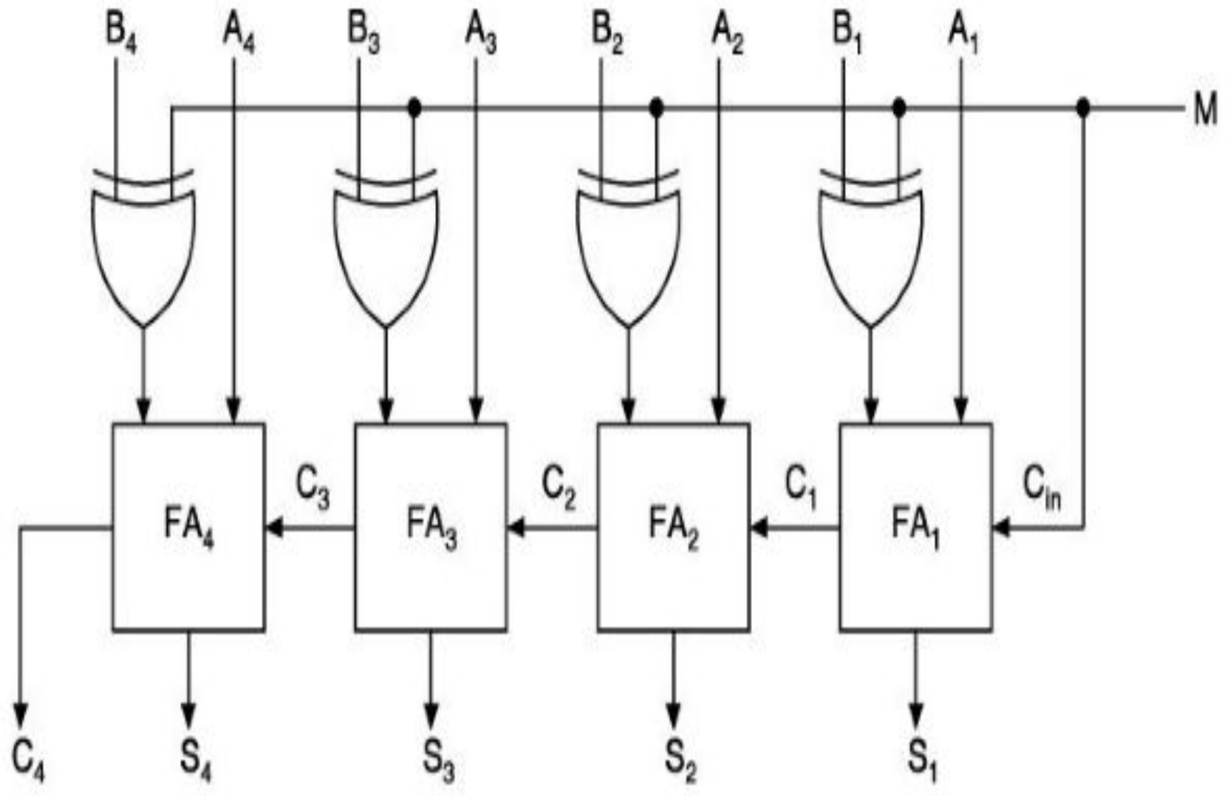


Figure 7.22 Logic diagram of a 4-bit binary adder-subtractor.

- Here the addition and subtraction operations are combined into one circuit with one common binary adder.

- This is done by including an X-OR gate with each full-adder.

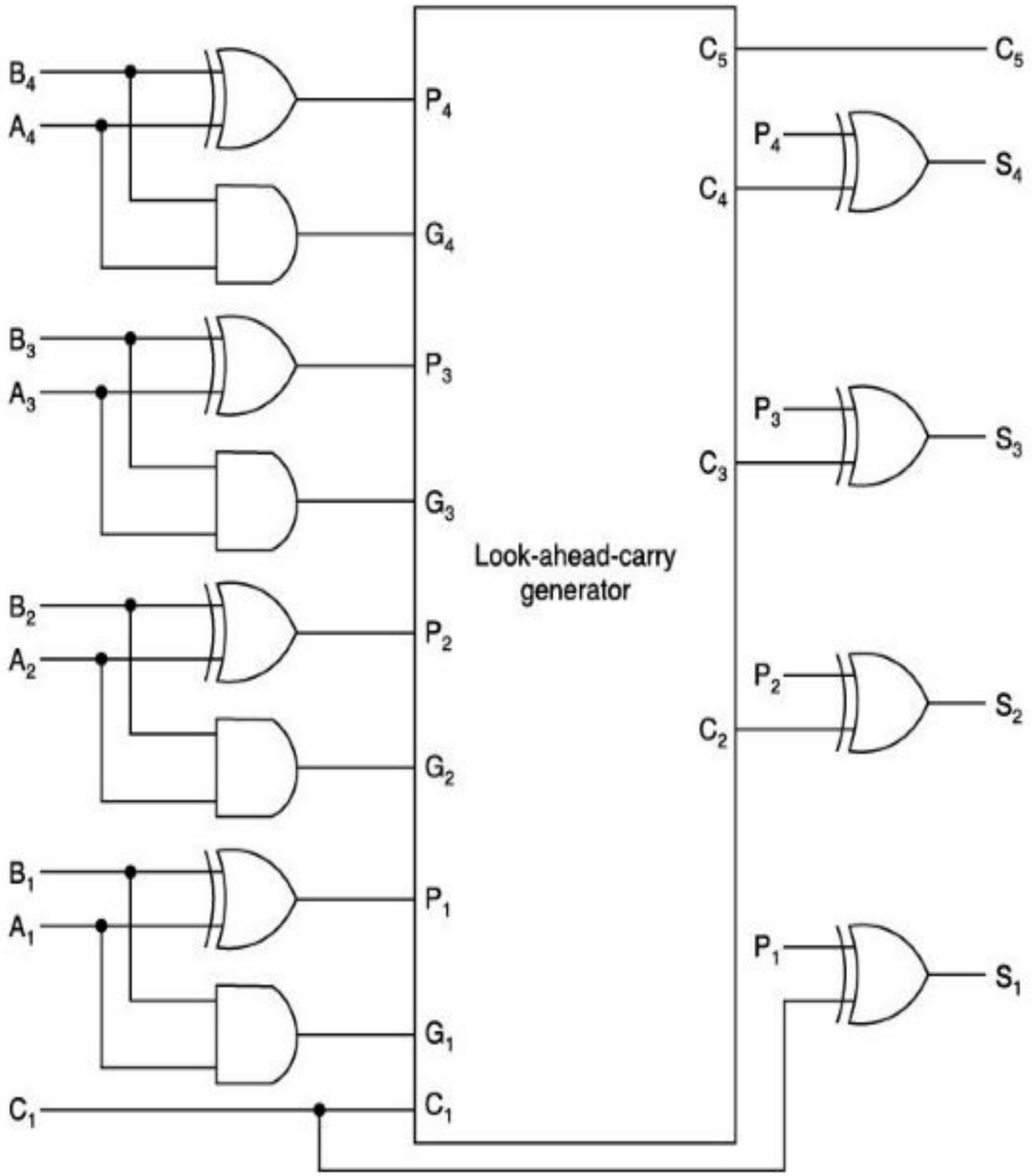
- The mode input M controls the operation. When M = 0, the circuit is an adder, and when M = 1, the circuit becomes a subtractor.

- Each X-OR gate receives input M and one of the inputs of B.

- When M = 0, we have $B \oplus 0 = B$. The full-adder receives the value of B, the input carry is 0 and the circuit performs A + B.

- When M = 1, we have $B \oplus 1 = \overline{B}$ and C1 = 1. The B inputs are complemented and a 1 is added through the input carry. The circuit performs the operation A plus the 2’ s complement of B.

