2 Combinational Circuit

2.1 INTRODUCTION

Introduction to logic design:

Combinational circuits are those whose current input decides the current output. These circuits process the operations which can be fully logically specified with the help of Boolean functions. These circuits are formed of logic gates and input/output variables.

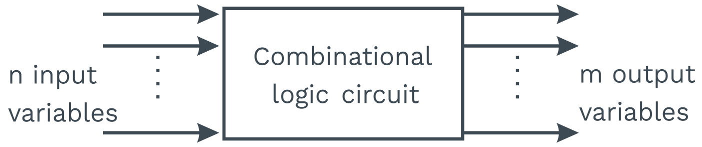
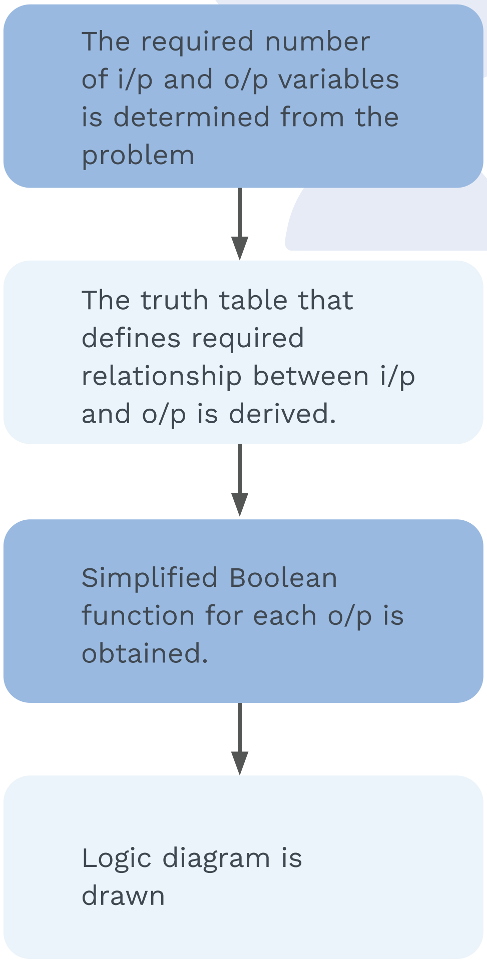


Fig. 2.1 Block Diagram of Combinational Circuit

A block diagram of a combinational circuit is shown in figure 2.1

Design procedure:



The required number of i/p and o/p variables is determined from the problem

The truth table that defines required relationship between i/p and o/p is derived.

Simplified Boolean function for each o/p is obtained.

Logic diagram is drawn

2.2 ADDER

Half adder:

Note:

- OR-AND realisation is equivalent to NOR-NOR realisation. (both of 2 levels)
  f(A, B, C, D) = (A + B) (C + D)

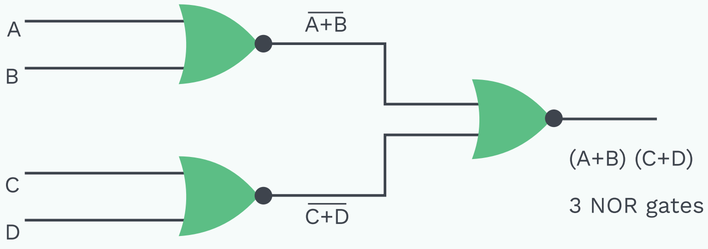


Fig. 2.2 NOR-NOR Realisation

Note:

- AND-OR realisation is equivalent to NAND-NAND realisation. f(A, B, C, D) = AB + CD

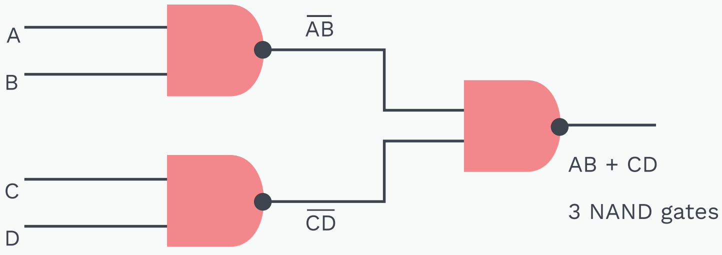


Fig. 2.3 NANDc-NAND Realisation

A half adder adds the 2 inputs and produces sum and carry bits.

Truth table			
A	**B**	**Sum**	**Carry**
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Table 2.1 Truth Table for Half Adder

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

Carry = AB

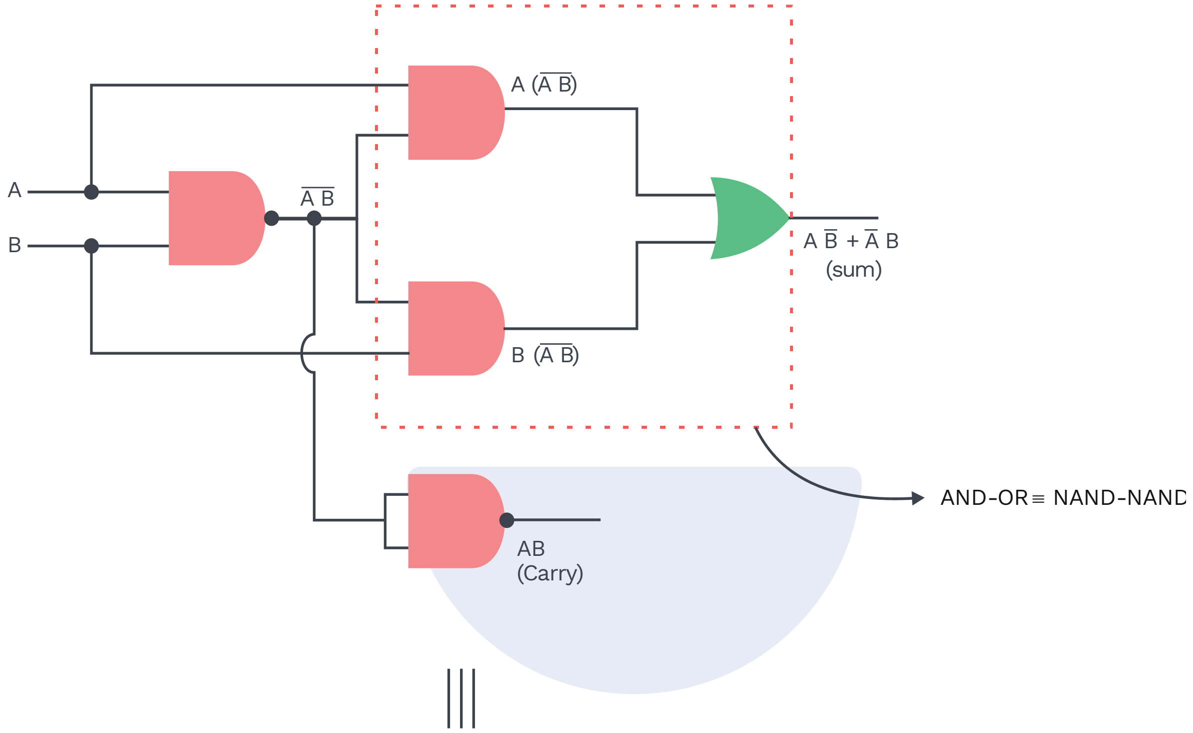


Fig. 2.4 Half Adder using AND-OR Realization

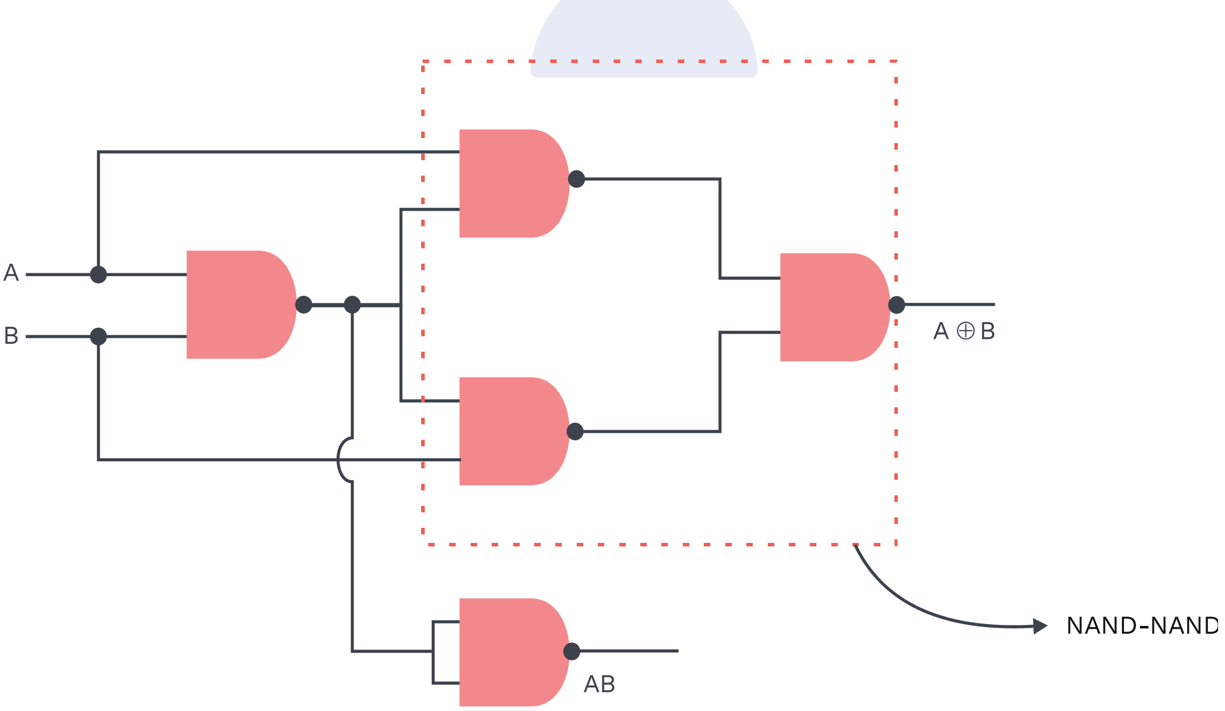
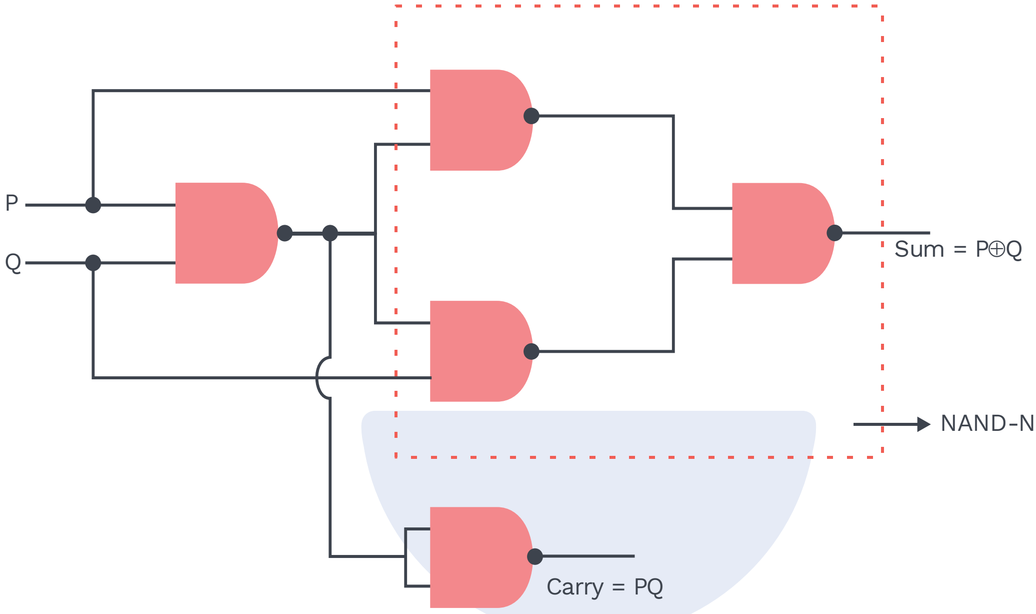


