Simplification of Boolean functions:

The Map Method: This method is known as k-map method. It is graphical method, which consists of 2^n cells for n variables. The adjacent cells are differed only in single bit position.

Two-AND Three-variable maps:

	yz: 00	01	11	10
x: 0	0000 m_0	0001 m_1	0011 m_3	0010 m_2
1	0100 m_4	0101 m_5	0111 m_7	0110 m_6
1	1100 m_{12}	1101 m_{13}	1111 m_{15}	1110 m_{14}
0	1000 m_8	1001 m_9	1011 m_{11}	1010 m_{10}

← Adjacency maintain করার জন্য 10, এভাবে লিখা

Using sum of product:

Given, $f(x,y,z) = x'y'z' + x'y'z + xy'z' + xyz$

x \ yz	00	01	11	10
0	1	1	0	0
1	1	0	0	1

$F = x'y' + xz'$

Given, $F = A'C + A'B + AB'C + BC$

A \ BC	00	01	11	10
0		1	1	1
1		1	1	

$F = C + A'B$

n 23

Given, F — Inspection

Process of creating four values map simplification.

→ One square represents one minterm giving a term of four literals. (n)

→ Two adjacent squares represent a term of three literals. (n-1)

→ Four adjacent squares represent a term of two literals. (n-2)

→ Eight adjacent squares represent a term of one literal.

→ Sixteen adjacent squares represent the function equal to 1. (2^n)

Given, $F(w,x,y,z) = \Sigma(0,1,2,4,5,6,8,9,12,13,14)$

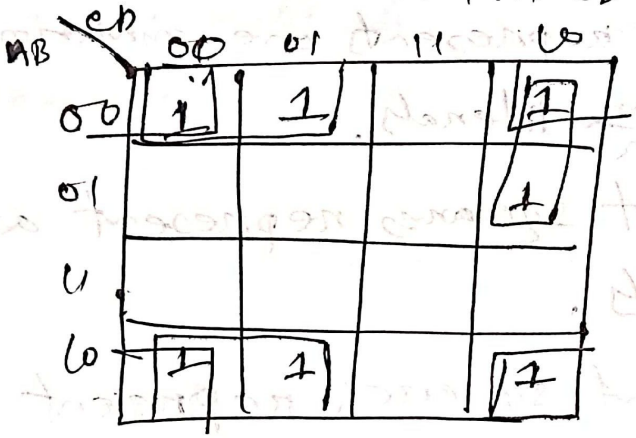
wx \ yz	00	01	11	10
00	1	1		1
01	1	1		1
11	1	1		1
10	1	1		

$F = y' + w'z' + xz'$

Ex-3.6 $F = A'B'C'D' + B'CD' + A'BCD' + AB'C'$

$A'B'C'D$

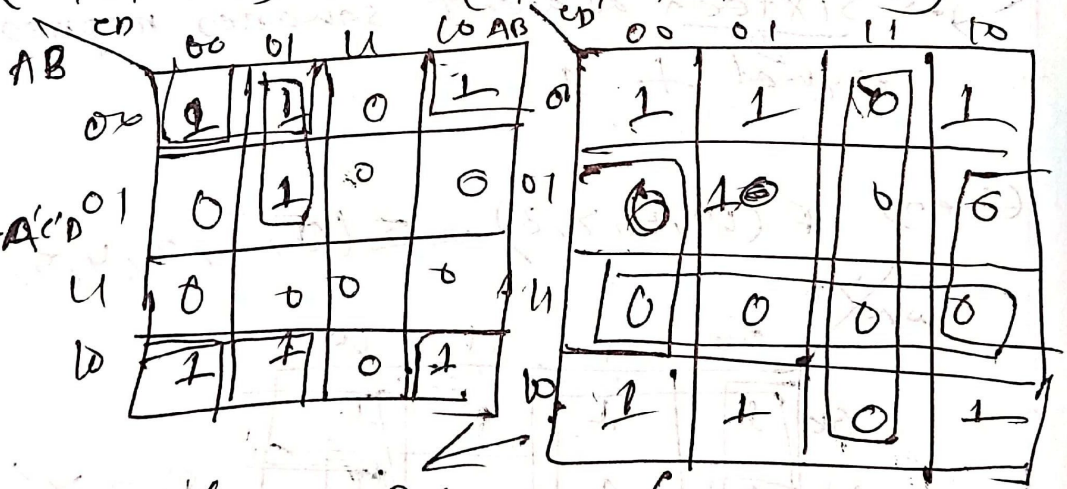
$= A'B'C'D' + A'B'CD' + AB'CD' + A'BCD' + AB'C'D + AB'C$



$F = B'C' + B'D' + A'CD'$

Product of sum:

$F(A, B, C, D) = \Sigma(0, 1, 2, 5, 8, 9, 10)$



Pentium:

$B'D' + B'C' + A'C'D$

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

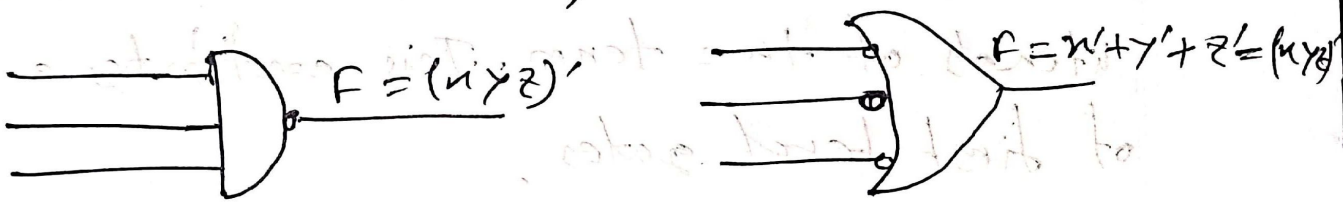
$= A$

$F_p = B'D' + B'C' + \cancel{A}C'D$

$=$

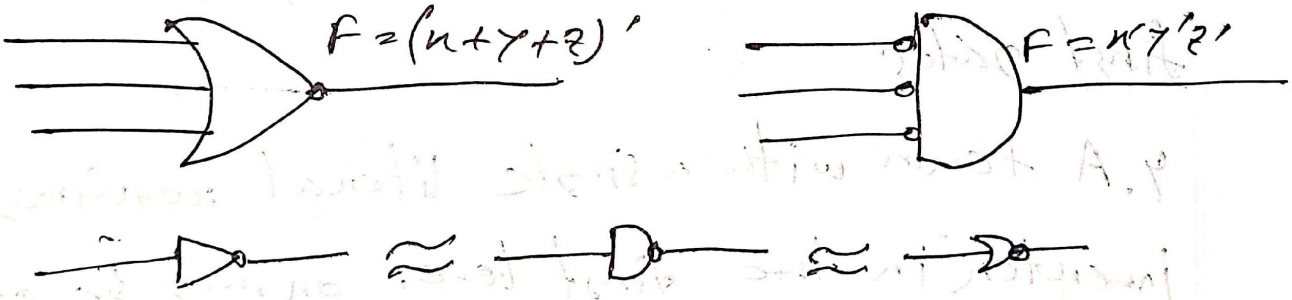
NAND AND NOR Implementation: Digital circuits are more frequently constructed with NAND or NOR gates than with AND and OR gates. NAND and NOR gates are easier to fabricate with electronic components and are the basic gates used in all IC digital logic families.

symbols of NAND gates:

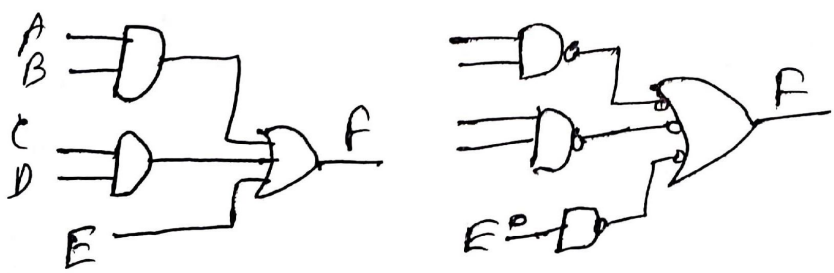


Two graphic symbols for NAND gate.

symbols of NOR gates:



$F = AB + CD + E$



The rules for obtaining the NAND logic diagram from a Boolean function is as follow:

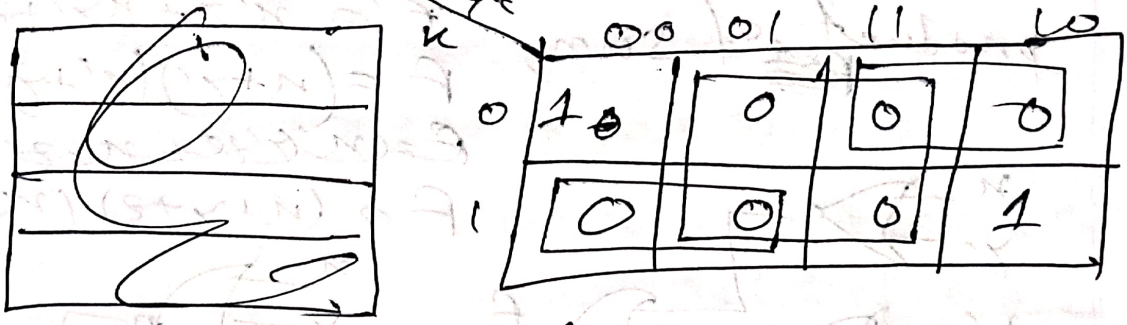
1. Simplify the function and express it in sum of products.