THE LOOK-AHEAD-CARRY ADDER



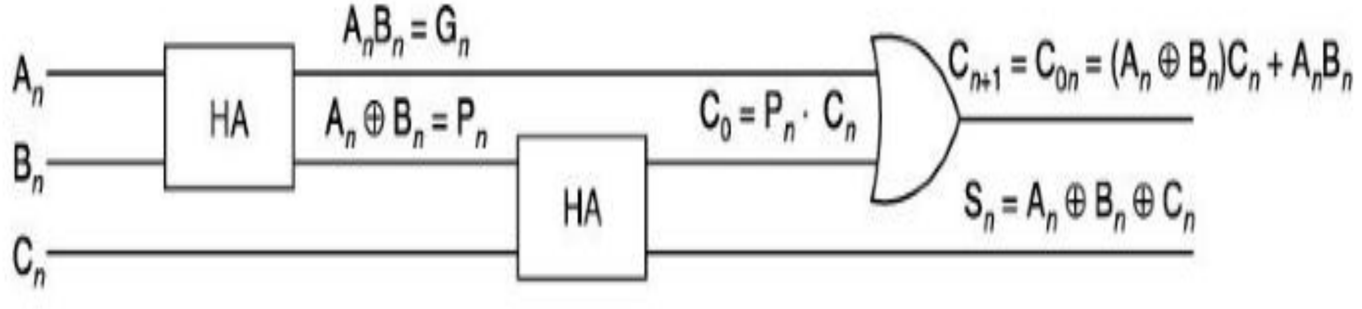
- In the case of the parallel adder, the speed with which an addition can be performed is governed by the time required for the carries to propagate or ripple through all of the stages of the adder.

- The look-ahead-carry adder speeds up the process by eliminating this ripple carry delay.

- It examines all the input bits simultaneously and also generates the carry-in bits for all the stages simultaneously.

- The method of speeding up the addition process is based on the two additional functions of the full-adder, called the *carry generate* and carry propagate functions.

- Consider one full adder stage; say the nth stage of a parallel adder.



$A_n B_n = G_n$
A_n
B_n
C_n
HA
$A_n \oplus B_n = P_n$
HA
$C_0 = P_n \cdot C_n$
$C_{n+1} = C_{0n} = (A_n \oplus B_n)C_n + A_n B_n$
$S_n = A_n \oplus B_n \oplus C_n$

Figure 7.23 A full adder (nth stage of a parallel adder).

- It is made of two half-adders and that the half-adder contains an X-OR gate to produce the sum and an AND gate to produce the carry.

- If both the bits A_n and B_n are 1s, a carry has to be generated in this stage regardless of whether the input carry C_{in} is a 0 or a 1. This is called <u>generated carry</u>, expressed as

$$G_n = A_n . B_n$$

- There is another possibility of producing a carry out.
- X-OR gate inside the half-adder at the input produces an intermediary sum bit, P_n which is expressed as
- $P_n = A_n \quad \oplus \; {}_n$.
- Next P_n and C_n are added using the X-OR gate inside the second half adder to produce the final sum bit
- $S_n = P_n \quad C_n \oplus A_n \quad \oplus \quad \oplus C_n$ and
- output carry $C_0 = P_n \cdot C_n = (A_n \oplus B_n).C_n$ which becomes input carry for the (n + 1) th stage.

Consider the case of both P_n and C_n being 1. The input carry C_n has to be propagated to the output only if P_n is 1. If P_n is 0, even if C_n is 1, the AND gate in the second half-adder will inhibit C_n. We may thus call P_n as the propagated carry as this is associated with enabling propagation of C_n to the carry output of the *n*th stage which is denoted as C_{n+1} or C_{0n}. So, we can say that the carryout of the *n*th stage is 1 when either $G_n = 1$ or $P_n \,|.\, C_n = 1$ or both G_n and $P_n \cdot C_n$ are equal to 1.

For the final sum and carry outputs of the *n*th stage, we get the following Boolean expressions.

$$S_n = P_n \oplus C_n \text{ where } P_n = A_n \oplus B_n$$

$$C_{0n} = C_{n+1} = G_n + P_nC_n \text{ where } G_n = A_n \,.\, B_n$$

The carry-in to each stage is the carry-out of the previous stage.

Based on these, the expressions for the carry-outs of various full adders are

$C_1 = G_0 + P_0 . C_0$

$C_2 = G_1 + P_1 . C_1 = G_1 + P_1 . G_0 + P_1 . P_0 .C_0$

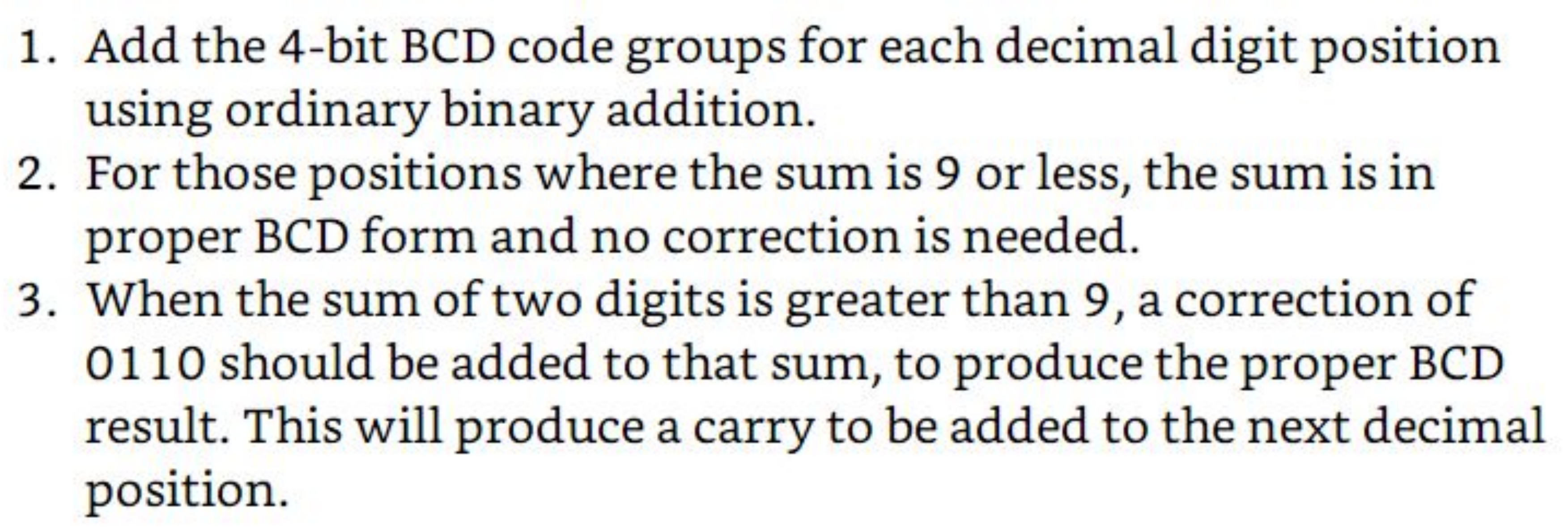
$C_3 = G_2 + P_2 .C_2 = G_2 + P_2 . G_1 + P_2 . P_1 .G_0 + P_2 . P_1 . P_0 . C_0$

$C_4 = G_3 + P_3 .C_3 = G_3 + P_3 . G_2 + P_3 . P_2 . G_1 + P_3 . P_2 . P_1 .G_0 + P_3 . P_2 . P_1 .P_0 . C_0$

The general expression for *n* stages designated as 0 through ($n - 1$) would be

$C_n = G_{n-1} + P_{n-1} . C_{n-1} = G_{n-1} + P_{n-1} . G_{n-2} + P_{n-1} . P_{n-2} .G_{n-3} + \dots + P_{n-1} . \dots P_0 . C_0$