Fig. 2.5 Half Adder using NAND-NAND Realization

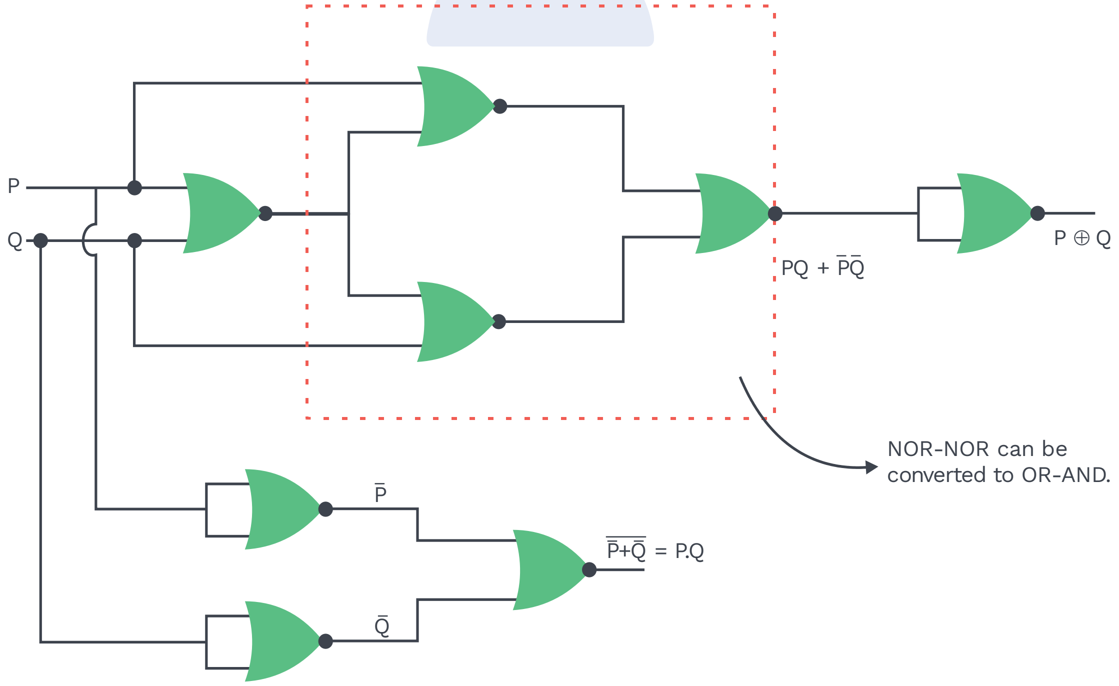
Half adder using NAND gate and NOR gate:



NAND-NAND≡ AND-OR

Fig. 2.6 Half Adder Circuit Implemented using only NAND Gate

Figure 2.6 represents a half adder circuit, implemented using only the NAND gate.



NOR-NOR can be converted to OR-AND.

Fig. 2.7 Half Adder Circuit Implemented using only NOR Gate

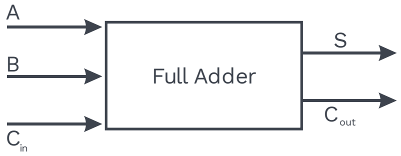
Figure 2.7 represents a half adder circuit, implemented using only NOR gate.

Full adder:

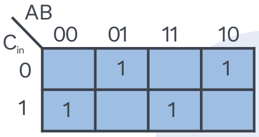
The full adder adds 3 bits and outputs sum bit S and carry bit, C_{out}.

Truth table				
Inputs			Sum	Carry
A	B	C_{in}	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Table 2.2



For sum:



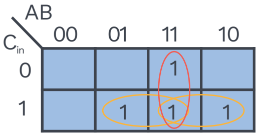
C_{in} \ AB	00	01	11	10
0		1		1
1	1		1	

Fig. 2.8

It is a neutral function since the number of maxterms = the number of minterms.

$$\boxed{S = A \oplus B \oplus C_{in}}$$

For carry:



C_{in} \ AB	00	01	11	10
0			1	
1		1	1	1

Fig. 2.9

$$\boxed{C = AB + BC_{in} + AC_{in}}$$

OR

$$\boxed{C = C_{in}(A \oplus B) + AB}$$

We have got the Boolean expression for sum and carry, now we will implement the Boolean expression using logic gates.

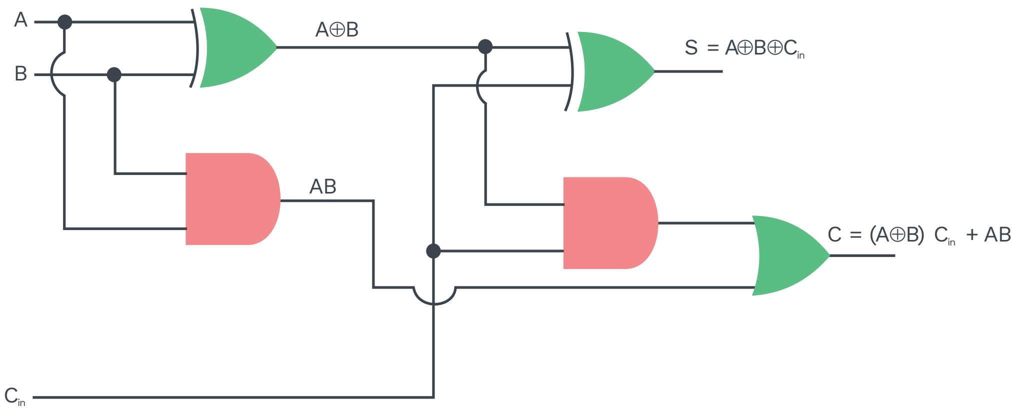


Fig. 2.10 Full Adder using 2 Half Adders and OR Gate

Parallel adder:

Definition

A binary parallel adder is a digital circuit that holds two binary inputs in parallel form and produces 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.

Figure 2.11 shows the connection between four full–adder (FA) circuits to provide a 4 bit parallel adder.

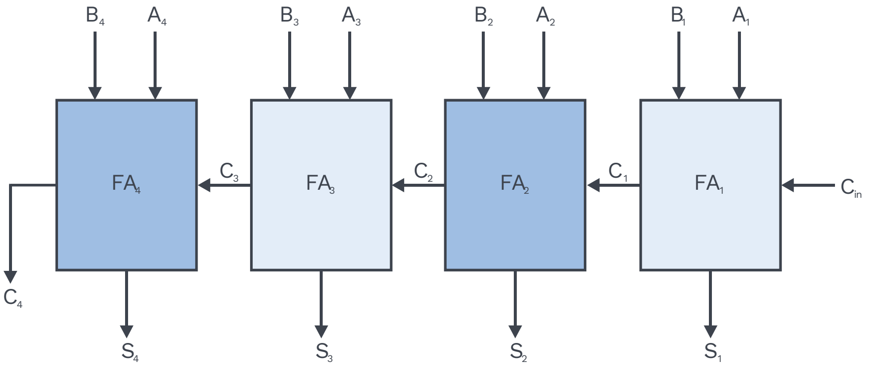
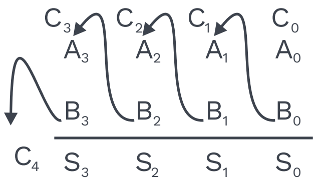


Fig. 2.11 Logic Diagram of a Binary Parallel Adder

The parallel adder in which carry-out of each full adder is the carry-in to the next most significant adder is called ripple carry adder. The greater the number of bits that a ripple carry adder must add, the greater is the time required to perform addition.

Carry look ahead adder:



$C_0 \leftarrow$ initial carry
$C_4 \leftarrow$ final carry

$\boxed{C_1 = C_0(A_0 \oplus B_0) + A_0B_0}$

↳(C_1 is generated if $\boxed{A_0 = 1, B_0 = 1}$

OR

$\boxed{\begin{array}{l} A_0 = 1\ B_0 = 0 \\ \quad \& \\ C_0 = 1 \end{array}}$

OR

$\boxed{\begin{array}{l} A_0 = 0\ B_0 = 1 \\ \quad \& \\ C_0 = 1 \end{array}}$)

$\boxed{C_2 = A_1B_1 + C_1(A_1 \oplus B_1)}$ ...(2)

$\boxed{C_3 = A_2B_2 + C_2(A_2 \oplus B_2)}$ ...(3)

$\boxed{C_4 = A_3B_3 + C_3(A_3 \oplus B_3)}$ ...(4)

In all 4 equations, each carry is dependent on the previous carry. If we perform back-substitution, put the value of C_1 in C_2 then C_2 will be dependent on C_0, which is available at the beginning.
We can continue with this process.
In this way, we can generate carry at each stage independently.

Let

$G_i = A_i\, B_i$ i = stage number

G = generator

$P_i = A_i \oplus B_i$ (it determines whether carry in stage i should be propagated to stage i + 1)

Rewriting the equations of carry in terms of G_i and P_i:

$C_1 = C_0P_0 + G_0$ ...(1)

$C_2 = C_1P_1 + G_1$ ...(2)

$C_3 = C_2P_2 + G_2$ ...(3)

$C_4 = C_3P_3 + G_3$ ...(4)

After substitution:

$C_1 = C_0P_0 + G_0$ ...(1)

$C_2 = C_0P_0P_1 + G_0P_1 + G_1$ ...(2)

$C_3 = (C_0P_0P_1 + G_0P_1 + G_1)\, P_2 + G_2$ ...(3)

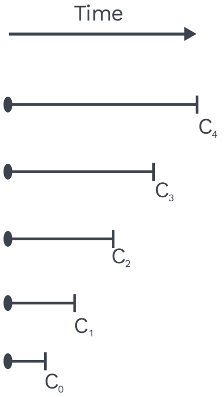
$C_4 = C_0P_0P_1P_2P_3 + G_0P_1P_2P_3 + G_1P_2P_3 + G_2P_3 + G_3$ ...(4)

Note:

Carry is dapendent G_i, P_i, C_0

Time complexity of carry lookahead adder:

(if we do not have gates of required fan-in)