2. Draw a NAND gates for each product term of the function that has at least two literals. The inputs to each NAND gate are the literals of the term. This constitutes a group of first level gates.

3. Draw a single NAND gate (using AND-invert or invert-OR graphic symbol) in the second level, with the inputs comming from outputs of the first gates.

4. A term with a single literal requires an inverter in the first level or may be complemented and applied as an input to the second level NAND gate.

Ex-9] $f(x,y,z) = \Sigma(0,6)$

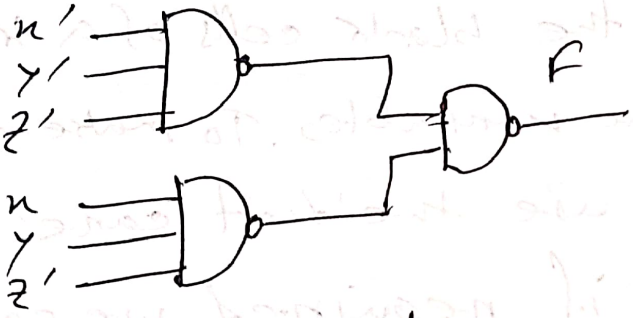


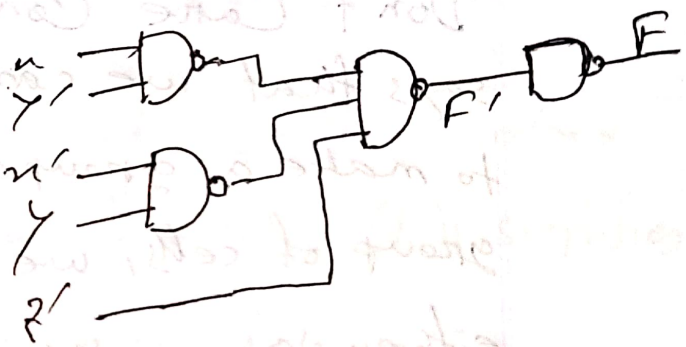
$F = (x'+y)(x+y')z'$

$\therefore F' = xy' + x'y + z$

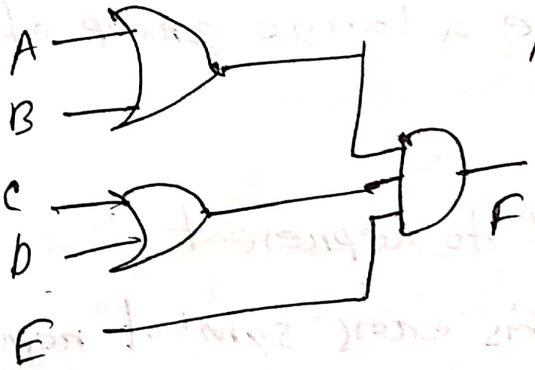
$\therefore F = x'y'z' + xyz'$

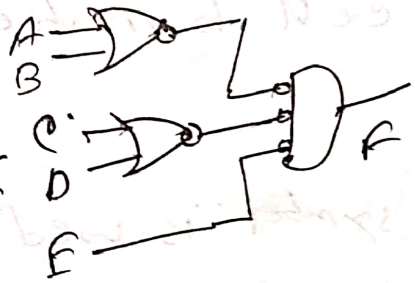
$F' = xy' + x'y + z$

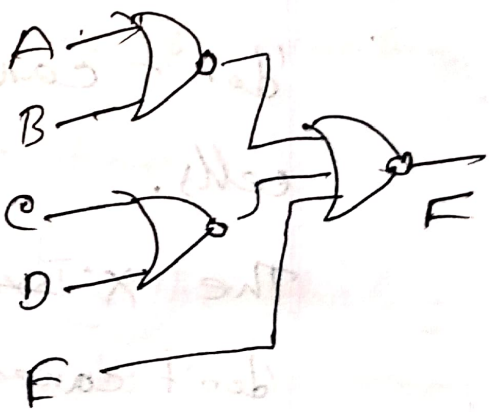




NOR Implementation: $F = (A+B)(C+D)E$







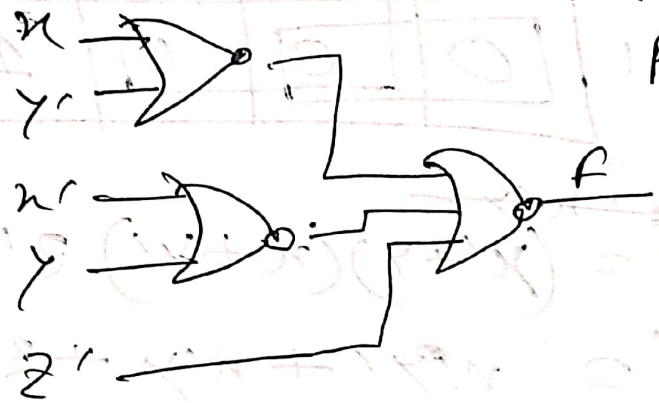
Three ways to implement $F = (A+B)(C+D)E$

Ex-10) $F(x,y,z) = \Pi(0,6)$ using product of sum;

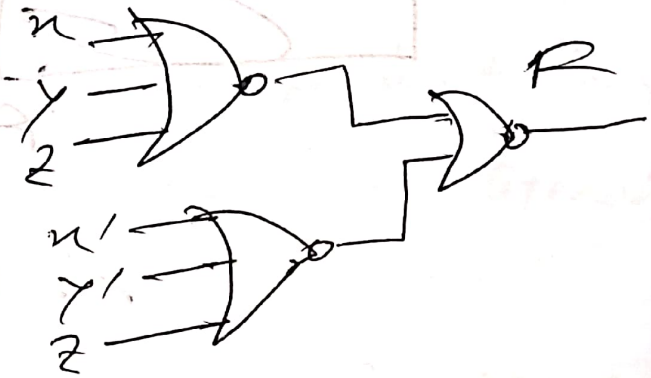
$F' = x'y + xy' + z$

$F = (x+y')(x'+y)z'$

$F = (x+y+z)(x'+y'+z)$



(a) $F = (x+y')(x'+y)z'$



(b) $(x+y+z)(x'+y'+z)$

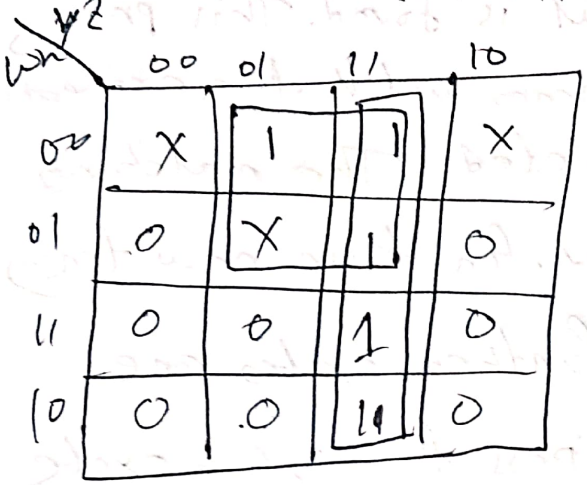
Don't Care Conditions: The don't care condition says that we can use the blank cells of a k-map to make a group of the variables. To make a group of cells, we can use the 'don't care' cells either '0' or '1', and if required, we can also ignore that cell. We mainly use the 'don't care' cell to make a large group of cells.

The 'X' ~~symbol~~ symbol is used to represent the 'don't care' cell in k-map. This cross symbol represents and invalid combination.