BCD ADDER

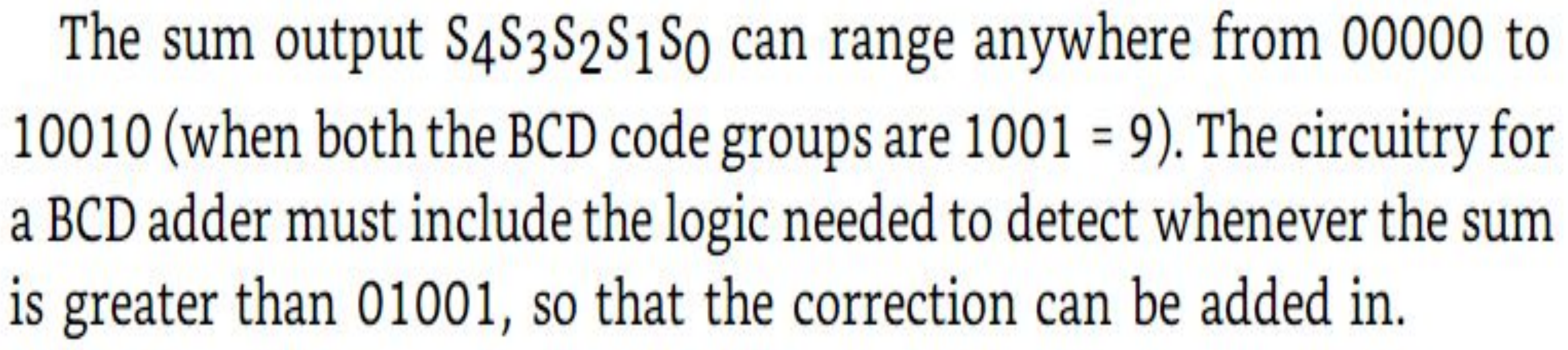


1. Add the 4-bit BCD code groups for each decimal digit position using ordinary binary addition.
2. For those positions where the sum is 9 or less, the sum is in proper BCD form and no correction is needed.
3. When the sum of two digits is greater than 9, a correction of 0110 should be added to that sum, to produce the proper BCD result. This will produce a carry to be added to the next decimal position.

- **A BCD Adder circuit must be able to do the following.**
- **1.Add two 4-bit BCD code groups, using straight binary addition.**
- **2.Determine, if the sum of this addition is greater than 1001 (decimal 9); if it is, add 0110 (decimal 6) to this sum and generate a carry to the next decimal position.**

- The first requirement is easily met by using a 4-bit binary parallel adder

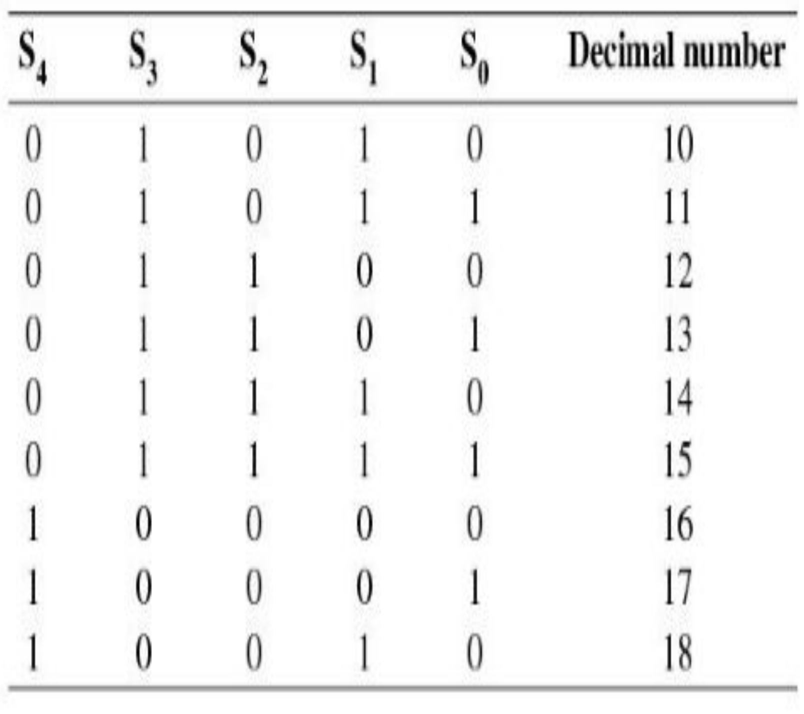
- If the two BCD code groups $A_3A_2A_1A_0$ and $B_3B_2B_1B_0$ are applied to a 4-bit parallel adder, the adder will output $S_4S_3S_2S_1S_0$, where S_4 is actually C_4, the carry-out of the MSB bits.



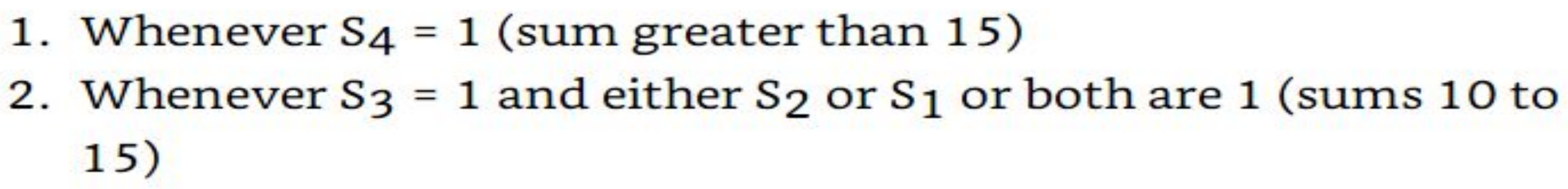
The sum output $S_4S_3S_2S_1S_0$ can range anywhere from 00000 to 10010 (when both the BCD code groups are 1001 = 9). The circuitry for a BCD adder must include the logic needed to detect whenever the sum is greater than 01001, so that the correction can be added in.