All the carry will be generated parallelly, but there will be a lag due to the presence of more SOP (sum of product) terms as we move from C_1 to C_4

$$C_4 = C_0P_0P_1P_2P_3 + G_0P_1P_2P_3 + G_1P_2P_3 + G_2P_3 + G_3$$

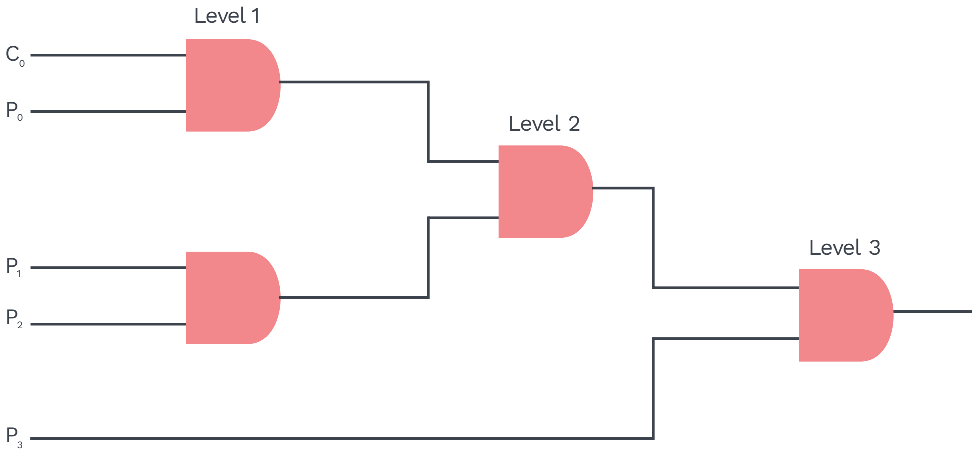


Fig. 2.12 Realization of Longest Term in Final Carry with Basic Gates of FAN-in 2

T_{AND} → Propagation delay of AND gate

$$\boxed{\text{Time complexity} = 3 \times T_{AND}}$$

Generalisation:

If there are "n+1" terms and we have AND gate of fan-in 'k', then $\left(\frac{n+1}{k^i}\right)$ is the number of gates required at i^{th} level.

$$(n+1) \rightarrow \left(\frac{n+1}{k}\right) \rightarrow \left(\frac{n+1}{k^2}\right) \rightarrow \ldots\ldots\ldots\ldots \frac{n+1}{k^i}$$

$\frac{n+1}{k^i} = 1 \quad i \leftarrow$ number of level.

$\Rightarrow n + 1 = k^i$

$$\boxed{\log_k(n+1) = i}$$

Time complexity $- \Theta(\log_k(n+1) \times T_{AND})$

Previous Years' Question

In a look-ahead carry generator, the carry generate function G_i and the carry propagate function P_i for inputs A_i and B_i are given by:

$P_i = A_i \oplus B_i$ and $G_i = A_iB_i$

The expressions for the sum bit C_{i+1} and the carry bit City of the look-ahead carry adder are given by:

$S_i = P_i \oplus C_i$ and $C_{i+1} = G_i + P_i\, C_i$, where C_0 is the input carry.

Consider a two-level logic implementation of the look-ahead carry generator. Assume that all P, and G are available for the carry generator circuit and that the AND and OR gates needed to implement the look-ahead carry generator for a 4-bit adder with S_3, S_2, S_1, S_0 and C_4 as its outputs are respectively:

1) 6, 3
2) 10, 4
3) 6, 4
4) 10, 5

Sol: 2) **(CS-2007)**

BCD adder:

Let's briefly review the process of BCD addition

1) For each digit in a decimal number, add its 4 bits BCD group.

2) If the addition result comes to be a number whose sum is less than 9, then that sum is in proper bcd form. No further updations are required.

3) But if the result of addition comes to be greater than 9, then simply add 0110 to the result to get the proper BCD. This will produce a carry to be added to the next decimal position.

Definition

A BCD adder circuit must be able to operate in accordance with the above steps. In other words, the 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 '9' (1001), if it is then add 0110 (decimal 6) to this sum and generate carry for next decimal position.

SOLVED EXAMPLES

Q1 **Perform the following decimal addition in 8421 code.**
a) 25+13 **b) 679.6+536.8**

Sol: **a)**

$$\begin{array}{rcll}
25 & \Rightarrow & 0010\quad 0101 & (25 \text{ in BCD}) \\
+\,13 & \Rightarrow & 0001\quad 0011 & (13 \text{ in BCD}) \\
\hline
38 & & 0011\quad 1000 & \rightarrow \text{No carry, no illegal code.}
\end{array}$$

b)

$$\begin{array}{rcllll}
679.6 & \Rightarrow & 0110 & 0111 & 1001 & .\ 0110 \\
+\,536.8 & \Rightarrow & 0101 & 0011 & 0110 & .\ 1000 \\
\hline
1216.4 & & 1011 & 1010 & 1111 & .\ 1110 \leftarrow \text{All are illegal codes} \\
 & & 0110 & 0110 & 0110 & .\ 0110 \\
\hline
 & & \textcircled{1}0001 & 0000 & 0101 & .\ 0100 \\
 & & +1 & +1\ + & 1\ + & 1 \\
\hline
 & & \underbrace{0001}_{1}\ \underbrace{0010}_{2} & \underbrace{0001}_{1} & \underbrace{0110}_{6} & .\ \underbrace{0100}_{4}
\end{array}$$

Sol: 1216.4

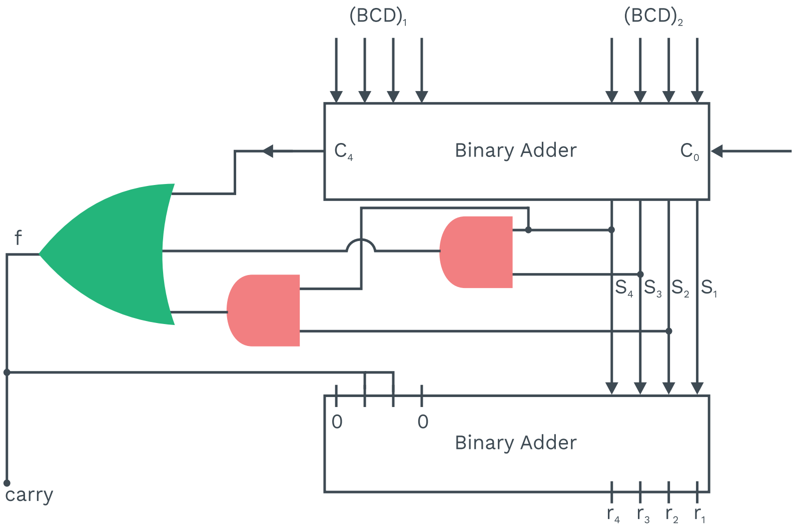


Fig. 2.13 Logic Diagram of BCD Adder

2’s Complement adder:

For representing negative numbers and in the process of subtraction best way is to represent the numbers in the 2’s compliment form. Both the addition and subtraction operation of signed numbers can be done only by using the addition operation if we use 2’s compliment form to represent negative numbers.

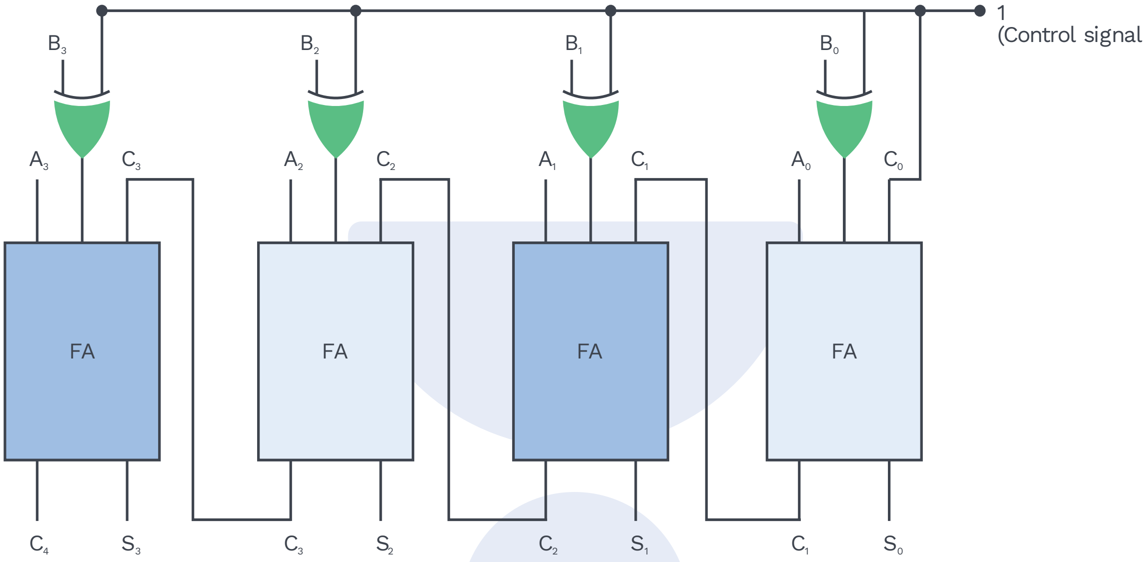


Fig. 2.14 2’s Compliment Adder

Figure 2.14 shows a complete circuit that performs both addition and subtraction in 2’s compliment. When the control signal is ‘1’, the output of all the XOR gates will be $\overline{B}$ (complement of B), adding ‘1’ to which will give 2’s complement of B. So we connect the initial carry (C_0) to the control signal.

$\boxed{\text{The circuit performs} \rightarrow A + 2\text{'s compliment of } B = A - B}$

↘ subtraction

Similarly, when the control signal is 0, the circuit behaves like a parallel adder.

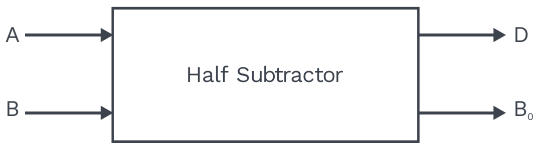
2.3 SUBTRACTOR

Half subtractor:

A half subtractor is a combinational circuit that performs the subtraction of one bit with the other and gives the result as a difference. There is an output to specify if a ‘1’ has been borrowed. It subtracts the LSB of the subtrahend from the LSB of the minuend.