Ex-3.12 $F(w,x,y,z) = \Sigma(1,3,7,11,15)$ simplify

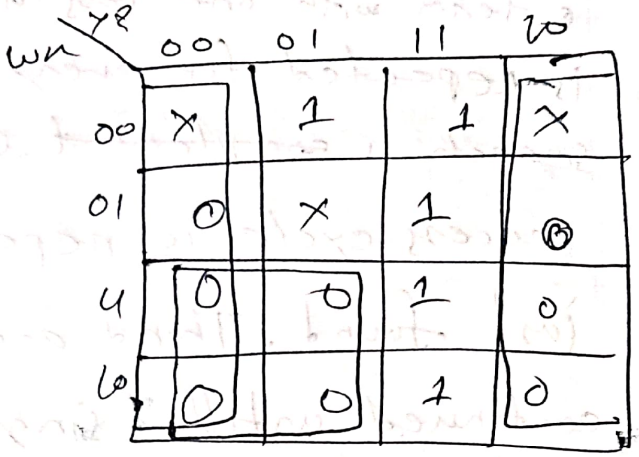
This and the don't care conditions:

$$d(w,x,y,z) = \Sigma(0,2,5)$$



SOP, $F = yz + w'z$

Combining 1's and X's



POS; $F = z(w'+y)$

Combining 0's and X's.

The Tabulation Method: The tabulation method is used to minimize the boolean functions. It is simplifies boolean expression into the simplified form using prime implecants. This method is convenient to simplify boolean expressions with more than 4 input variables. It uses an automatic simplification routine.

Determination of prime-~~im~~implicants: The starting point of the tabulation method is the list of minterms that specifies the function. The first tabular operation is to find the prime-implicants by using a matching

Process. This process compares each minterm with every other minterm. If two minterms differ in only one variable, that variable is removed and a term with one less literal is found. This process is repeated for every minterm until the exhaustive search is ~~complement~~ completed. The matching process cycle is repeated for those new terms just found. Third and further cycles are continued until a single pass through a cycle yeilds no further elimination of literals.

Example-3.14 $F(w,x,y,z) = \Sigma(1,4,6,7,8,9,10,11,15)$

(a)		(b)		(c)	
	w x y z				
1	0 0 0 1 ✓	(1,9)	- 0 0 1	(8,10,9,11)	1 0 - -
4	0 1 0 0 ✓	(4,6)	0 1 - 0	(8,9,10,11)	1 0 - -
8	1 0 0 0 ✓				
6	0 1 1 0 ✓	(8,10)	1 0 - 0 ✓		
9	1 0 0 1 ✓	(8,9)	1 0 0 - ✓		
10	1 0 1 0 ✓	(6,7)	0 1 1 -		
7	0 1 1 1 ✓	(9,11)	1 0 - 1 ✓		
11	1 0 1 1 ✓	(10,11)	1 0 1 - ✓		
15	1 1 1 1 ✓	(7,15)	- 1 1 1		
		(11,15)	1 - 1 1		

Prime Implicants

Decimal	Binary w x y z	Term
(1,9) 8	– 0 0 1	x'y'z
(4,6) 2	0 1 – 0	w'xz'
(6,7) 1	0 1 1 –	w'xy
(7,15) 8	– 1 1 1	xyz.
(11,15) 4	1 – 1 1	wyz
(8,10,9,11) (1,2)	1 0 – –	wx'

Now,

		1	4	6	7	8	9	10	11	15
x'y'z	(1,9)	✓					✓			
w'xz'	(4,6)		✓	✓						
w'xy	6,7			✓	✓					
xyz	(7,15)			✓	✓					✓
wyz	(11,15)								✓	✓
wx'	(8,10,9,11)					✓	✓	✓	✓	
		✓	✓	✓	✓	✓	✓	✓	✓	✓

$$F = x'y'z + w'xz' + wx' + xyz$$

We add xyz in our simplified term because 7, and 15 is not included in the other terms.

Ex 3.13) Given $F = \Sigma(0, 1, 2, 8, 10, 11, 14, 15)$

Determination of prime implicants:

(a)

	w	x	y	z	
0	0	0	0	0	✓
1	0	0	0	1	✓
2	0	0	1	0	✓
8	1	0	0	0	✓
10	1	0	1	0	✓
11	1	0	1	1	✓
14	1	1	1	0	✓
15	1	1	1	1	✓

(b)

	w	x	y	z	
(0, 1)	0	0	0	–	
(0, 2)	0	0	–	0	✓
(0, 8)	–	0	0	0	✓
(2, 10)	–	0	1	0	✓
(8, 10)	1	0	–	0	✓
(10, 11)	1	0	1	–	✓
(10, 14)	1	–	1	0	✓
(11, 15)	1	–	1	1	✓
(14, 15)	1	1	1	–	✓

(c)

	w	x	y	z
(0, 2, 8, 10)	–	0	–	0
(0, 8, 2, 10)	–	0	–	0
(10, 11, 14, 15)	1	–	1	–
(10, 14, 11, 15)	1	–	1	–

Prime implicants

Decimal	Binary w x y z	Terms
(0, 1) 1	0 0 0 –	w' x' y'
(0, 2, 8, 10) (2, 8)	– 0 – 0	x' z'
(10, 11, 14, 15) (1, 4)	1 – 1 –	w y

Now;

		0	1	2	8	10	11	14	15
✓ w'x'y'	(0, 1)	✓	✓						
✓ x'z'	0, 2, 8, 10	✓		✓	✓	✓			
✓ wy	(0, 11, 14, 15					✓	✓	✓	✓
		✓	✓	✓	✓	✓	✓	✓	✓

$\therefore f = w'x'y' + x'z' + wy$

Combinational Logic (4)

Combinational Circuit: Combinational circuit is a circuit in which we combine the different gates in the circuit. It consists of input variables, logic gates, and output variables.
For example: Encoder, Decoder, multiplexer, and demultiplexer.

Block diagram:

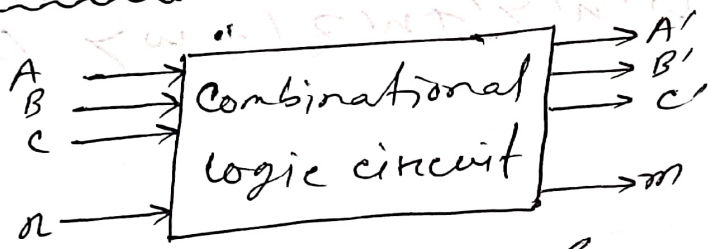


Figure: Block diagram of a combinational circuit

The n input variables come from external source. The m output variables go to an external destination.

Design procedure: The design of combinational circuit starts from the verbal outline of the problem and ends in a logic circuit diagram, or a set of boolean functions from which the logic diagram can be easily obtained.
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 relationships between inputs and outputs is derived.
5. The simplified boolean function for each output is obtained.
6. The logic diagram is drawn.

Adder: Adder is a device that will add together two or three bits and give the result as output.

Half adder: A half adder is a combinational circuit that performs the arithmatic addition of two bits. It consists of two binary inputs such as A and B and two binary output such as S (for sum) and C (for carry). In half adder sum output will be taken from X-OR gate and carry output will be taken from ~~X-OR~~ AND gate.

From the truth table, we see that

Table: Truth table for half adder.

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

The simplified boolean functions for the two outputs can be obtained directly from the truth table. The simplified sum of product expression are,

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

$$C = AB$$

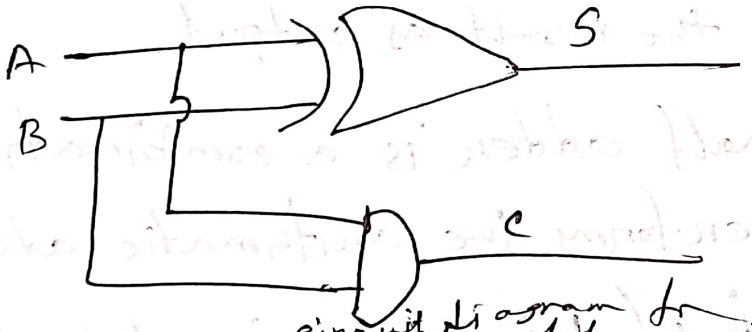


Figure 1: circuit diagram for Half Adder

Full Adder: A full adder is a combinational circuit that performs the arithmatic addition of three bits. It consists of three binary inputs and two binary outputs. The three binary inputs one denoted by A, B, Ci and

and two binary outputs are denoted by S (for sum) and C_o (for carry). In full adder sum output will be taken from X-OR gate and carry output will be taken from OR gate.

From the truth table we see that

Table-02: Truth table for full adder.