• Cases where sum greater than 9

Table 7.1

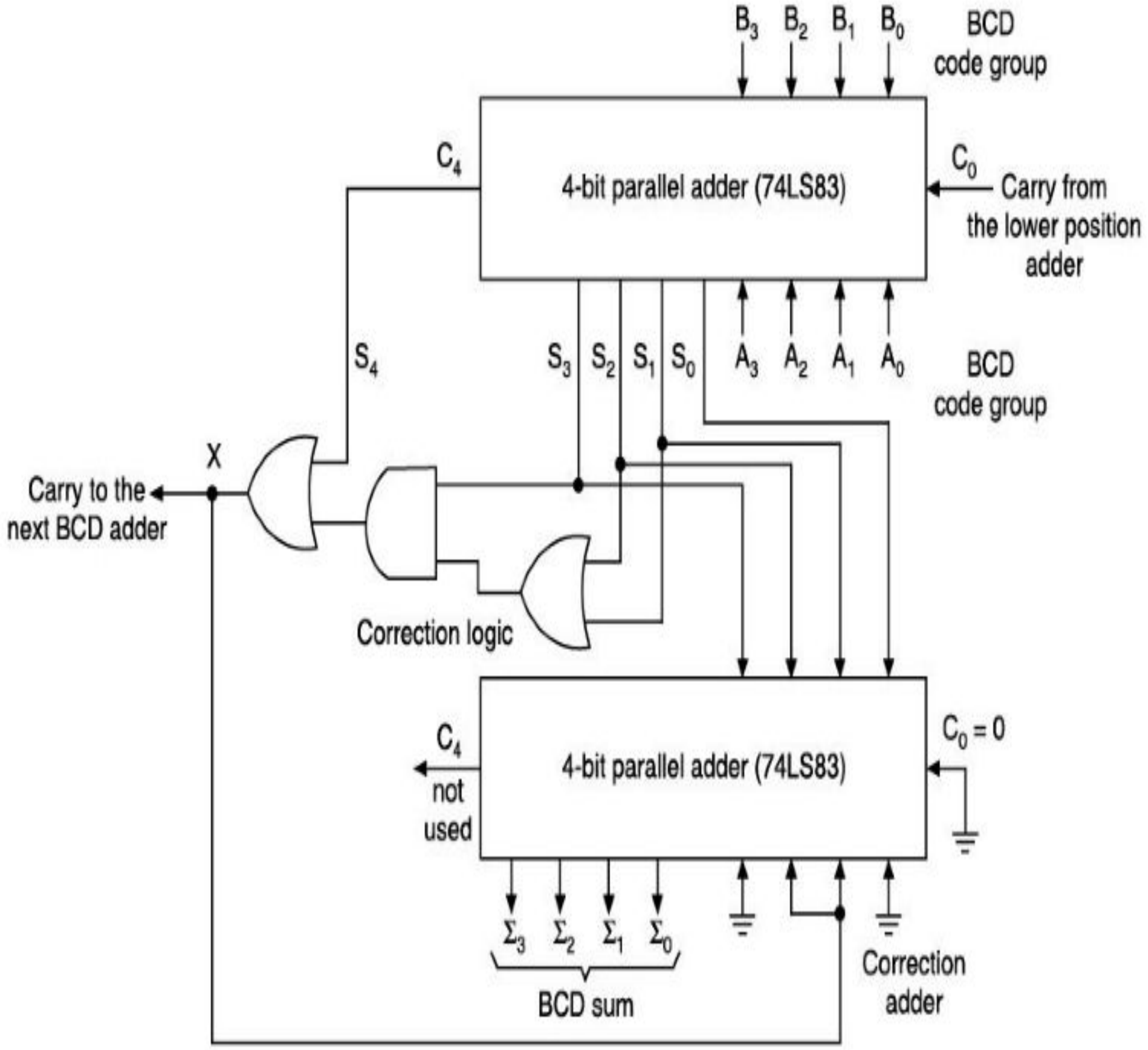


S_4	S_3	S_2	S_1	S_0	Decimal number
0	1	0	1	0	10
0	1	0	1	1	11
0	1	1	0	0	12
0	1	1	0	1	13
0	1	1	1	0	14
0	1	1	1	1	15
1	0	0	0	0	16
1	0	0	0	1	17
1	0	0	1	0	18



1. Whenever $S_4 = 1$ (sum greater than 15)
2. Whenever $S_3 = 1$ and either S_2 or S_1 or both are 1 (sums 10 to 15)

This condition can be expressed as **X = S4 + S3(S2 + S1)**


B_3 B_2 B_1 B_0
BCD code group
C_4
4-bit parallel adder (74LS83)
C_0
Carry from the lower position adder
S_4
S_3 S_2 S_1 S_0
A_3 A_2 A_1 A_0
BCD code group
X
Carry to the next BCD adder
Correction logic
C_4
not used
4-bit parallel adder (74LS83)
$C_0 = 0$
Σ_3 Σ_2 Σ_1 Σ_0
BCD sum
Correction adder

CODE CONVERTERS

- A code converter is a logic circuit whose inputs are bit patterns representing numbers (or characters) in one code and whose outputs are the corresponding representations in a different code.

Design of a 4-bit Binary-to-Gray Code Converter

- The input to the 4-bit binary-to-Gray code converter circuit is a 4-bit binary and the output is a 4-bit Gray code.

$$G_4 = B_4$$

$$G_3 = B_4 \oplus B_3$$

$$G_2 = B_3 \oplus B_2$$

$$G_1 = B_2 \oplus B_1$$

4-bit binary				4-bit Gray			
B_4	B_3	B_2	B_1	G_4	G_3	G_2	G_1
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

(a) Conversion table

$$G_4 = \sum m(8, 9, 10, 11, 12, 13, 14, 15)$$

$$G_3 = \sum m(4, 5, 6, 7, 8, 9, 10, 11)$$

$$G_2 = \sum m(2, 3, 4, 5, 10, 11, 12, 13)$$

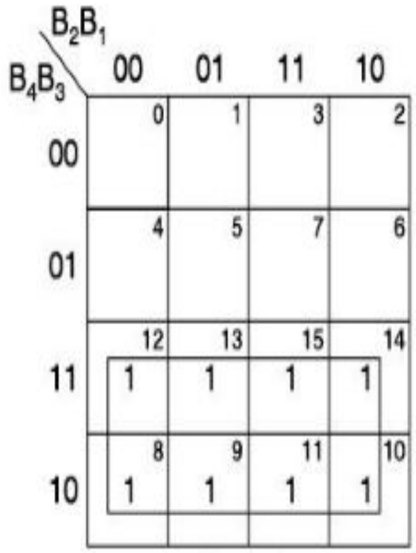
$$G_1 = \sum m(1, 2, 5, 6, 9, 10, 13, 14)$$

$$G4 = B4$$

$$G3 = B4 \oplus B3 = \overline{B4}B3 + B4\overline{B3}$$

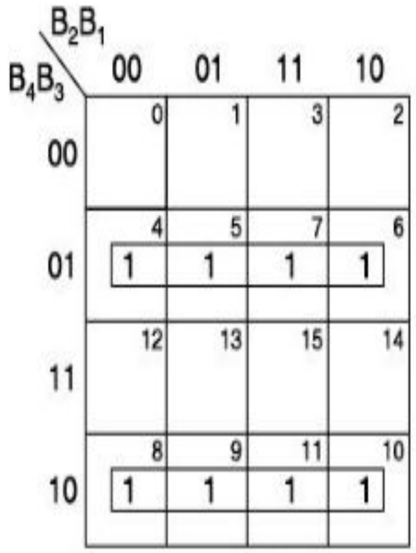
$$G2 = B3 \oplus B2 = \overline{B3}B2 + B3\overline{B2}$$

$$G1 = B2 \oplus B1 = \overline{B2}B1 + B2\overline{B1}$$



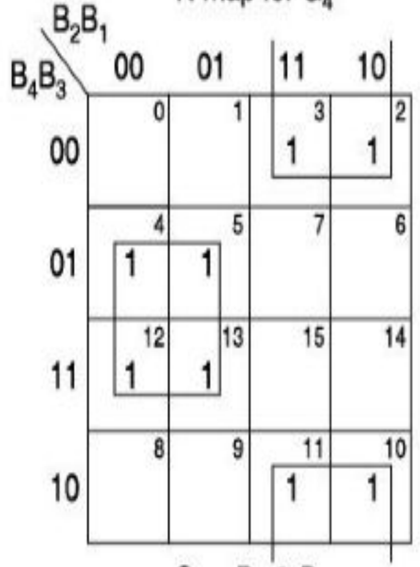
$G_4 = B_4$

K-map for G_4



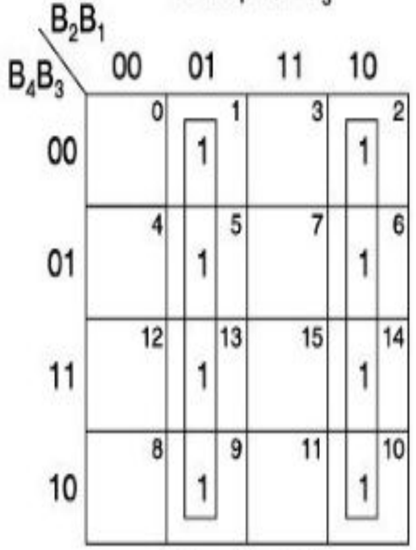
$G_3 = B_4 \oplus B_3$

K-map for G_3



$G_2 = B_3 \oplus B_2$

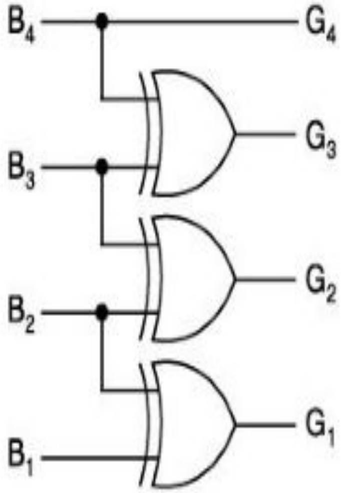
K-map for G_2



$G_1 = B_2 \oplus B_1$

K-map for G_1

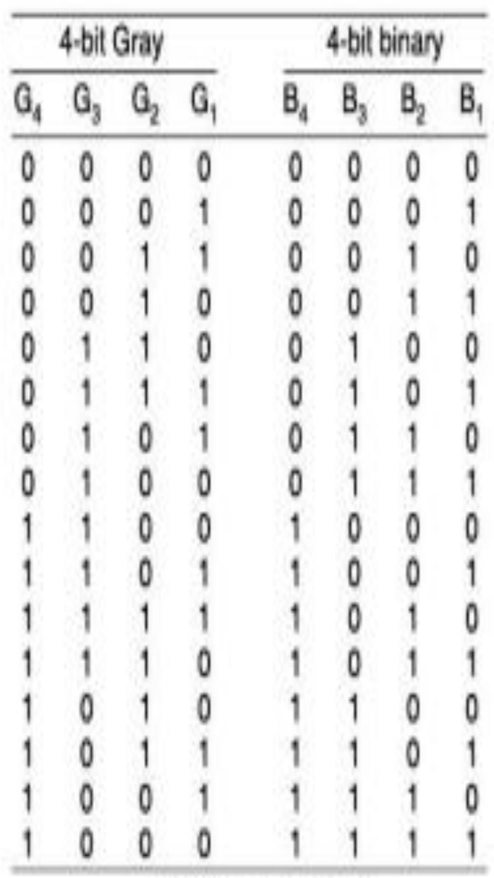
(b) K-maps



(c) Logic diagram

Design of a 4-bit Gray-to-Binary Code Converter

- The input to the 4-bit Gray-to-binary code converter circuit is a 4-bit Gray code and the output is a 4-bit binary.