Input		Output	
A	**B**	**D**	**B_0**
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

a) Truth table



b) Block diagram

The output borrow B_0 is ‘0’ as long as A ≥ B. It is 1 for A = 0 and B = 1. The logical expression of difference and Borrow is as follows:

$$\boxed{D = A\bar{B} + \bar{A}B = A \oplus B}$$

$$\boxed{B_0 = \bar{A}B}$$

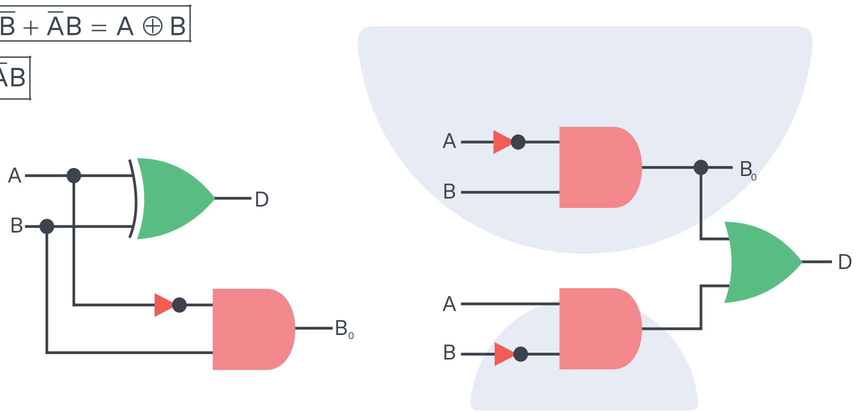


Fig. 2.15 Logic Diagrams of a Half Subtractor

Half subtractor Using NAND gate AND NOR gate:

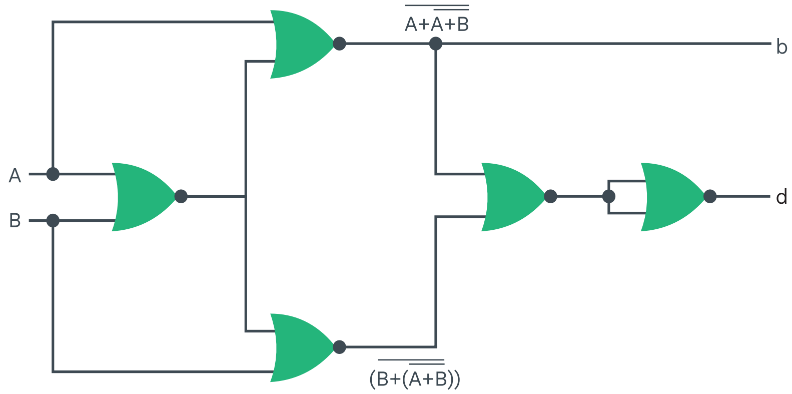

$\overline{A+\overline{A+B}}$
b
A
B
d
$(\overline{B+(\overline{A+B})})$

Fig. 2.16 Half-Subtractor using 2-Input NOR Gates

Logic diagram of half-subtractor using only 2-input NOR gates

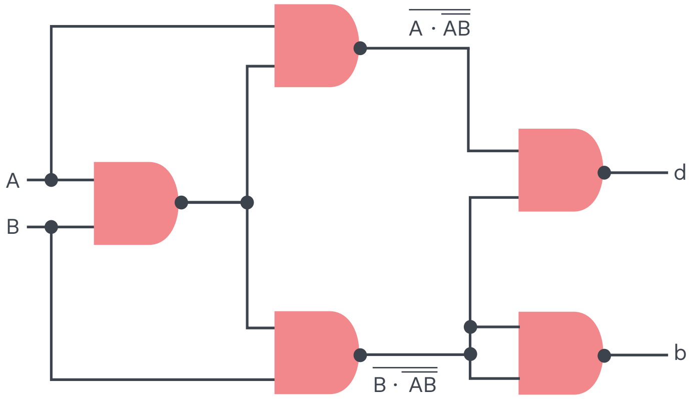

$\overline{A \cdot \overline{AB}}$
A
B
d
b
$\overline{B \cdot \overline{AB}}$

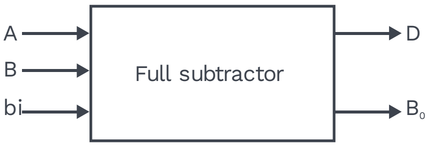
Fig. 2.17 half-Subtractor using 2-Input NAND Gates

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

Full subtractor:

The half subtractor can perform only LSB subtraction. If, in result borrow is present, then this borrow is forwarded to the next higher bits.

Input			Difference	Borrow
A	**B**	**b_i**	**D**	**B_o**
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

From the truth table, a circuit that will give the correct difference and bits in response to every possible combination of A, B and bi is given as:

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

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

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

$$\boxed{D = A \oplus B \oplus b_i}$$

$$B_0 = \bar{A}\bar{B}b_i + \bar{A}B\bar{b}_i + \bar{A}Bb_i + ABb_i = \bar{A}B(b_i + \bar{b}_i) + (AB + \bar{A}\bar{B})b_i$$

$$\boxed{B_0 = \bar{A}B + (\overline{A \oplus B})b_i}$$

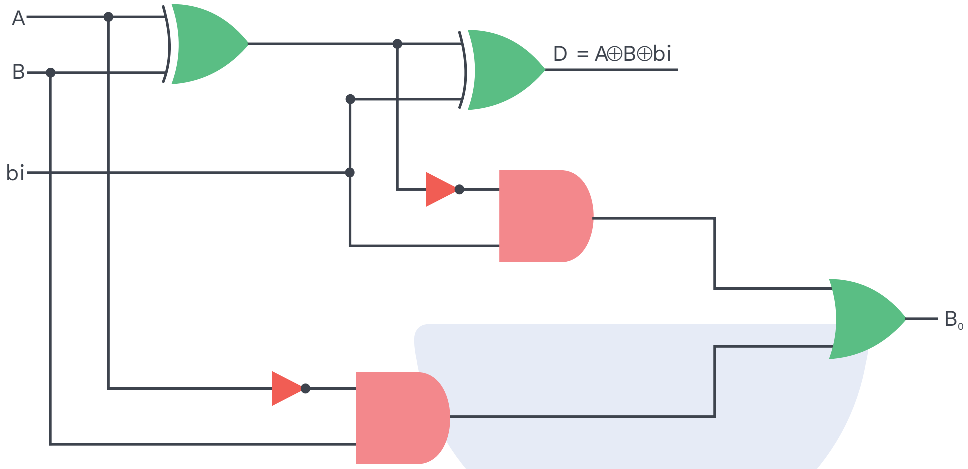


Fig. 2.18 Logic Diagram of a Full-Subtractor

Full subtractor using NAND Gate AND-NOR gate:

$D = A \oplus B \oplus b_i$

$B_0 = \bar{A}B + (\overline{A \oplus B})b_i$

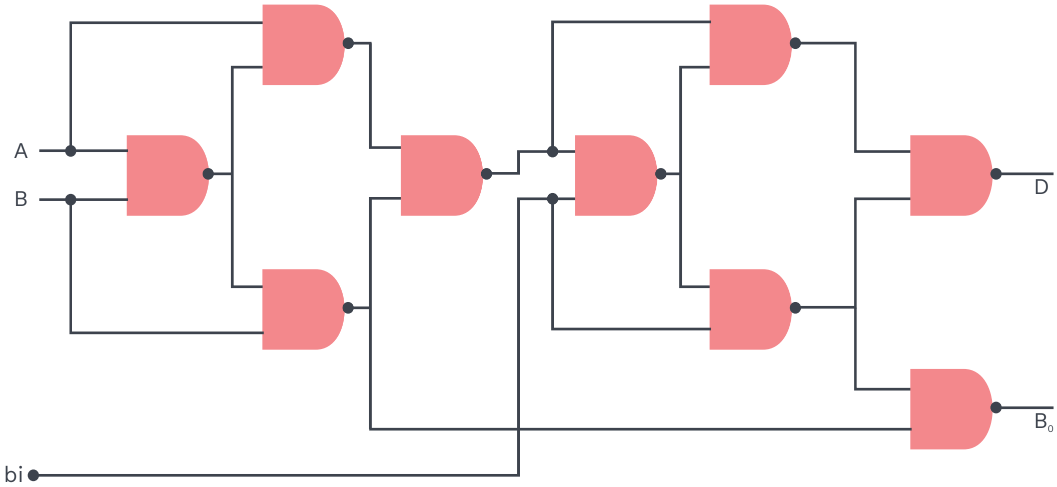


Fig. 2.19 Logic Diagram of a Full Subtractor using 2 Input NAND Gates.

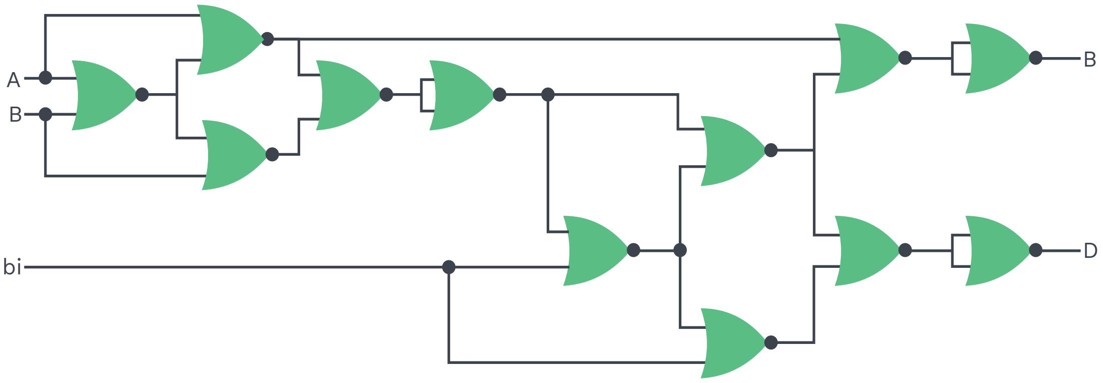


Fig. 2.20 Logic Diagram of a Full Subtractor using only 2-Input NOR Gates.

Parallel subtractor:

Complementation helps in carrying out subtraction in an easy manner. The subtraction A-B can be performed with the help of 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.

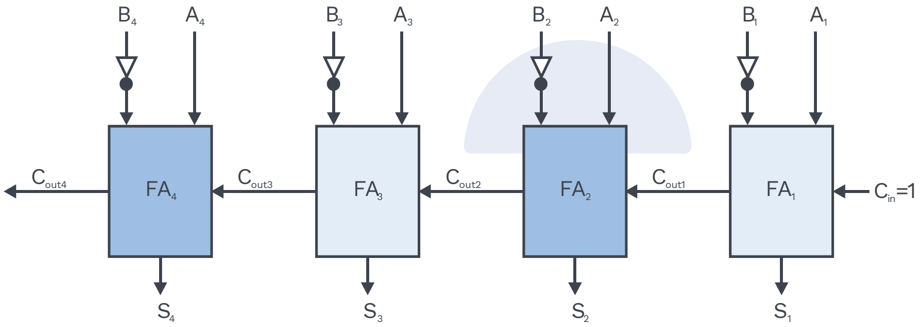


Fig. 2.21 Logic Diagram of a 4 Bit Parallel Subtractor

2.4 ENCODER

Introduction:

A digital encoder is a device whose inputs are decimal digits and whose output is the coded representation of those inputs.

Figure 2.21 shows a block diagram of an encoder with M inputs and N outputs. Here inputs are active high.

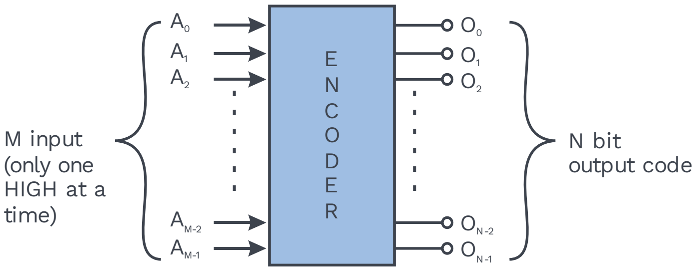


Fig. 2.22 Block Diagram of an Encoder

Octal to binary encoder:

An octal to a binary encoder (8 lines to 3 line encoder) accepts 8 input lines and produces a 3 bit output code corresponding to the particular input. From the truth table, we observe that A_2 is 1 if any of the digits D_4 or D_5 or D_6 or D_7 is 1.

Therefore,

$A_2 = D_4 + D_5 + D_6 + D_7$

Similarly,

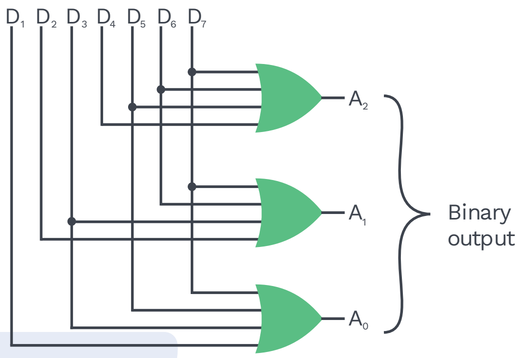
$A_1 = D_2 + D_3 + D_6 + D_7$

$A_0 = D_1 + D_3 + D_5 + D_7$

Note:

D_0 is not present in any of the expression, So D_0 is considered as "don't care".