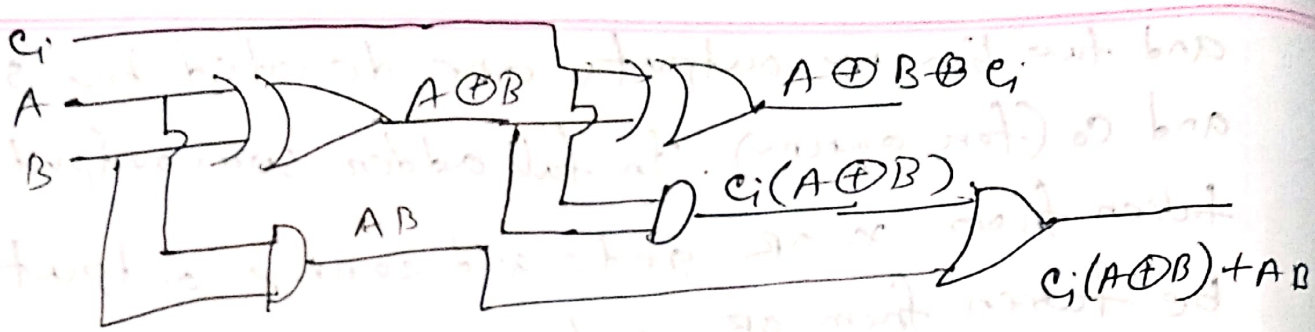
A	B	C_i	S	C_o
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

The simplified boolean functions for the two outputs can be obtained directly from the truth table.

The simplified SOP expressions are

$$S = A'B'C_i + A'BC_i' + AB'C_i' + ABC_i$$
$$= C_i(A'B' + AB) + C_i'(BA'B + AB')$$
$$= C_i(A \odot B) + C_i'(A \oplus B)$$
$$= A \oplus B \oplus C_i$$

$$C_o = A'BC_i + AB'C_i + ABC_i' + ABC_i$$
$$= C_i(A \oplus B) + AB$$



Subtractor: Subtractor is an electronic logic circuit that generates an output which is the substraction of two binary numbers, the minuend and the number to be subtracted which is called subtrahend. It gives out two outputs, difference and borrow.

Half subtractor: The half subtractor is a combinational circuit which is used to perform subtraction of two bits. It has two inputs, A (minuend) and B_i (subtrahend) and two output D(difference) and B_o(borrow). The truth table is shown in bellow:

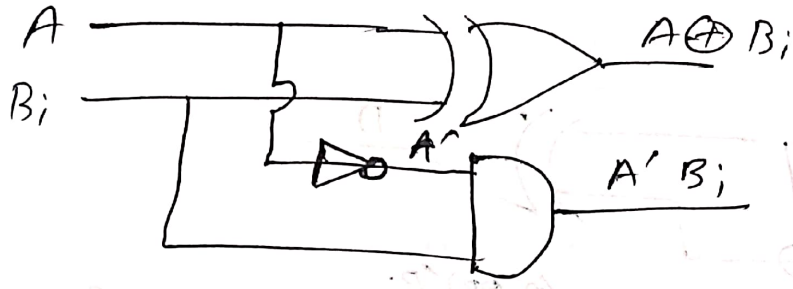
Table-01: Truth table for half subtractor.

A	B_i	D	B_o
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

From the above truth table we can find the boolean expression.

$$D = A'B_i + AB_i' = A \oplus B_i$$

$$B_o = A'B_i$$



Full subtractor: A full subtractor is a combinational circuit that performs subtraction involving three bits, namely A (minuend), B (subtrahend) and B_{in} (borrow-in). It accepts three inputs A, B and B_{in}, and it produces two outputs: D (Differences) and B_{out} (borrow out). The truth table are shown bellow:

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

From the above truth table we can find the boolean express

$$D = A'B'B_{in} + A'BB_{in}' + AB'B_{in}' + ABB_{in}$$

$$= B_{in}(A \oplus B) + B_{in}'(A \oplus B)$$

$$= A \oplus B \oplus B_{in}$$

$$B_{out} = A'B'B_{in} + A'B\,B_{in} + A'B\,B_{in} + AB\,B_{in}$$
$$= B_{in}(A \odot B) + A'B$$
$$= B_{in}(A \oplus B)' + A'B$$

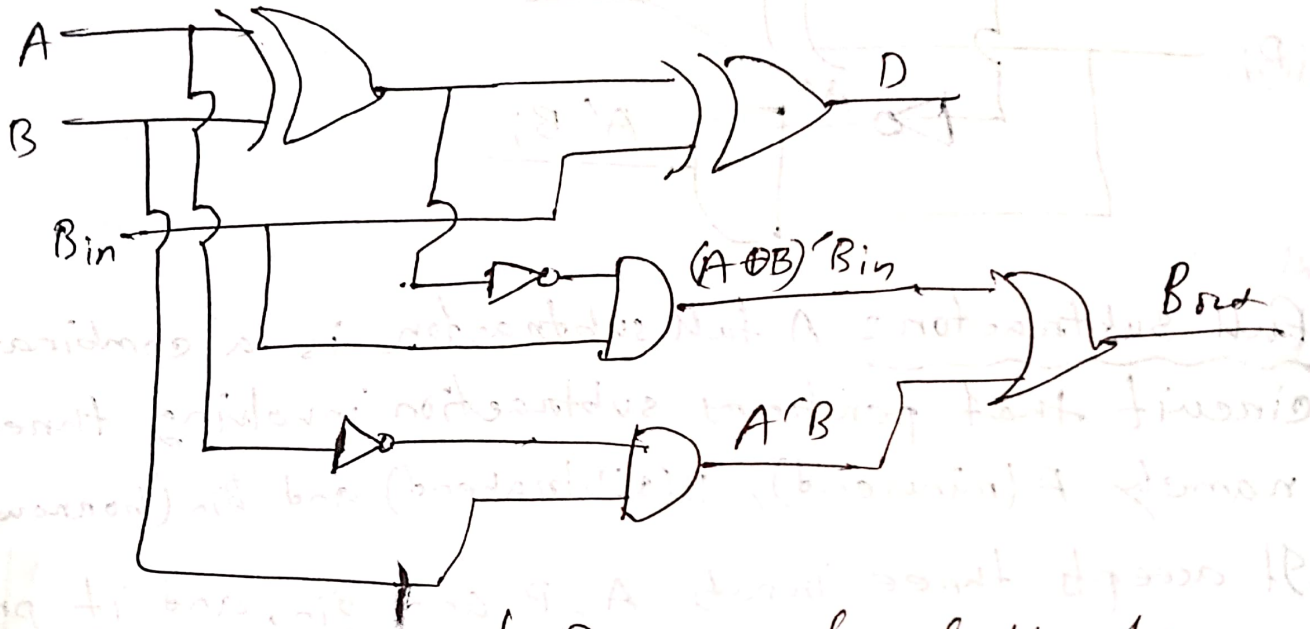


Figure-02: Circuit Diagram for full adder.

BCD to excess-3 code converter:

BCD stands for Binary coded decimal is a a class of binary encodings of decimal ~~a~~ numbers where each digit is represented by a ~~fixed~~ four number of bits.

The excess-3 code can be calculated by adding 3 to each four digit BCD code.

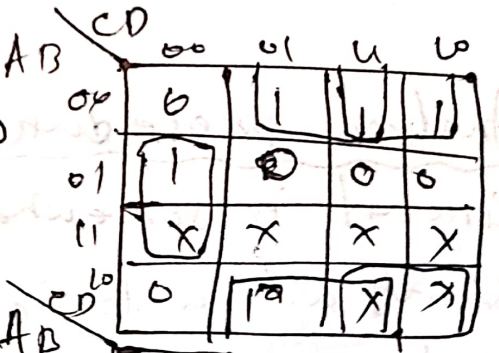
Bellow is the truth table.

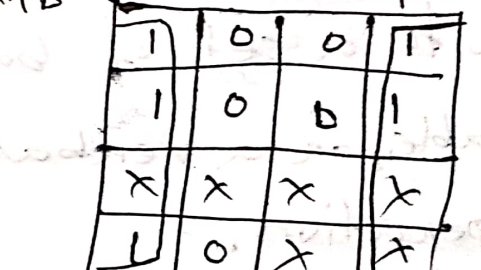
58 → BCD → 0101 1000

Table-1: Truth table for code conversion example.