4-bit Gray				4-bit binary			
G_4	G_3	G_2	G_1	B_4	B_3	B_2	B_1
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	1	0	0	1	0
0	0	1	0	0	0	1	1
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	1
0	1	0	1	0	1	1	0
0	1	0	0	0	1	1	1
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	1
1	1	1	1	1	0	1	0
1	1	1	0	1	0	1	1
1	0	1	0	1	1	0	0
1	0	1	1	1	1	0	1
1	0	0	1	1	1	1	0
1	0	0	0	1	1	1	1

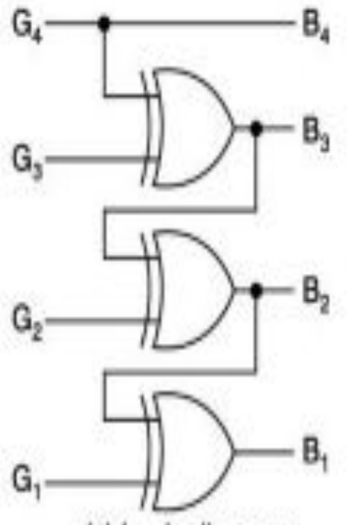
(a) Conversion table

$B4 = G4$

$B3 = G4 \oplus G3$

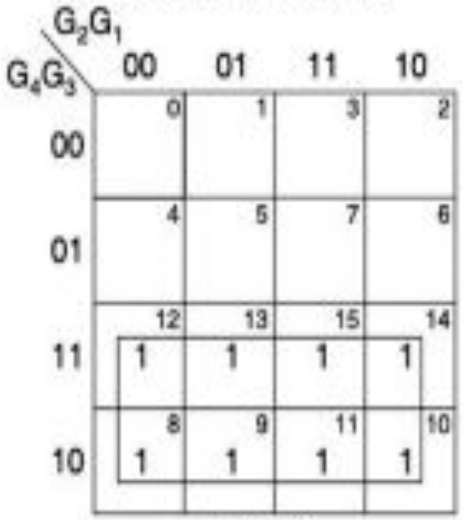
$B2 = B3 \oplus G2$

$B1 = B2 \oplus G1$



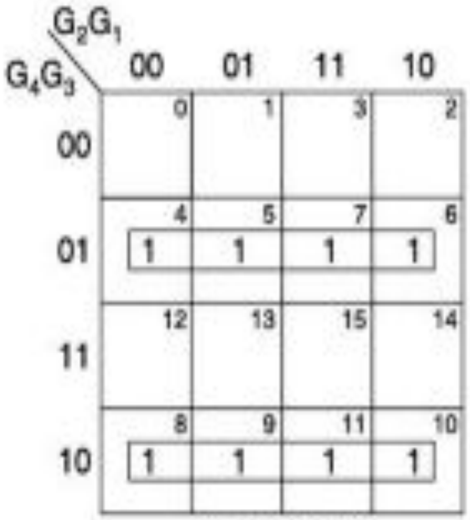
(c) Logic diagram

(a) Conversion table



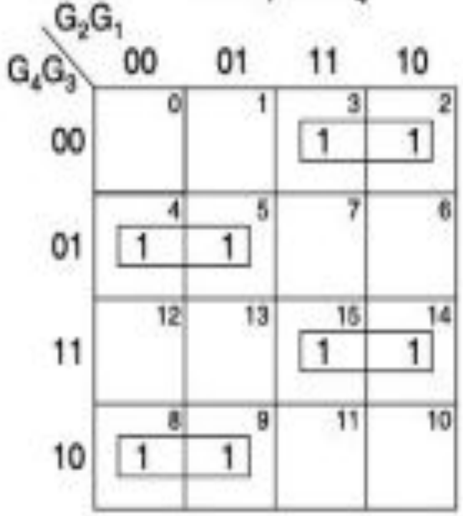
$B_4 = G_4$

K-map for B_4



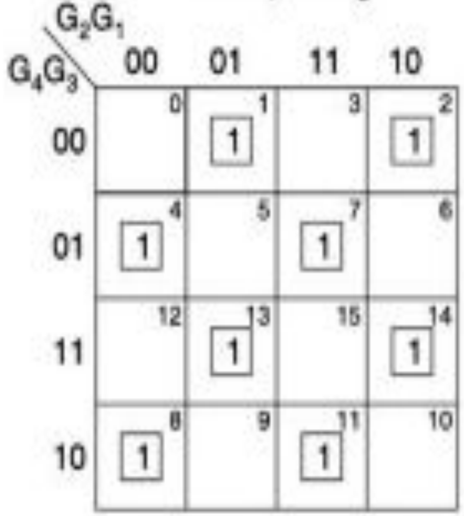
$B_3 = G_4 \oplus G_3$

K-map for B_3



$B_2 = G_4 \oplus G_3 \oplus G_2$

K-map for B_2



$B_1 = G_4 \oplus G_3 \oplus G_2 \oplus G_1$

K-map for B_1

(b) K-maps

DECODERS

- A decoder is a logic circuit that converts an N-bit binary input code into M output lines such that only one output line is activated for each one of the possible combinations of inputs.

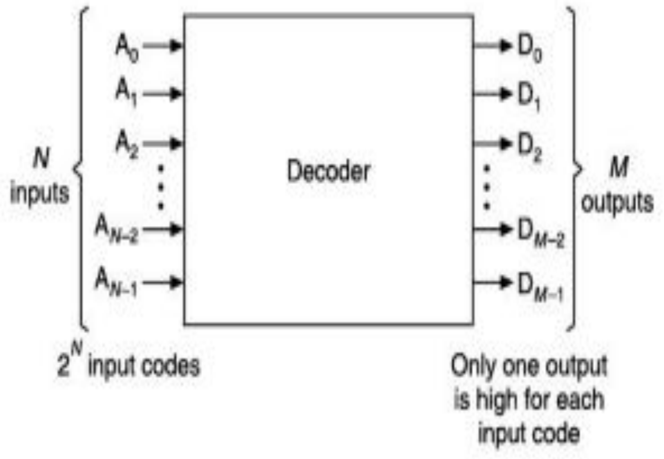
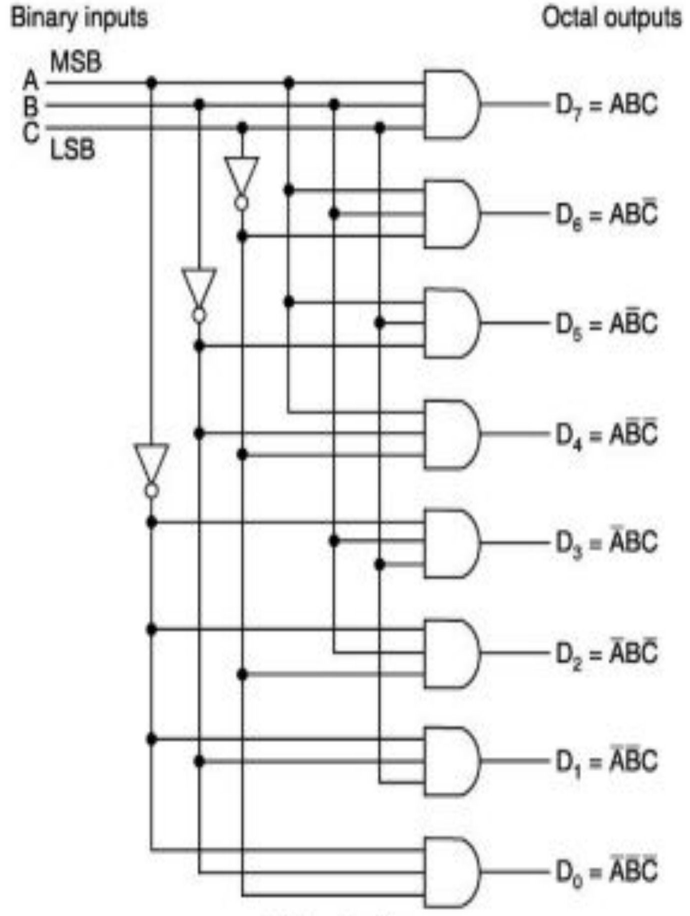


Figure 7.68 General block diagram of a decoder.