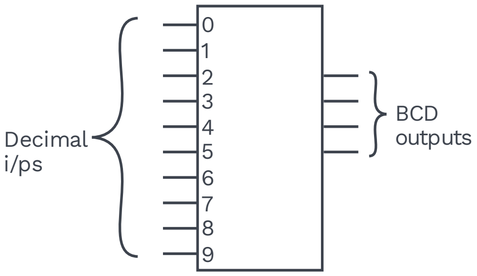
Octal digits		Binary		
		A_2	A_1	A_0
D_0	0	0	0	0
D_1	1	0	0	1
D_2	2	0	1	0
D_3	3	0	1	1
D_4	4	1	0	0
D_5	5	1	0	1
D_6	6	1	1	0
D_7	7	1	1	1



a) Truth table **b) Logic Diagram**

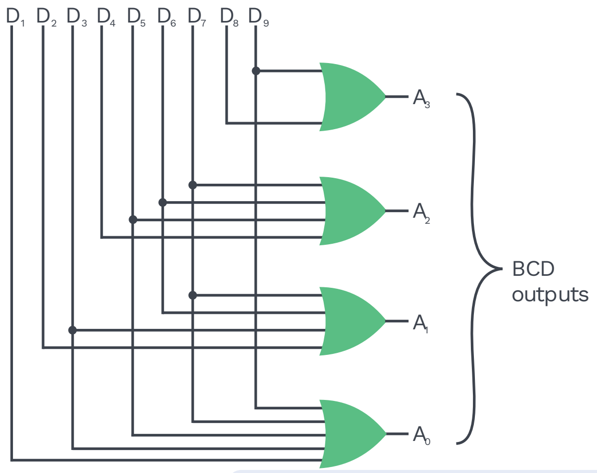
Decimal to BCD encoder:

This type of encoder has 10 inputs – one for each decimal digit and 4 outputs that correspond to the BCD code.



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

a) Logic symbol b) Truth table



c) Logic Design

Hexadecimal to binary encoder:

In hexadecimal number system, radix = 16

Range – 0 to F (15)

Total available numbers in base 16 = 16 = $2^m \Rightarrow m = \log_2 16 = \log_2 2^4 = 4$

$\therefore$ 4 o/p bits are required.

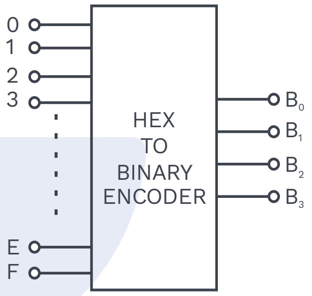
From Fig. 2.17

$B_0 = 1 + 3 + 5 + 7 + 9 + B + D + F$

$B_1 = 2 + 3 + 6 + 7 + A + B + E + F$

Similarly, we can find out B_2 & B_3 from the truth table.

I/p	O/p			
Hex	B_3	B_2	B_1	B_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
A	1	0	1	0
B	1	0	1	1
C	1	1	0	0
D	1	1	0	1
E	1	1	1	0
F	1	1	1	1



Priority encoder:

The encoders we discussed so far will operate correctly, provided that only one decimal input is high at any given time. In some practical systems, two or more decimal inputs may be HIGH at the same time.

A priority encoder is a logic circuit that responds to just one input in accordance with some priority system, among all those that may be simultaneously high.

The most common priority system is based on the relative magnitude of the inputs; whichever decimal input is largest is the one that is encoded.

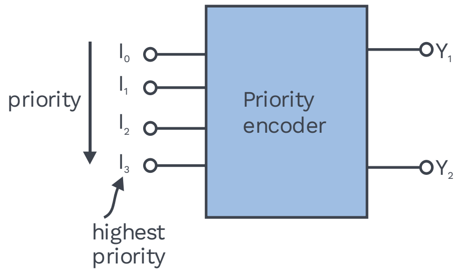


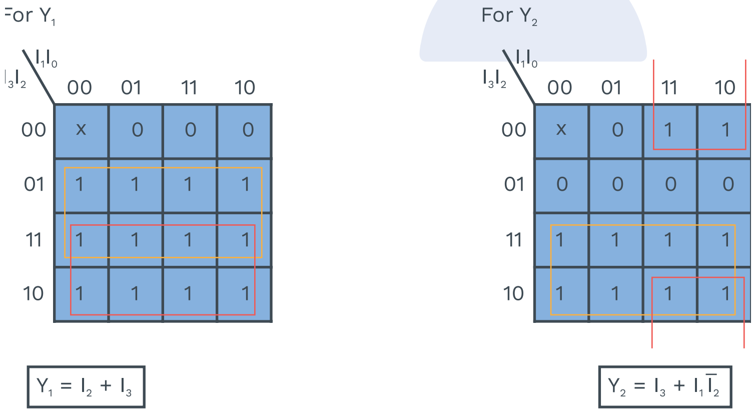
Fig. 2.23 Priority Encoder

Let's look at the truth table:

I_3	I_2	I_1	I_0	Y_1	Y_2
0	0	0	0	X	X
0	0	0	1	0	0
0	0	1	X	0	1
0	1	X	X	1	0
1	X	X	X	1	1

Table 2.3

X → don't care



For Y_1

I_3I_2 \ I_1I_0	00	01	11	10
00	x	0	0	0
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

$Y_1 = I_2 + I_3$

For Y_2

I_3I_2 \ I_1I_0	00	01	11	10
00	x	0	1	1
01	0	0	0	0
11	1	1	1	1
10	1	1	1	1

$Y_2 = I_3 + I_1\overline{I_2}$

Fig. 2.24

2.5 DECODER

Introduction:

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

Figure 2.24 shows the general decoder diagram. Since each of the N inputs can be a 0 or 1, there are 2^N possible input combinations. For each of these input combinations, there are only one of the 'M' outputs that will be active (HIGH), all other outputs will remain inactive (LOW).

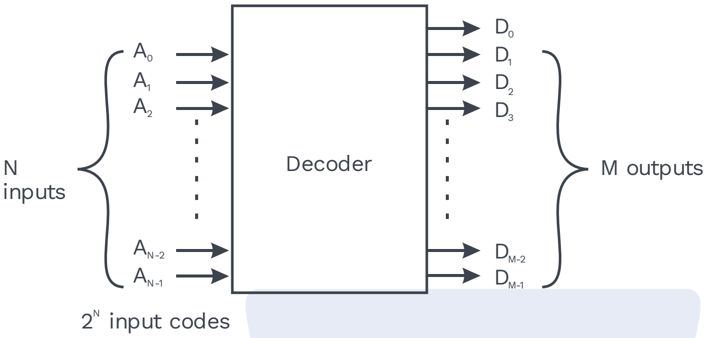


Fig. 2.25 General Block Diagram of Decoder

BCD to 7 segment decoder:

This type of decoder accepts BCD code and provides output to lighten seven segment display devices to produce a decimal read out.

Each segment is made up of a material that emits eight when the current is passed through it.

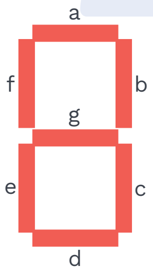


Fig. 2.26 Letters used to Designate the Segment

Decimal digit	8-4-2-1 BCD				Seven Segment Code						
	A3	A2	A1	A0	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1

Table 2.4 Functional Table

We can find a simplified Boolean expression for each of the segments from the functional table.

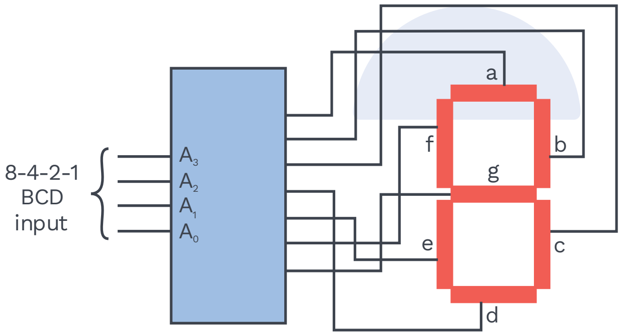
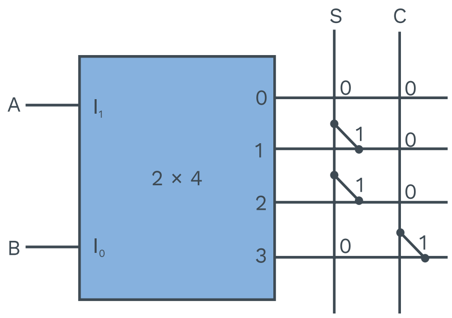


Fig. 2.26 Logic Circuit

Logic construction of ROM:

- ROM can be used to realize combinational functions by storing appropriate values at locations.
- Every ROM is expressed in terms of the ROM matrix and decoder.
- ROM matrix contains a set of links and connections. The lines entering the matrix and leaving are called connections and intersections between rows and columns are called links.



a)

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

b)

Fig. 2.27 Logic Diagram and Truth Table of ROM

Implementing function using only decoder:

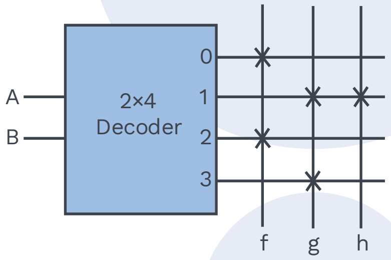


Fig. 2.28

Note:

The number of vertical lines gives the number of functions.

f = $\sum$ (0, 2)
g = $\sum$ (1, 3)
h = $\sum$ (1)

Implementing function using decoder (ROM) and multiplexer:

Consider the function, F = $\sum$ (0, 1, 5, 7)

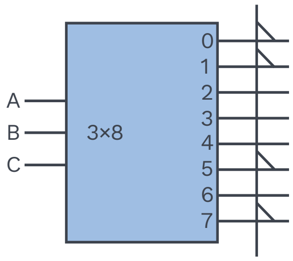


Fig. 2.29 Implementation of 'F' Using ROM

The above decoder (ROM) has → 8 + 1 connections

→ 1 Function

→ 8 × 1 = 8 Links

We can represent the same function using (2 × 4) decode & 2 × 1 MUX.

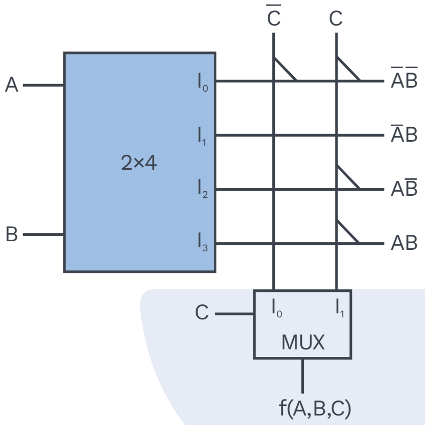


Fig. 2.30 Implementing Function Using ROM and Multiplexer

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

$\Rightarrow$ A = 0 B = 0 C = 0

$F = \overline{A}\overline{B}\overline{C}$

$\Rightarrow$ A = 0 B = 0 C = 1

$F = \overline{A}\overline{B}C$

The above combinational circuit has → (4+2) connection

→ 4 × 2 = 8 links

For a 10 variables function and for a given set of decoder and MUX, let's find how many 'connections' and 'Links' exist.