Input BCD				Output Excess-3 code			
A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

We note that four binary variables may have 16 bit combinations, only 10 of which are listed in the truth table. The six bit combinations not listed in the for the input variables are don't care combination





$z = D'$

$y = CD + C'D' = CD + (C+D)'$

$x = B'C + B'D + BC'D' = B'(C+D) + B(C+D)'$

$w = A + BC + BD = A + B(C+D)$

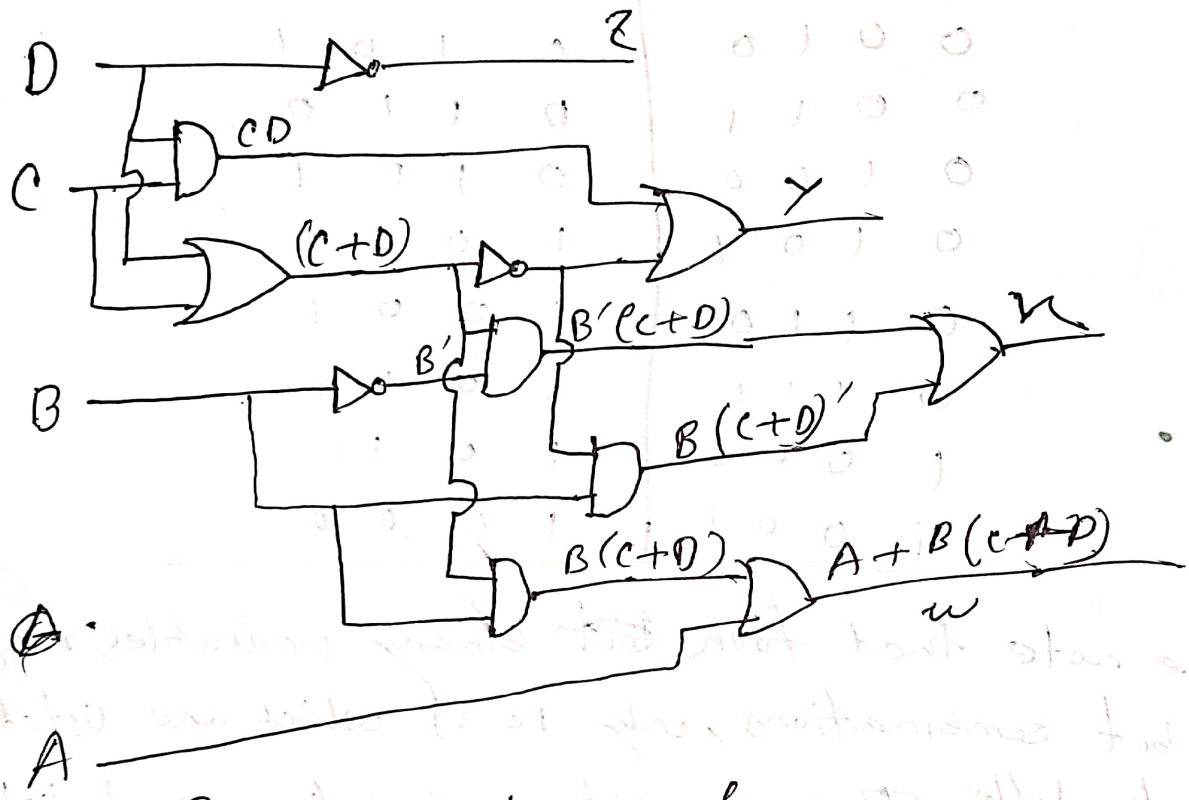


Figure: Logic diagram for BCD to excess-3 code converter.

Analysis procedure: The analysis of a combinational circuit is somewhat the reverse process. It starts with a logic diagram and culminates with a set of boolean functions, a truth table, or a verbal explanation of the circuit operation.

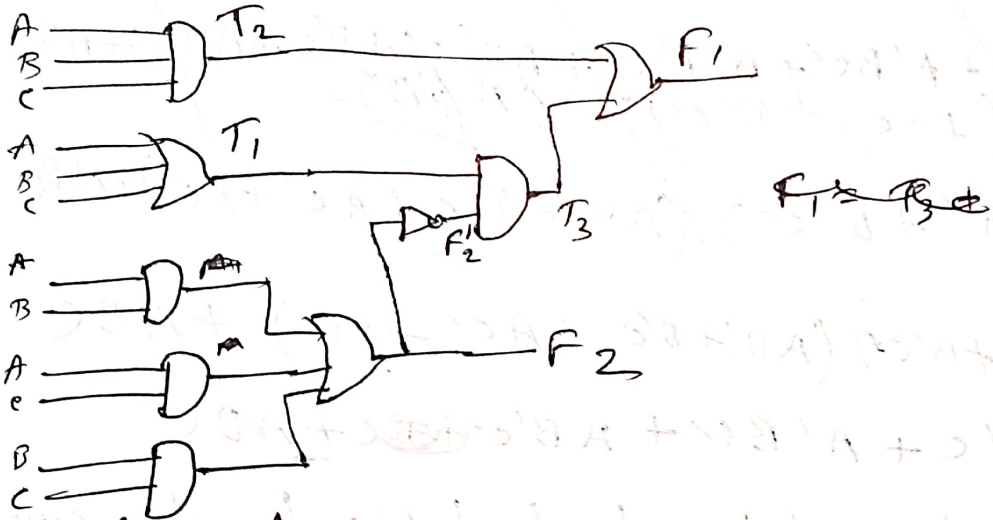
To obtain the output Boolean functions from a logic diagram, proceed as follows:

1. Label with arbitary symbols all gate outputs that are a function of the input variables. obtain the boolean function for each gate.

2. Label with other arbitary symbols those gates which are a function of input variables and or previously labeled gates.

3. Repeat the process outlined in step 2 until the outputs of the circuit are obtained.

4. By repeated substitution of previously defined functions, obtain the output boolean functions in term of input variables only.



We notice that the circuit has three binary inputs A, B, and C and two binary outputs F_1 and F_2. the outputs of various gates are labeled with intermediate symbols. The outputs of gates that are a function of input variables only are

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

f_2, T_1 and T_2. The Boolean functions for these three outputs are:

$$F_2 = AB + AC + BC$$

$$T_1 = A + B + C$$

$$T_2 = ABC$$

Next we consider outputs of gates which are a function of already defined symbols.

$$T_3 = F_2' T_1$$

$$F_1 = T_3 + T_2$$

$$= F_2' T_1 + ABC$$

$$= (AB + AC + BC)'(A + B + C) + ABC$$

$$= (A'+B')(A'+C')(B'+C')(A+B+C) + ABC$$

$$= (A' + A'C' + B'C')(AB' + B'C + AC' + BC') + ABC$$

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

$$= (A'(1 + C') + B'C')(A(B + B') + \dots$$

$$= (A' + B'C')(AB' + B'C + AC' + BC') + ABC$$

$$= (A' + B'C')(AB' + B'C + AC' + BC') + ABC$$

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

The derivation of the truth table for the circuit is a straight forward process once the output boolean functions are known. To obtain the

Truth table directly from the ~~boolean function~~ logic diagram without going through the derivations of the boolean functions, proceed as follows:

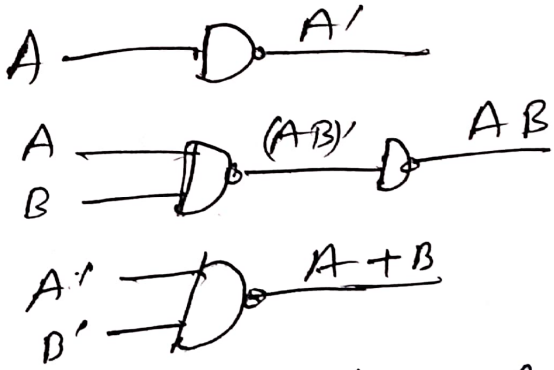
1. Determine the number of input variables to the circuit. For n inputs, form the 2^n possible input combinations of ~~the~~ 1's and 0's by listing the binary numbers from 0 to $2^n - 1$

4-7 Multilevel NAND Circuit:

Combinational circuits are more frequently constructed with NAND and NOR gates rather than AND and OR gates. NAND and NOR gates are more common from the hardware point of view because they are readily available in integrated circuit form.

Universal gate: The NAND gate is said to be a universal gate because any digital system can be implemented with it. Combinational circuits and sequential circuits as well can be constructed with this gate.

Any boolean function can be implemented with NAND gates. We need only show that the logical operation, AND, OR and NOT can be implemented with NAND gate.



Implementation of AND, OR and NOT operations with NAND gates.