- Since each of the N inputs can be a 0 or a 1, there are 2^N possible input combinations or codes.

- For each of these input combinations, only one of the M outputs will be active (HIGH), all the other outputs will remain inactive (LOW).

3-Line-to-8-Line Decoder



(a) Logic diagram

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

(b) Truth table

Figure 7.69 3-line to 8-line decoder.

- It can be called a **3-line to 8-line** decoder because it has three input lines and eight output lines.

- It is also called a **binary-to-octal** decoder because it takes a 3-bit binary input code and activates one of the eight (octal) outputs corresponding to that code.

- It is also referred to as a **1-of-8 decoder** because only one of the eight outputs is activated at one time.

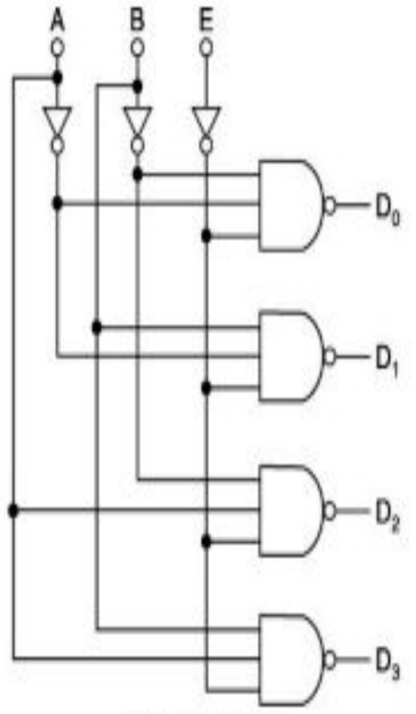
- <u>Enable Inputs</u>

- Some decoders have one or more ENABLE inputs that are used to control the operation of the decoder.

- For example, in the 3-line to 8-line decoder, if a common ENABLE line is connected to the fourth input of each gate, a particular output as determined by the A, B, C input code will go HIGH only when the ENABLE line is held HIGH.

- When the ENABLE is held LOW, however, all the outputs will be forced to the LOW state regardless of the levels at the A, B, and C inputs.

2-Line-to-4-Line Decoder



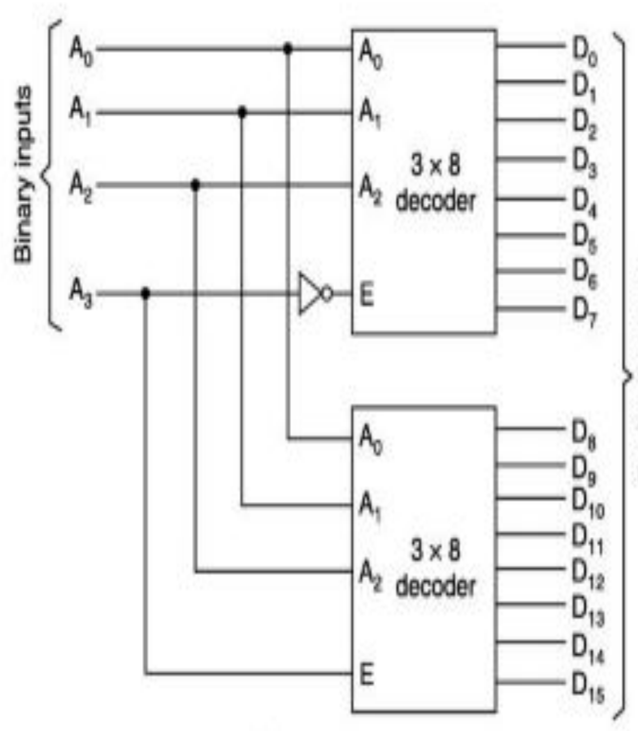
(a) Logic diagram

E	A	B	D_0	D_1	D_2	D_3
1	×	×	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

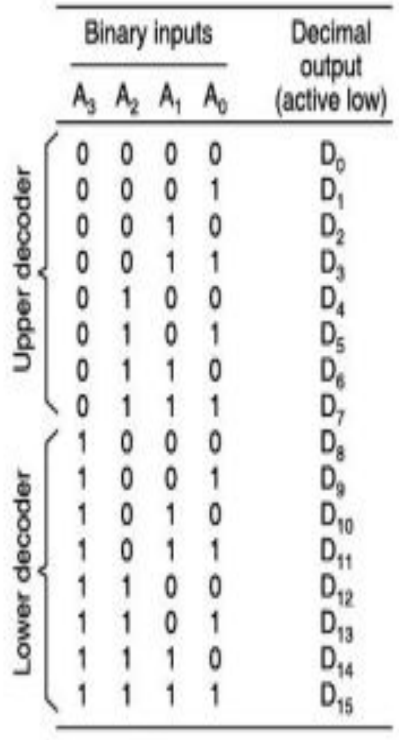
(b) Truth table

Figure 7.70 2 line-to-4 line decoder with NAND gates.

4-to-16 Decoder from Two 3-to-8 Decoders



(a) Logic diagram



	Binary inputs				Decimal output (active low)
	A_3	A_2	A_1	A_0	
Upper decoder	0	0	0	0	D_0
	0	0	0	1	D_1
	0	0	1	0	D_2
	0	0	1	1	D_3
	0	1	0	0	D_4
	0	1	0	1	D_5
	0	1	1	0	D_6
	0	1	1	1	D_7
Lower decoder	1	0	0	0	D_8
	1	0	0	1	D_9
	1	0	1	0	D_{10}
	1	0	1	1	D_{11}
	1	1	0	0	D_{12}
	1	1	0	1	D_{13}
	1	1	1	0	D_{14}
	1	1	1	1	D_{15}

(b) Function table

Figure 7.72 Connecting two 74138 3-to-8 decoders to obtain a 4-to-16 decoder.

- The most significant input bit A3 is connected through an inverter to on the upper decoder (for D0 through D7) and directly to E on the lower decoder (for D8 through D15).
- Thus, when A3 is LOW, the upper decoder is enabled and the lower decoder is disabled.
- The bottom decoder outputs all 0s, and top 8 outputs generate minterms.
- When A3 is HIGH, the lower decoder is enabled and the upper decoder is disabled.
- The bottom decoder outputs generate minterms 1000 to 1111 while the outputs of the top decoder are all 0s.

ENCODERS

- An encoder is a device whose inputs are decimal digits and/ or alphabetic characters and whose outputs are the coded representation of those inputs.