Decoder	MUX	Connections	Links
10	0	1024+1	1024
5	5	32+32=64	$2^5 \times 2^5$ = 1024
4	6	16+64=80	$2^4 \times 2^6$=1024
3	7	8+128=136	1024

Table 2.5

Note:

Beyond a certain limit number of connections starts increasing.

Rack Your Brain

Will always adding decoder and mux decrease number of connections?

Decoder with enable:

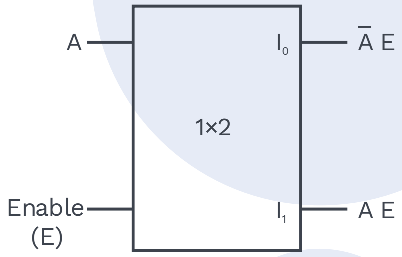
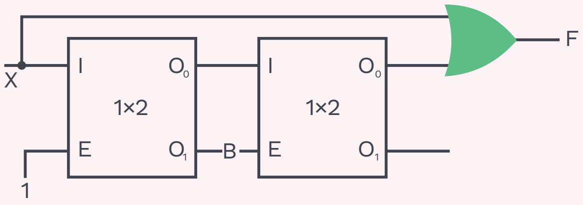


Fig. 2.31 1 X 2 decoder

E = 1 and A = 0 then $I_0 = 1 = \bar{A}E$

E = 1 and A = 1 then $I_1 = 1 = AE$

Rack Your Brain

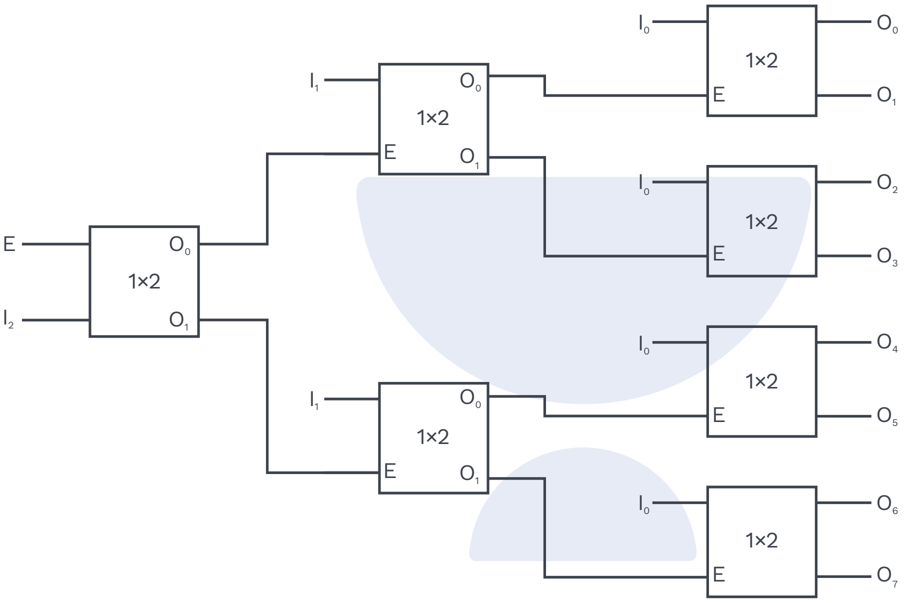


Find the Boolean expression for F.

SOLVED EXAMPLES

Q1 **Construct 3×8 decoder using 1×2 decoder.**

Sol:



Expansion of decoder in general

Q2 **Construct m×2^m decoder using n×2^n**

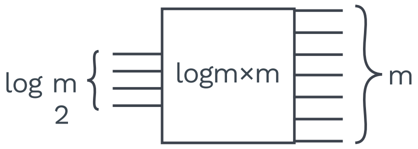
Sol: **n×2^n decoder**

1 device → “n” lines

1 line → $\frac{1}{n}$ device (not feasible, just for the sake of calculation)

Level 1

Let’s say,



$$\text{m lines} \rightarrow \frac{m}{n} \text{ devices}$$

Level 2

$$\left(\frac{m}{n}\right)\text{lines} \rightarrow \left(\frac{1}{n} \times \frac{m}{n}\right)\text{devices}$$

Level 3

$$\frac{m}{n^2}\text{lines} \rightarrow \left(\frac{m}{n^3}\right)\text{devices}$$

$$\begin{array}{ll} \vdots & \vdots \\ \vdots & \vdots \\ \vdots & \vdots \\ \vdots & \vdots \\ \vdots & \vdots \\ \vdots & \vdots \\ \vdots & \end{array}$$

Level K $\vdots$

$$\frac{m}{n^k} \approx 1$$

$$\frac{m}{n^k} \leq 1$$

$$\boxed{k \geq \left\lceil \left(\frac{\log_2 m}{\log_2 n}\right) \right\rceil}$$

Note:

The total number of decoders required for ‘k’ levels:

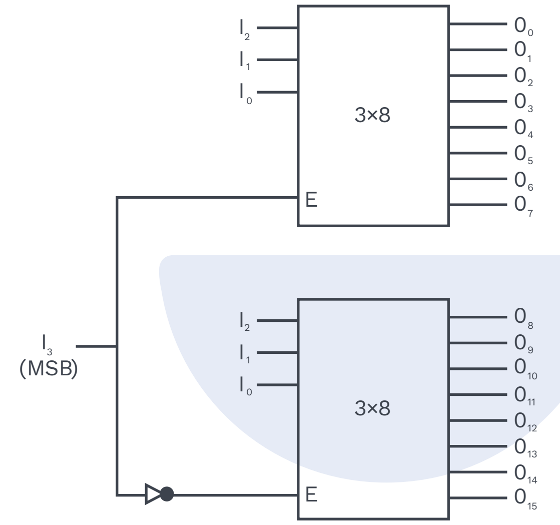
$$\left(\sum_{k=1}^{\frac{\log m}{\log n}} \frac{m}{n^k}\right)$$

Rack Your Brain

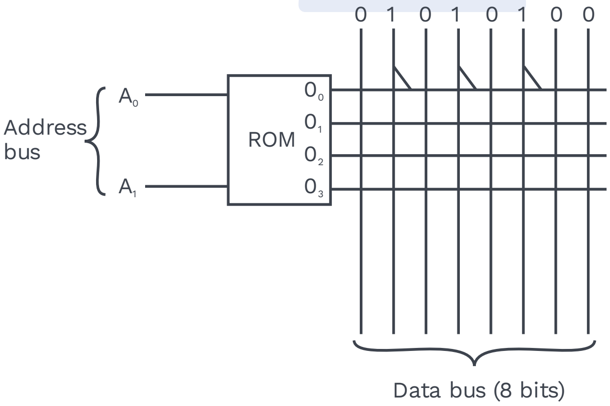
How many levels are required to construct a 7×128 decoder using a 1×2 decoder? Also, find the total number of decoders required.

Q3 **Using 2 (3×8) decoder construct 4×16 decoder.**

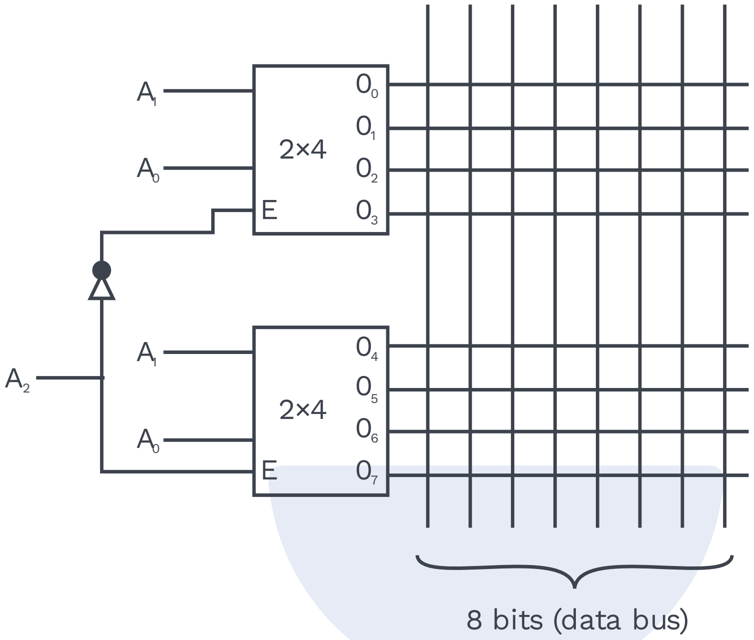
Sol:



Address expansion of ROM:

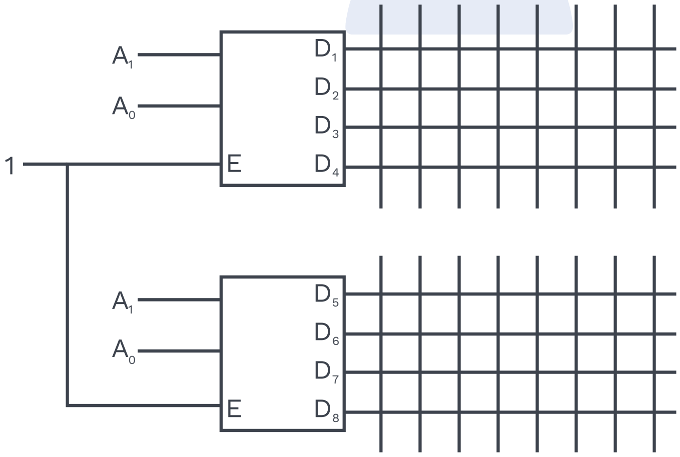


If we give A_0 =0, A_1 = 0 in the address, then we will get the data stored in 'O_0'
O_0 will 0 1 0 1 0 1 0 0
Total size = 4 words × 8 bits.
After expanding the address, i.e. increasing the number of words that can be stored (the size of each word is the same)



Total size = 8 word × 8 bit

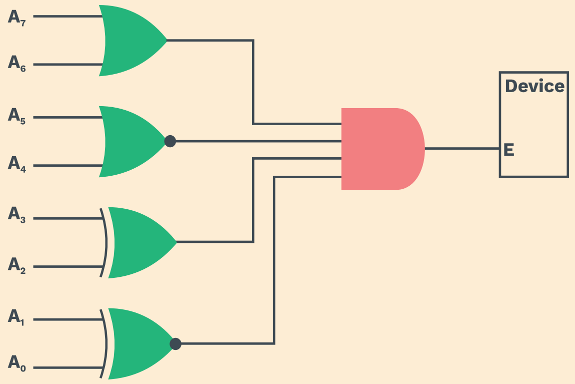
Word expansion of ROM:



When $A_1 = 0$, $A_0 = 0$, D_1 and D_5 are going to get selected.
(overall, we'll have 8+8 = 16 bits)
Size of word = 16 bits
Number of words = 4

SOLVED EXAMPLES

Q1 **Consider a device that is enabled using 8 address bits as shown below. For how many address the device is enabled:**



Sol:

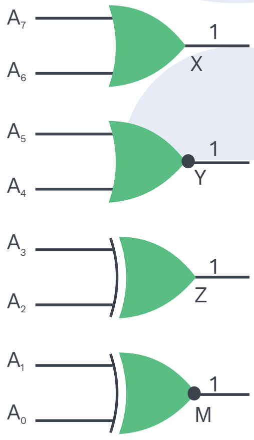


Fig. 2.32

'X' can be made '1' in 3 ways:

$A_7 = 1 \quad A_6 = 0$

$A_7 = 0 \quad A_6 = 1$

$A_7 = 1 \quad A_6 = 1$

Similarly,

Y → 1 way

Z → 2 ways

M → 2 ways

Total ways = 3 × 2 × 2 = 12 ways.