Boolean function implementation - Block diagram method

The implementation of Boolean functions with NAND gates may be obtained by means of a simple block diagram manipulation technic. Procedure are as follows:

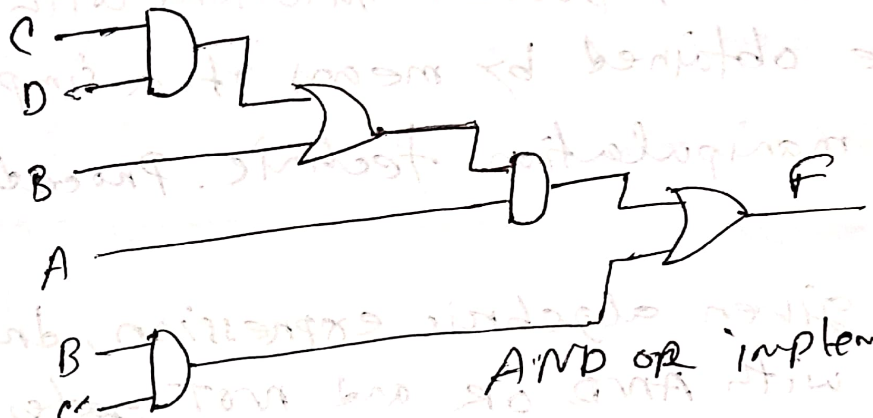
1. From the given algebric expression, draw the logic diagram with AND, OR and NOT gates, Assume that both the normal and complement inputs are available.

2. Draw ~~the~~ a second logic diagram with the equivalent NAND logic for each AND, OR and NOT gate.

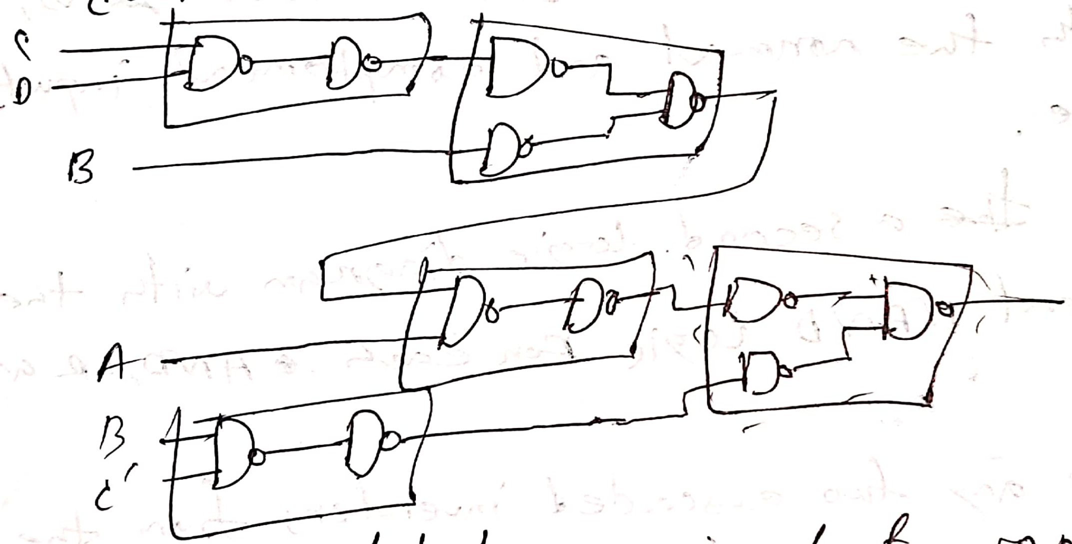
3. Remove any two cascaded inverters from the diagram. Since double inversion doesnot perform a logic function. Remove inverters connected to single external inputs and complement the corresponding input variable.