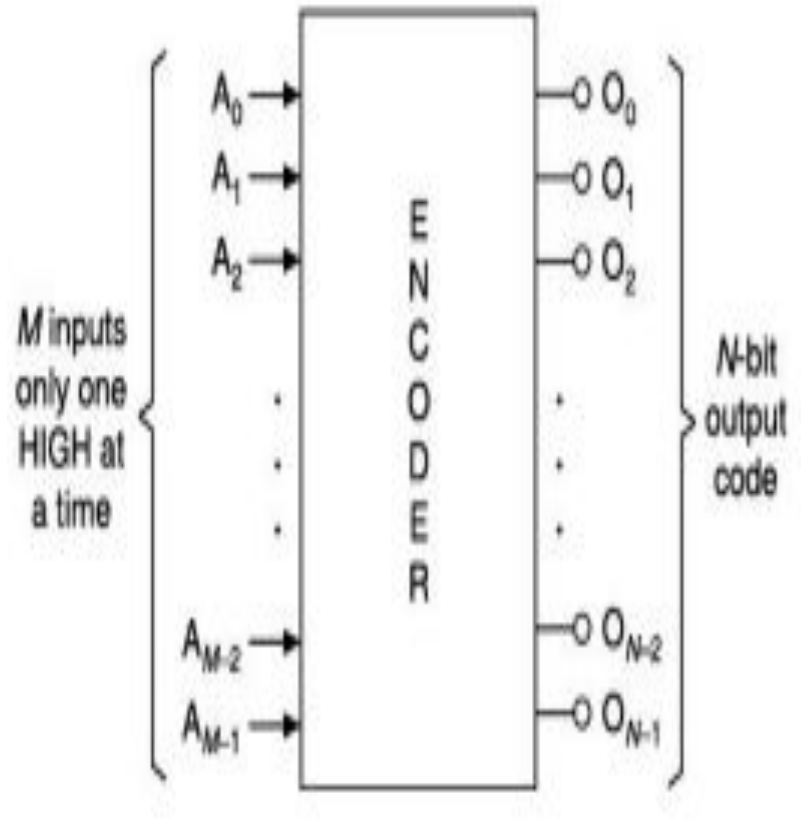
- i.e. an encoder is a device which converts familiar numbers or symbols into coded format.

- An encoder may be said to be a combinational logic circuit that performs the ‘reverse’ operation of the decoder.

- The opposite of the decoding process is called encoding.

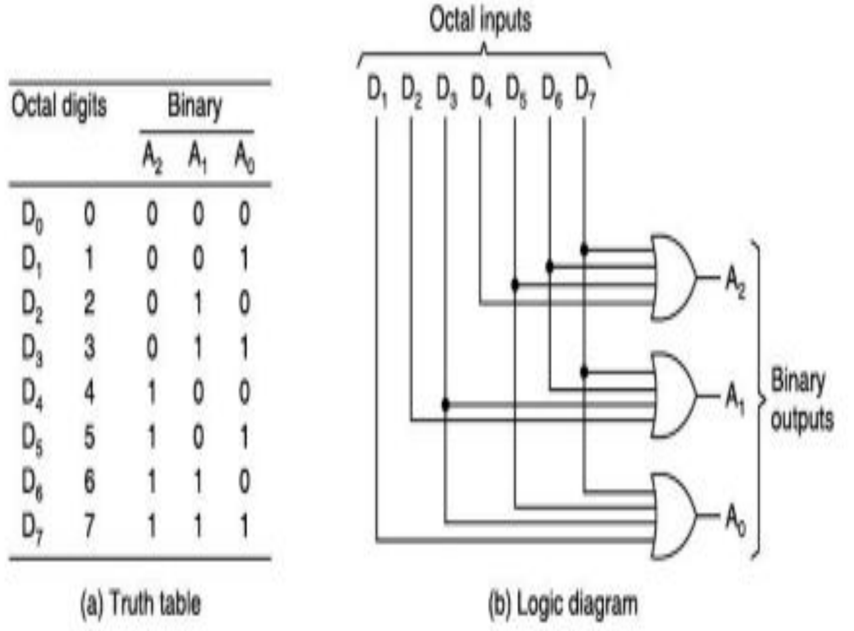
- An encoder has a number of input lines, only one of which is activated at a given time, and produces an N-bit output code depending on which input is activated.

Block diagram of encoder.



Octal-to-Binary Encoder

- An octal-to-binary encoder (8-line to 3-line encoder) accepts 8 input lines and produces a 3-bit output code corresponding to the activated input.



Octal digits		Binary		
		A_2	A_1	A_0
D_0	0	0	0	0
D_1	1	0	0	1
D_2	2	0	1	0
D_3	3	0	1	1
D_4	4	1	0	0
D_5	5	1	0	1
D_6	6	1	1	0
D_7	7	1	1	1

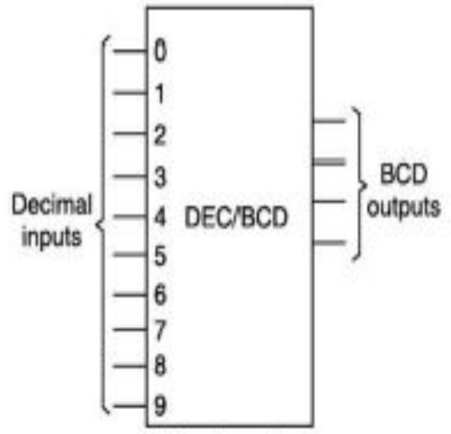
(a) Truth table

(b) Logic diagram

Figure 7.62 Octal-to-binary encoder.

- From the truth table, we see that A2 is a 1 if any of the digits D4 or D5 or D6 or D7 is a 1.
- Therefore, A2 = D4 + D5 + D6 + D7
- Similarly, A1 = D2 + D3 + D6 + D7 and
- A0 = D1 + D3 + D5 + D7
- D_0 is not present in any of the expressions. So, D_0 is a don't care.

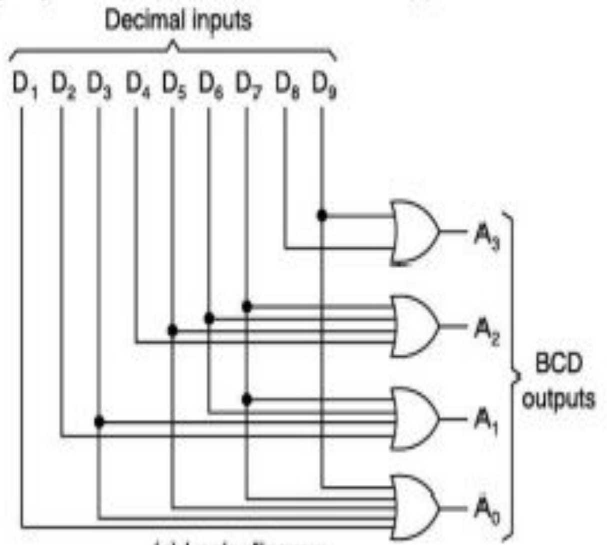
Decimal-to-BCD Encoder



(a) Logic symbol

Decimal inputs		Binary A_3	A_2	A_1	A_0
D_0	0	0	0	0	0
D_1	1	0	0	0	1
D_2	2	0	0	1	0
D_3	3	0	0	1	1
D_4	4	0	1	0	0
D_5	5	0	1	0	1
D_6	6	0	1	1	0
D_7	7	0	1	1	1
D_8	8	1	0	0	0
D_9	9	1	0	0	1

(b) Truth table



(c) Logic diagram

Figure 7.63 Decimal-to-BCD encoder.

- This type of encoder has 10 inputs— one for each decimal digit, and 4 outputs corresponding to the BCD code.

- This is a basic 10-line to 4-line encoder.