2.6 DEMULTIPLEXER

Definition

A multiplexer takes several inputs and transmits one of them to the output. A demultiplexer performs the reverse operations of multiplexer. It takes a single input and distributes it over many outputs. Basically Demultiplexer can be thought of a distributor, since it transmits same data to different destinations.

Note:

Demultiplexer takes one input and selectively distributes it to 1-of-N output channel like a multi position switch.

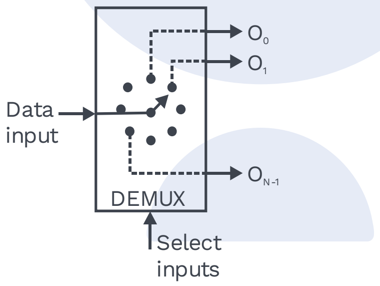
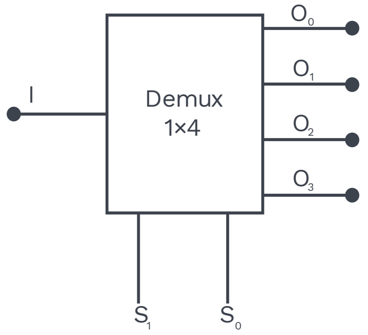


Fig. 2.33 Functional Diagram of a General Demultiplexer.

Let's look at (1×2^2) demultiplexer,

$$\underbrace{1}_{\text{1 i/p line}} \times \underbrace{2}_{\text{4 o/p line}}^{\boxed{2}\rightarrow \text{2 select lines}}$$



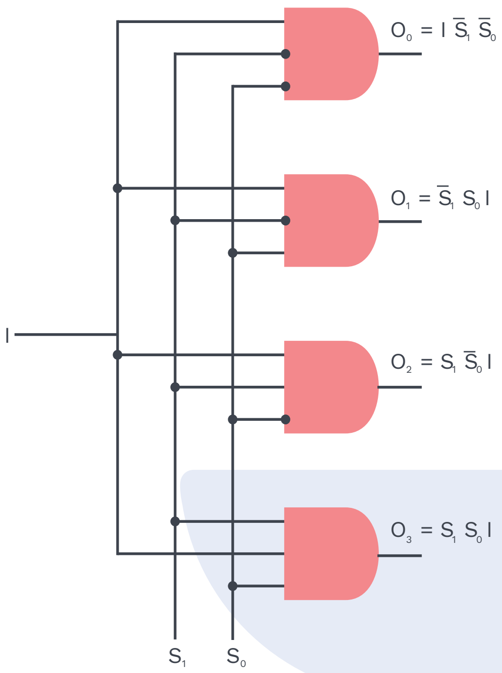

$O_0 = I\,\overline{S}_1\,\overline{S}_0$
$O_1 = \overline{S}_1\,S_0\,I$
$O_2 = S_1\,\overline{S}_0\,I$
$O_3 = S_1\,S_0\,I$
I
S_1
S_0

Fig. 2.34 Logic Circuit Diagram of 1×4 DeMUX

S_1	S_0	O_0	O_1	O_2	O_3
0	0	I	0	0	0
0	1	0	I	0	0
1	0	0	0	I	0
1	1	0	0	0	I

Table 2.6 Truth Table for 1×4 Demux

2.7 MULTIPLEXERS

Definition:

Multiplexing means sharing. A common example of multiplexing or sharing occurs when several hardware devices share a single transmission bus to communicate with a computer.

A multiplexer (MUX) is a logic circuit that accepts several data inputs and allows only one of them at a time to get through the output. The routing of the data input to the output is controlled by select lines.

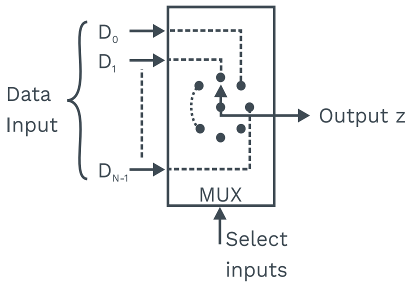
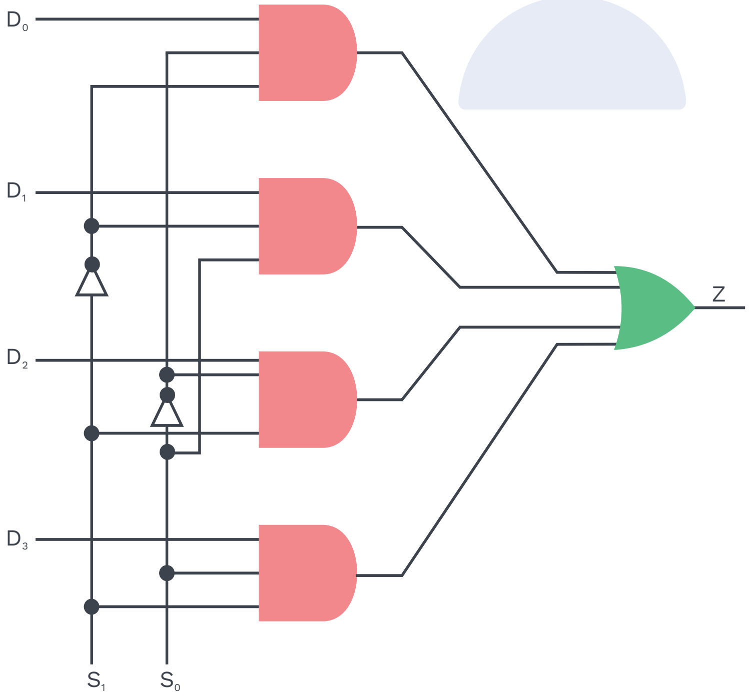


Fig. 2.35 Functional Diagram of Digital Multiplexer

The 4-input multiplexer:

Fig. 2.27 a) shows the logic circularly for a 4 input multiplexer with data inputs D0, D1, D2, and D3 and data select inputs S0 and S1. The logic levels applied to the S0 and S1 inputs determine which AND gate is enabled so that its data input passes through the OR gate to the output. The function table at Fig. 2.27 b) gives the output for the input select codes as

$$\boxed{Z = \overline{S_1}\,\overline{S_0}D_0 + \overline{S_1}S_0D_1 + S_1\overline{S_0}D_2 + S_1S_0D_3}$$



Select inputs		o/p
S_1	S_0	Z
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

a) Logic Diagram **b) Function Table**

MUX as universal logic gates:

Implementation of NAND using 2:1 MUX

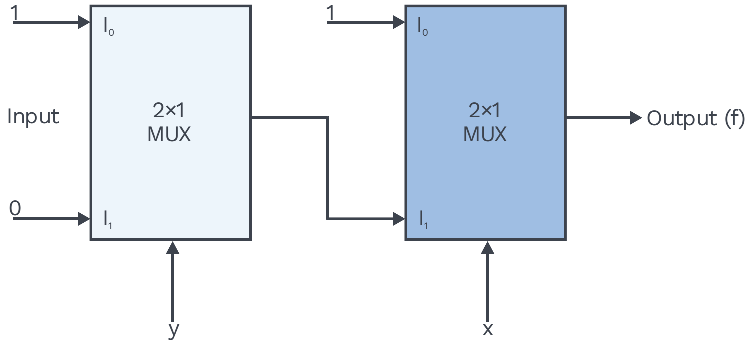


Fig. 2.36 Implementation of NAND Using MUX

The First multiplexer will act as NOT gate, which will produce complemented input to the second multiplexer.

Implementation of NOR gate using 2:1 MUX

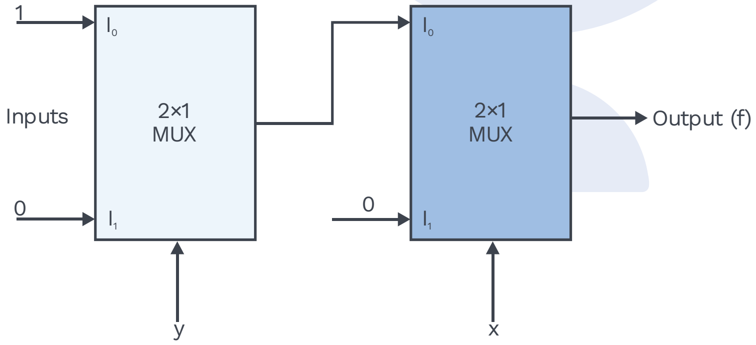
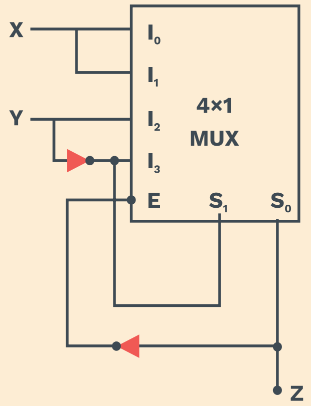


Fig. 2.37 Implementation of NOR Gate Using MUX

SOLVED EXAMPLES

Q1



Sol:

$$f = E(I_0\overline{S}_1\overline{S}_0 + I_1\overline{S}_1S_0 + I_2S_1\overline{S}_0 + I_3S_1S_0)$$

$$f = \overline{\overline{Z}}(XY\overline{\overline{Z}} + XYZ + Y\overline{Y}\overline{Z} + \overline{Y}\overline{Y}Z)$$

$$f = \overline{\overline{Z}}(XY + \overline{Y}Z)$$

$f = XY\overline{\overline{Z}}$ **Sol:**

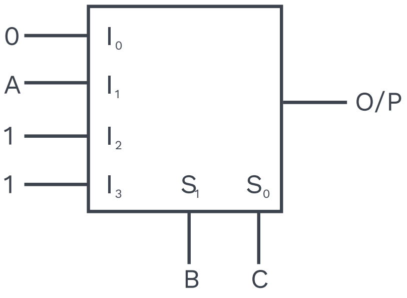
Q2

$F(A, B, C) = \Sigma(2, 3, 5, 6, 7)$
$S_1 = B$, $S_0 = C$
Implement using 4x1 MUX.