The new logic diagram obtained is the required NAND gate implementation

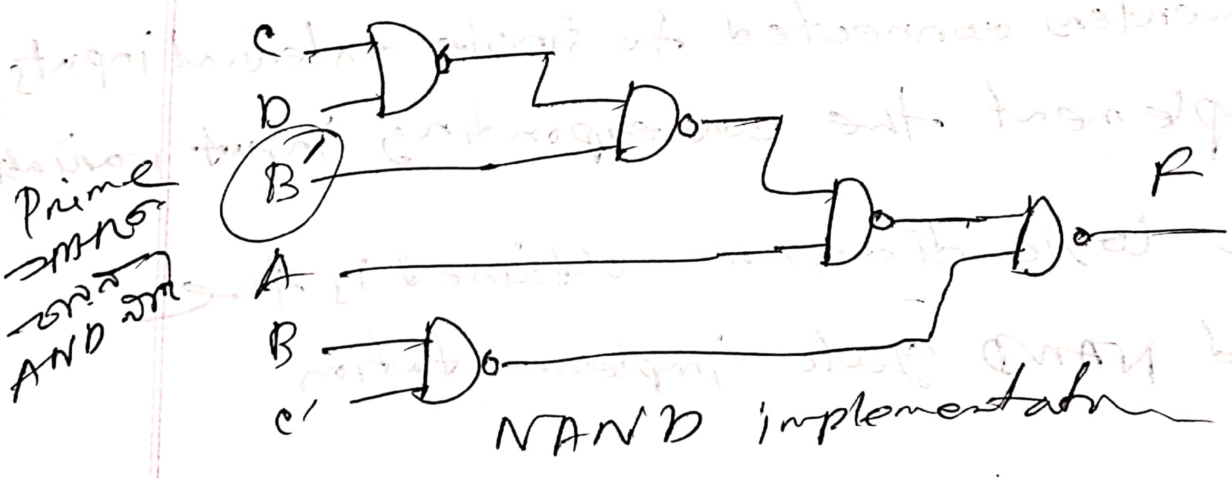
Ex: $F = A(B+CD)+BC'$



AND OR implementation

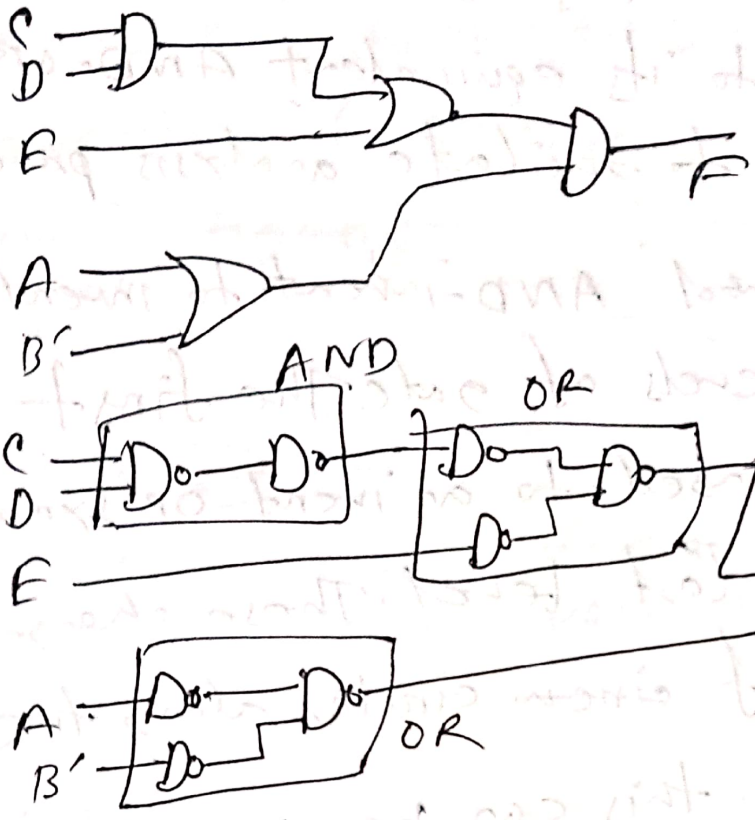


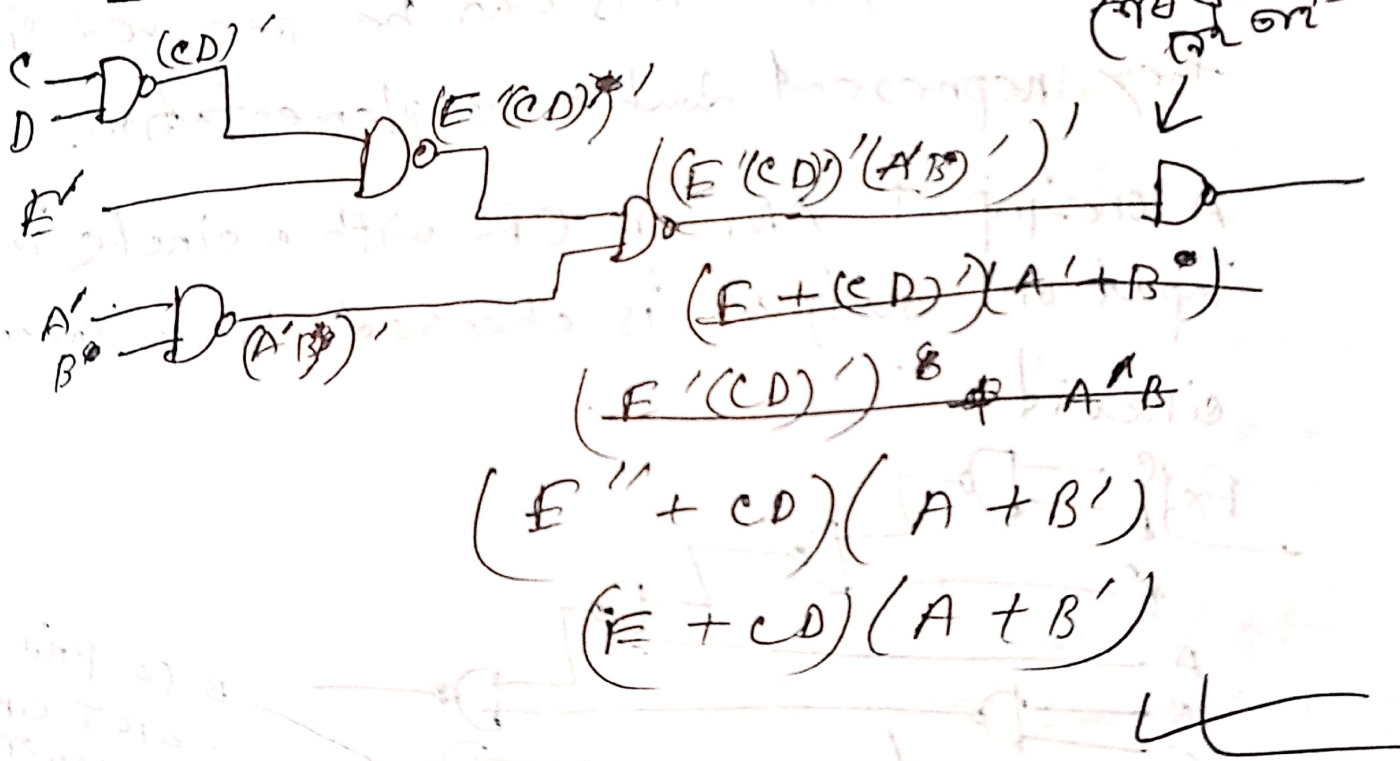
substituting equivalent NAND functions



NAND implementation

Ex: $F = (A+B')(CD+E)$





$(E'(CD)')'(A'B')'$

~~$(E+(CD)')(A'+B')$~~

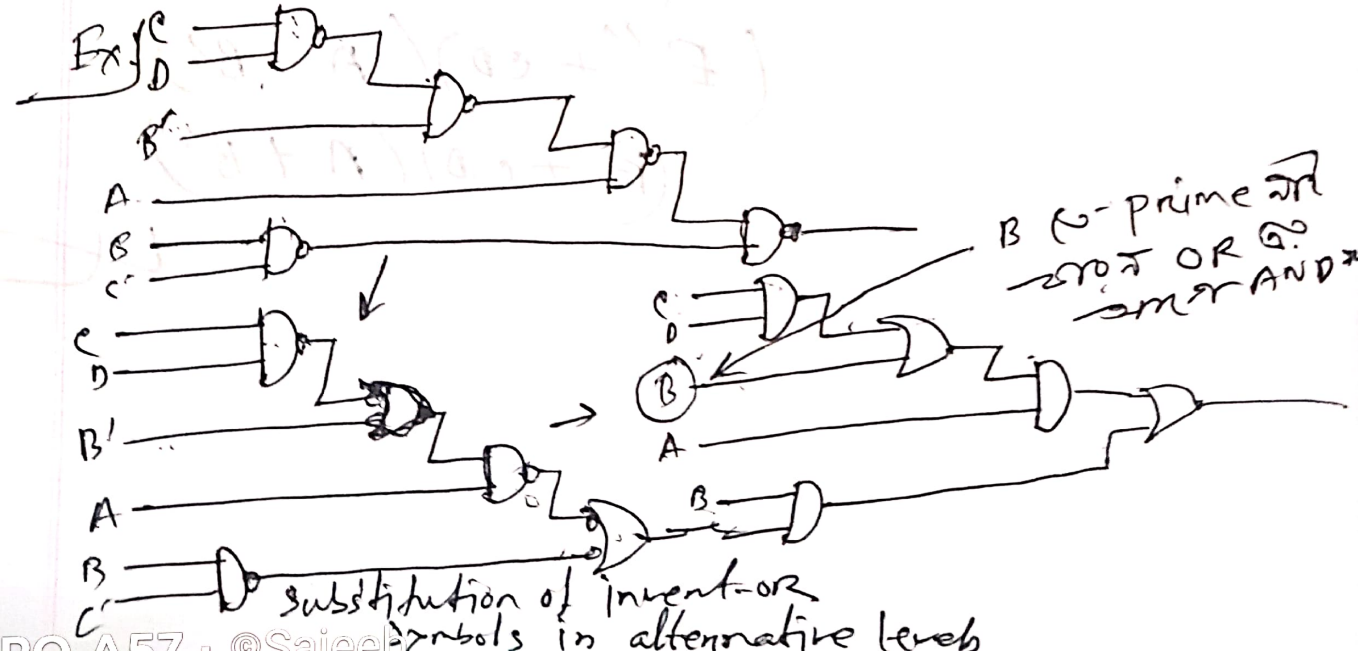
~~$(E'(CD)')' + A'B$~~

$(E''+CD)(A+B')$

$(E+CD)(A+B')$

Block diagram transformation: It is sometimes convenient to convert a NAND logic diagram to its equivalent AND-OR logic diagram to facilitate analysis procedure. For this, ~~so~~ symbol AND-invert to invert-OR in alternative levels of gate. The first level to be changed to an invert-OR symbol should be the last level. These changes produce pairs of ~~circu~~ circles along the same line, and this can be removed since they represent double complementation.

A one-input AND or OR with a circle in the input or output is changed to an inverter circuit.



Universal Gate: The NOR Gate is universal gate because any boolean function can be ~~implet~~ implemented with it, including a flip flop circuit.

The coversion of AND, OR and NOT to NOR is shown in figure:

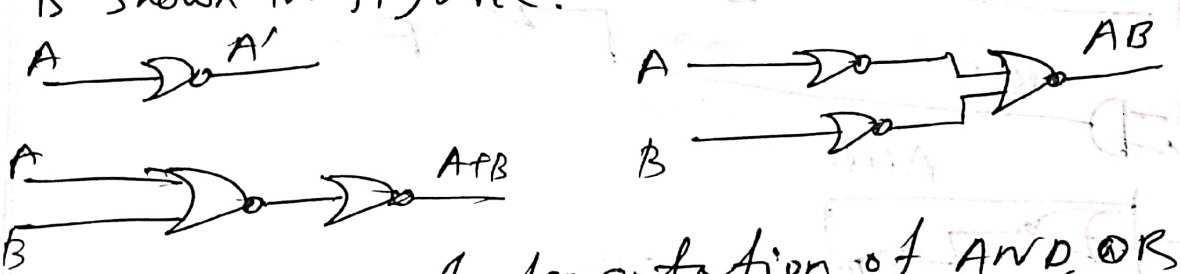


Figure: Implementation of AND, OR and NOT using NOR gate.

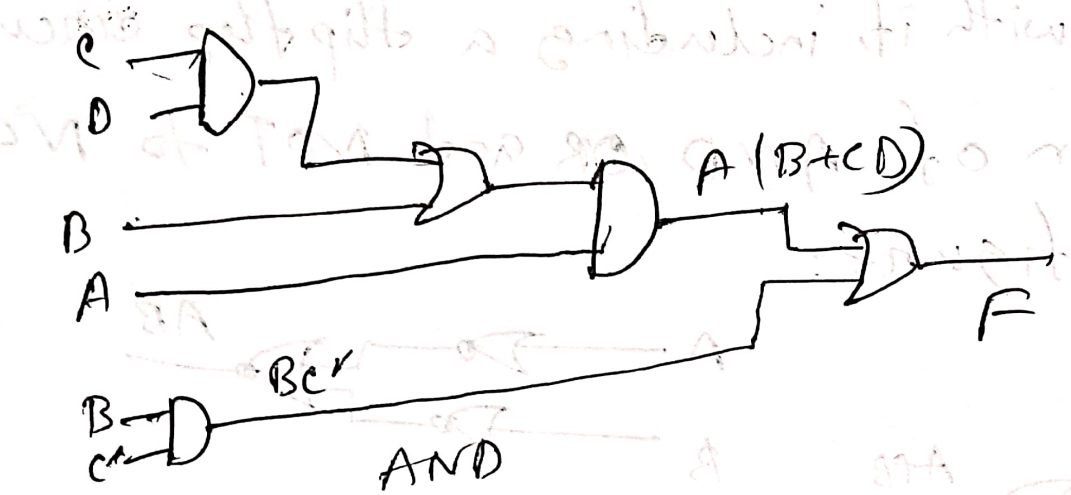
The block diagram procedure for implementing Boolean function with NOR gates is similar to the procedure outlined in the previous section for NAND gates.