- A3 = D8 + D9
- A2 = D4 + D5 + D6 + D7
- A1 = D2 + D3 + D6 + D7
- A0 = D1 + D3 + D5 + D7 + D9

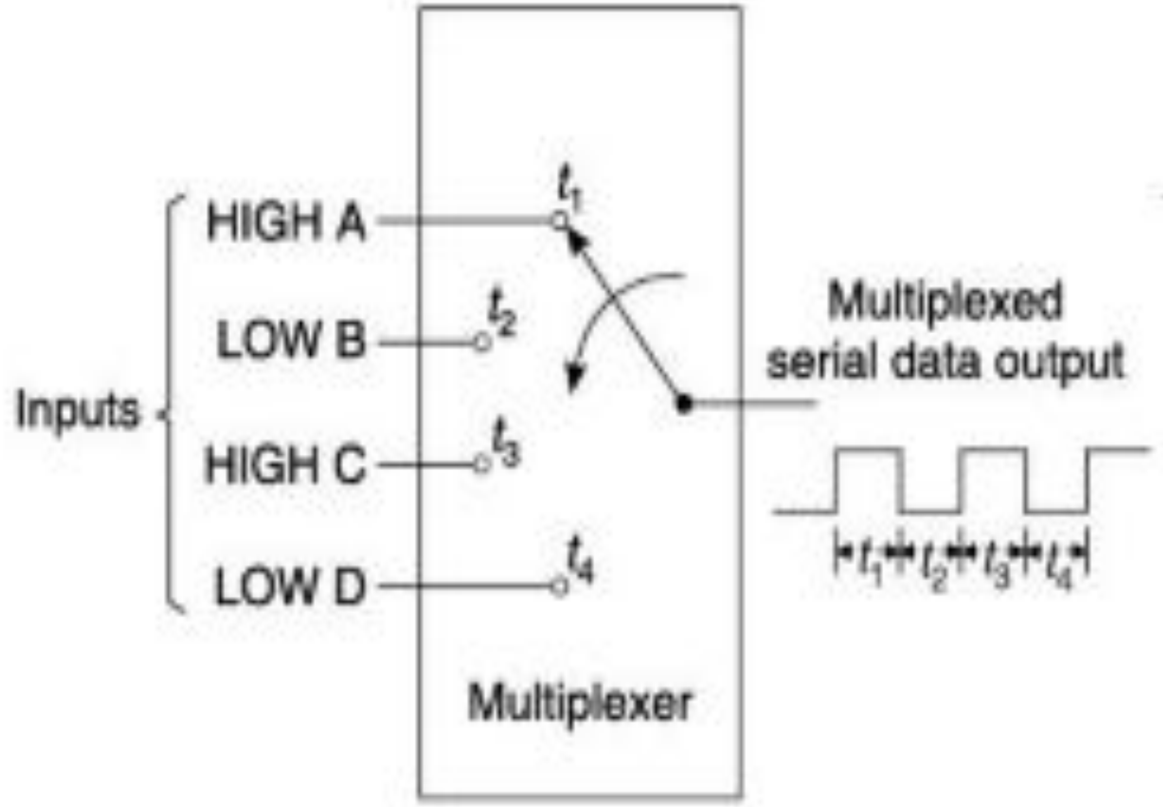
Multiplexer (Mux)

- Multiplexing means sharing. It is the process of switching information from several lines on to a single line in a specified sequence.

- A multiplexer or data selector is a logic circuit that accepts several data inputs and allows only one of them to get through to the output.

- It is an **N-to-1 device**.

4-to-1 Multiplexer



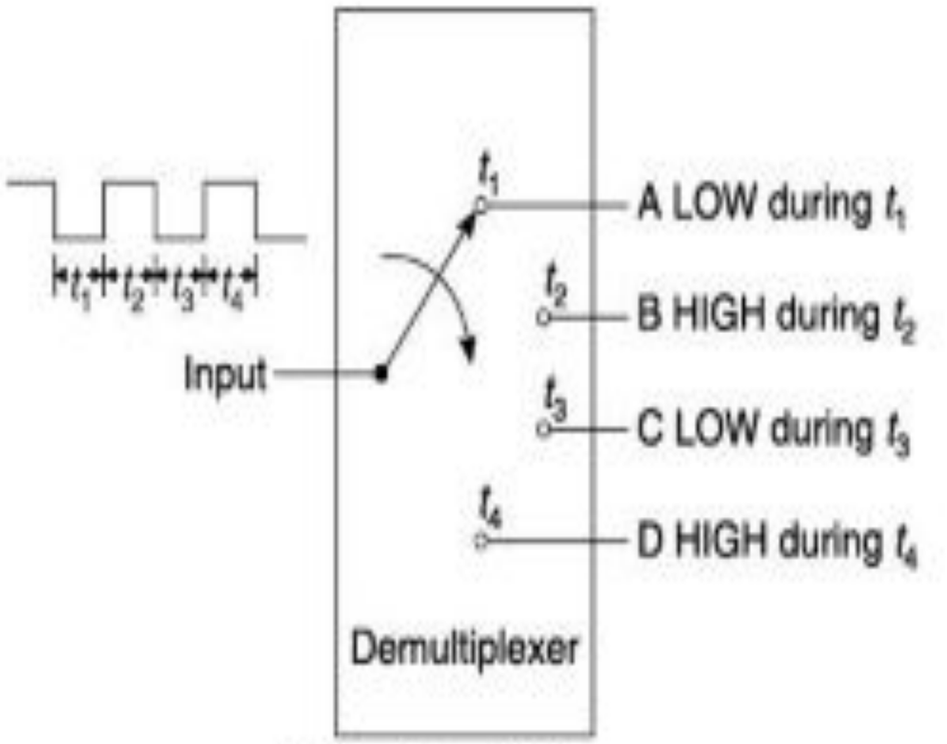
(a) The multiplexer

- if the switch is connected to input A for time t1, to input B for time t2, to input C for time t3 and to input D for time t4, the output of the multiplexer will be as shown in the figure.

DeMultiplexer (DeMux)

- Demultiplexing operation is the inverse of multiplexing.
- Demultiplexing is the process of switching information from one input line on to several output lines.
- A demultiplexer is a digital circuit that takes a single input and distributes it over several outputs.
- It is a **1-to-N device.**

1-to-4 Demultiplexer

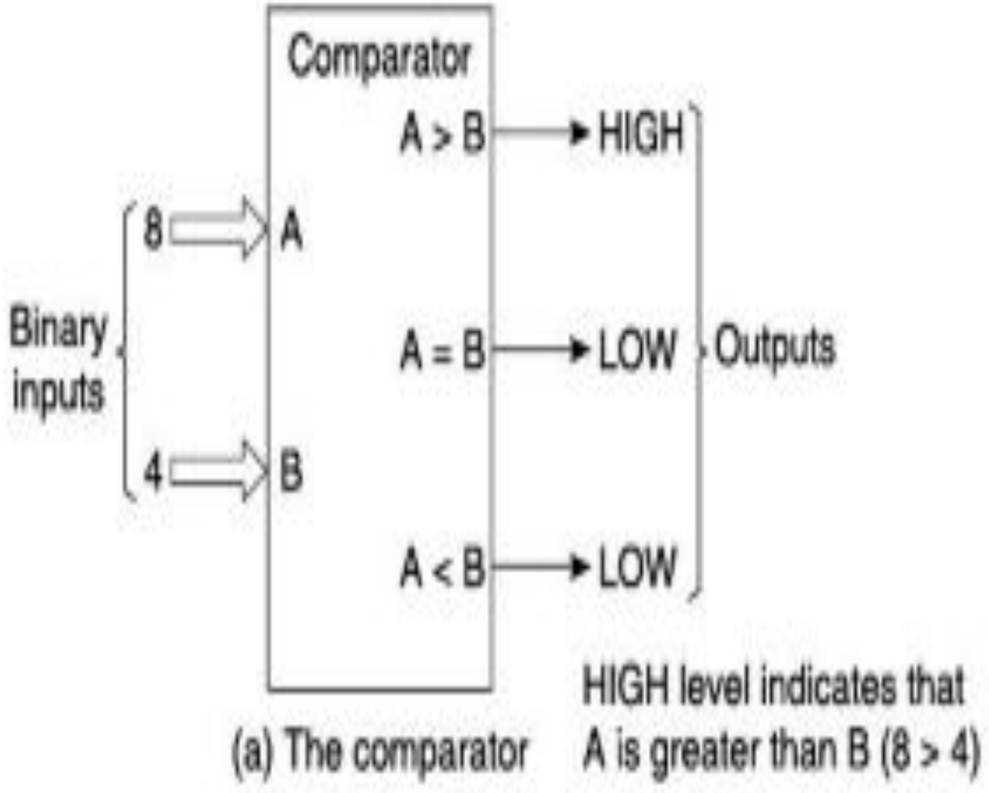


(b) The demultiplexer

• if the switch is connected to input A for time

Magnitude comparator

- Magnitude Comparator is a logic circuit used to compare two quantities and give an output signal indicating whether the two input quantities are equal or not, and if not, which of them is greater.



(a) The comparator

- The binary representations of the quantities A and B to be compared are applied as inputs to the comparator.

- One of the outputs, A < B, A = B or A > B goes HIGH, depending on the magnitudes of the input quantities.

- The figure illustrates comparison of 8 and 4, and the result is HIGH (8 > 4).