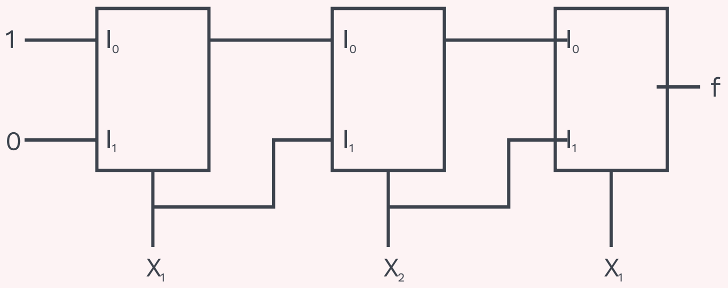
Sol:

$$F = \overline{A}B\overline{C} + \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$$

$$F = 0(\overline{B}\overline{C}) + A(\overline{B}C) + 1(B\overline{C}) + 1(BC)$$



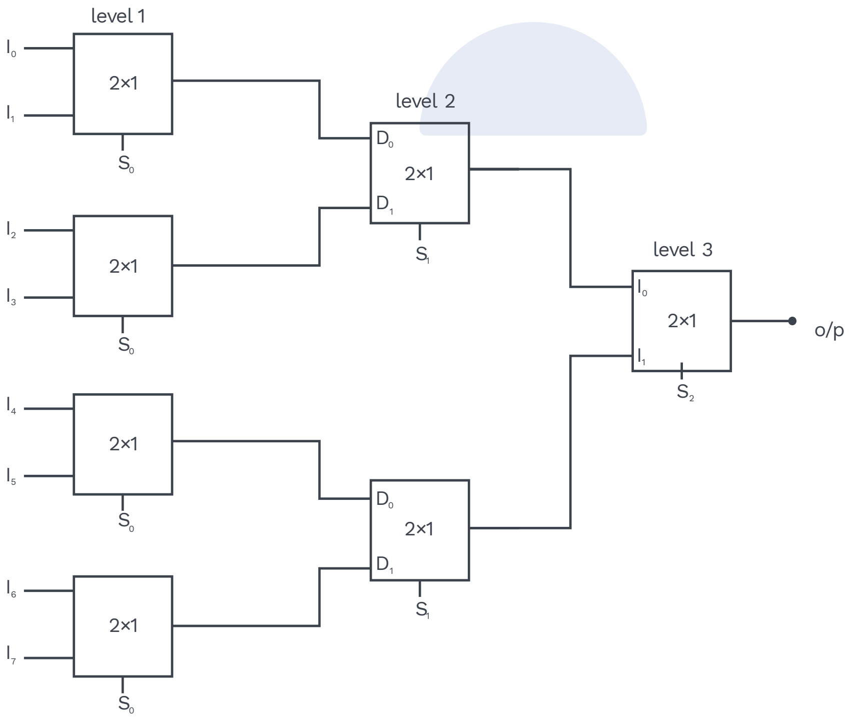
Rack Your Brain



Find ‘f’.

Q3 Construct t×1 MUX using 2×1

Sol:



When $S_2S_1S_0$ = 011, I_1, I_3, I_5, and I_7 are selected at level 1, D_1, D_1 will be selected at level 2, I_0 will be selected at level 3.

∴ I_3 will appear at the output.

M×1 MUX using N×1 MUX:

In N×1 MUX: 'N' lines → 1 device

$1 \text{ line} \rightarrow \frac{1}{N} \text{ device}$

For M lines, $\frac{M}{N}$ devices in 1st level

For $\frac{M}{N}$ lines, $\frac{M}{N} \times \frac{1}{N} = \frac{M}{N^2}$ devices in 2nd level

$\vdots$

$\vdots$

$\vdots$

$\frac{M}{N^K} \le 1$ at Kth level.

$N^K \ge M$

$\boxed{K \ge \log_N M}$ K → No. of level

Total no. of devices - $\boxed{D = \sum_{K=1}^{\lceil \log_N M \rceil} \left(\frac{M}{N^K}\right)}$

Applications:

Logic function generator:

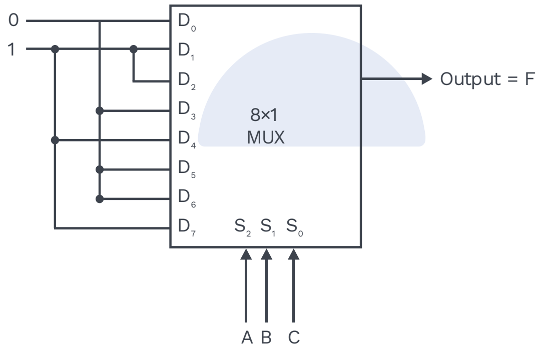
SOLVED EXAMPLES

Q1 **Use a multiplexer to implement the logic function F = A ⊕ B ⊕ C**

Sol: Since F = 1 when ABC = 001, 010, 100 and 111, we connect logic 1 to data inputs D_1, D_2, D_4 and D_7. Logic 0 is connected to other data inputs D_0, D_3, D_5 and D_6. When the data select inputs are any of the combinations for which F = 1, the output will be 1.

S_2	S_1	S_0	F = A⊕B⊕C
A	B	C	
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Table 2.7



Previous Years' Question

Suppose only one multiplexer and one inverter are allowed to be used to implement any Boolean function of n variables. What is the minimum size of the multiplexer needed?

1) 2^n line to 1 line

2) 2^{1+1} line to 1 line

3) 2^{n-1} line to 1 line

4) 2^{n-1} line to 1 line

Sol: 3) **(CS-2007)**

2.8 COMPARATORS

A comparator is a logic circuit used to compare the magnitudes of two binary numbers. Depending on the design, it may either simply provide an output that is active when the two numbers are equal or provide outputs that signify which of the numbers is greater when equality does not hold.

Note:

The X-NOR gate is a basic comparator, because it outputs a 1 only if it's two input bits are equal.

2-bit comparator:

Let two numbers be $\overbrace{(A_2A_1)}^{A}$ and $\overbrace{(B_2B_1)}^{B}$ where A_1 and B_1 are the least significant bits of two numbers.

Case (i) A = B

∵ X NOR gate is called equality detector.

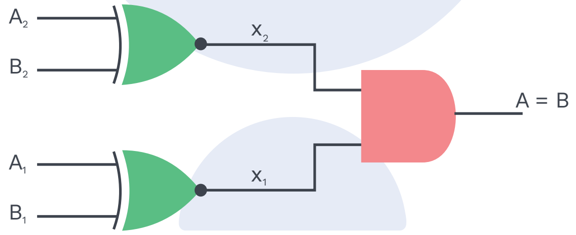


Fig. 2.38 "A = B" Comparator

If $x_1 = 1$ and $x_2 = 1$, o/p is 1

$\boxed{x_1.x_2 = 1}$

Case (ii) A > B

If $(A_2 > B_2)$ OR $(A_2 = B_2$ & $A_1 > B_1)$, o/p is 1

$A_2 = 1$ $\quad A_1 = 1$

$B_2 = 0$ $\quad B_1 = 0$

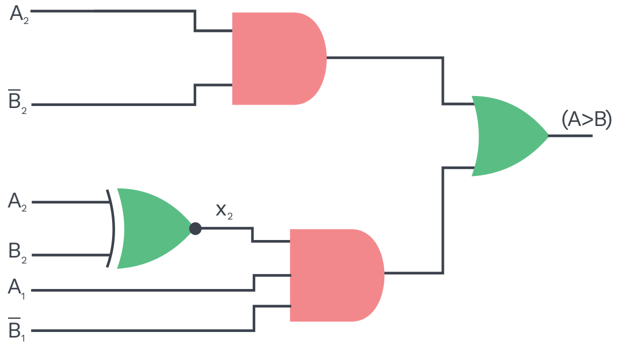


Fig. 2.39 "A>B" Comparator

For A > B, $\boxed{A_2\overline{B}_2 + x_2.(A_1\overline{B}_1)} = 1$

Case (iii) A < B

If (A2 < B2) OR (A_2 = B_2 & A_1 < B_1)

A_2 = 0 A_1 = 0

B_2 = 1 B_1 = 1

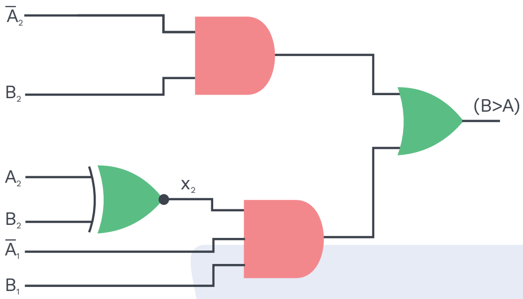
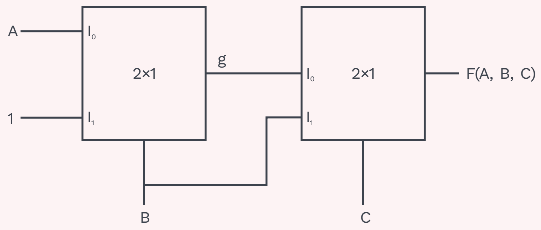


Fig. 2.40 "A<B" Comparator

For B > A, $\boxed{\overline{A}_2B_2 + x_2.(\overline{A}_1B_1)} = 1$

Rack Your Brain



What is the function 'F' in Canonical SOP?

2.9 HAZARDS

Hazards are unwanted switching transients that may appear at the output of a circuit because different paths have different propagation delays. Such a transient is also called a glitch or a spurious spike, which is caused by the hazardous behaviour of a logic circuit. Hazards occur in combinational circuits as well as in sequential circuits.

The permanent hazards are resulted due to open circuits or short circuits of the connecting lead. (terminals)

Note:

The hazards can be "stuck at 0" (s-a-0) or "stuck at 1" (s-a-1)

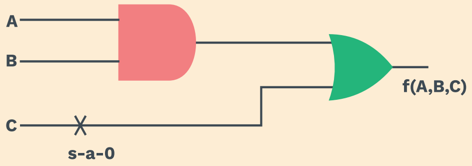
The single fault analysis is made using the "path sensitization" technique. This technique provides a test vector.

Procedure for finding the test vectors:

- Inactive all the other paths except the tested path.
  a) For "AND" or "NAND", apply logic '1' for inactivation.
  b) For "OR" or "NOR", apply logic '0' for inactivation.
- Apply the opposite logic value at the tested place.
- The i/p combinations satisfying the above requirements form the test vector.

SOLVED EXAMPLES

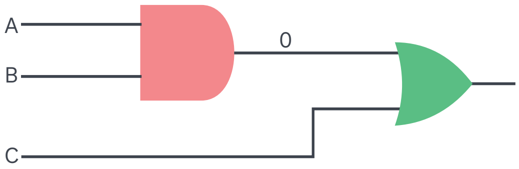
Q1



Is 'c' at s-a-0?

Sol:

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



If for all 3 i/ps, o/p is 1, then c is not s-a-0

Chapter Summary

- Half adder: 2 binary inputs and 2 binary outputs.
  $S = P \oplus Q$
  $C = PQ$
- Full adder: 3 binary inputs and 2 binary outputs.
  $S = P \oplus Q \oplus R$
  $C = R(A \oplus B) + AB$
- Carry look ahead adder:
  Carry is generated at each stage independently.
  Carry is dependent on generator (G_i), propagation (P_i)
  and initial carry (C_0)
- Parallel adder: Output carry from each full-adder is connected to the input carry of the next full adder.
- BCD adder: Adds two 4 bits BCD code groups using straight binary addition.
- Half subtractor: Difference = $A \oplus B$
  Borrow = AB
- Full subtractor: Difference = $A \oplus B \oplus C$
- Borrow = $AB + (A \oplus B)C$
- Encoder: Outputs are coded representations of the inputs.
- Priority encoder: It responds to just one input in accordance with some priority system.
- Decoder: Only one output line is activated for each possible combination of the input
- Demultiplexer: It takes one input data source and selectively distributes it to 1-of-N output channels
- Multiplexer: It accepts several data inputs and allows one of them at a time to get through the output.
- Hazards: Stuck-at-0
  Stuck-at-1