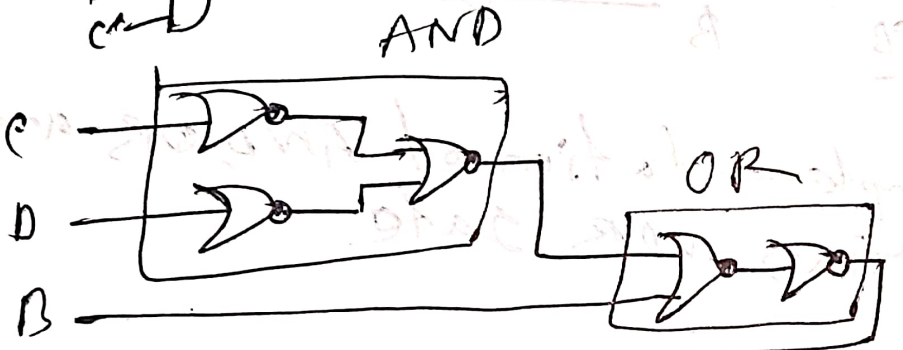
1. Draw the AND-OR logic diagram from the given algebric expression. Assume that both normal and the complement inputs are available.

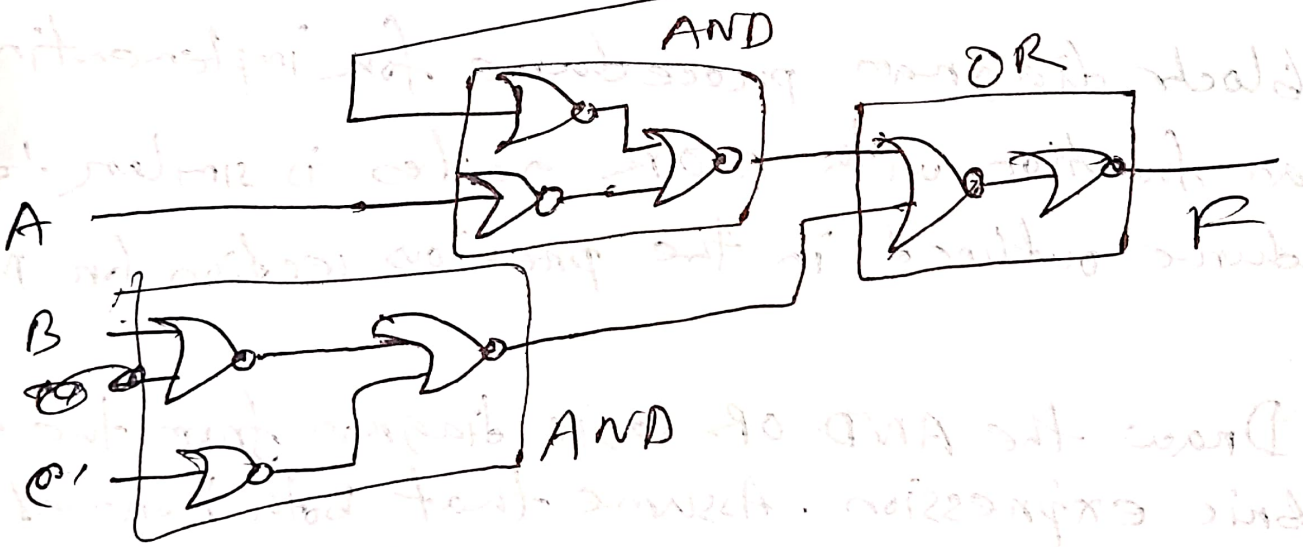
2. Draw a second logic diagram with equivalent NOR logic, substituted for each AND, OR and NOT gate.

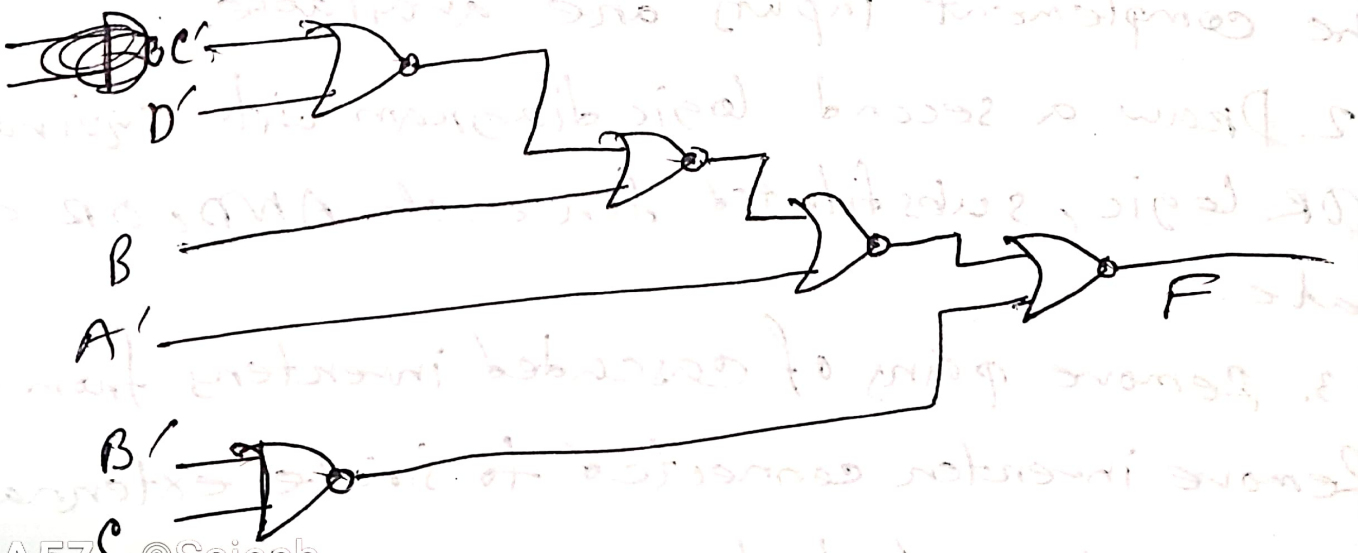
3. Remove pairs of cascaded inverters from the diagram. Remove inverters connected to single external inputs and complement ~~to~~ the corresponding input variable

Ex: $F = A(B+CD) + BC'$



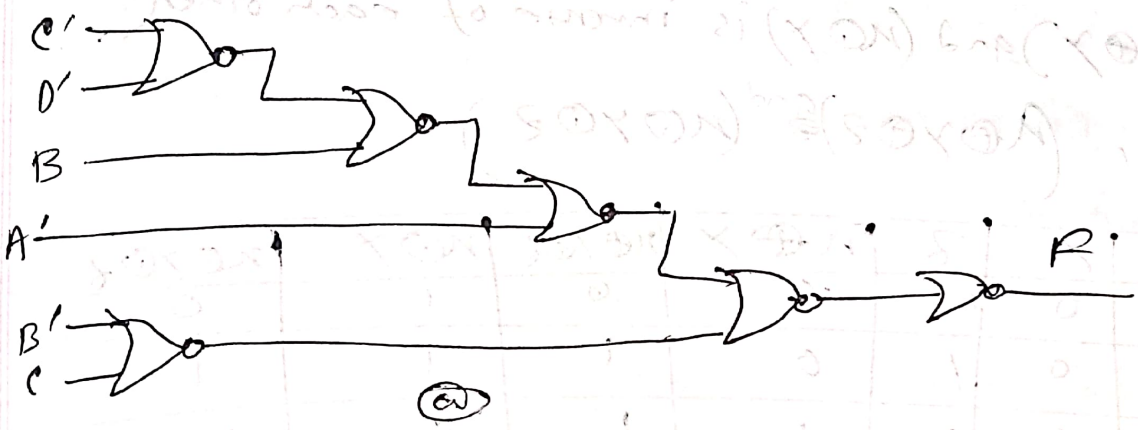




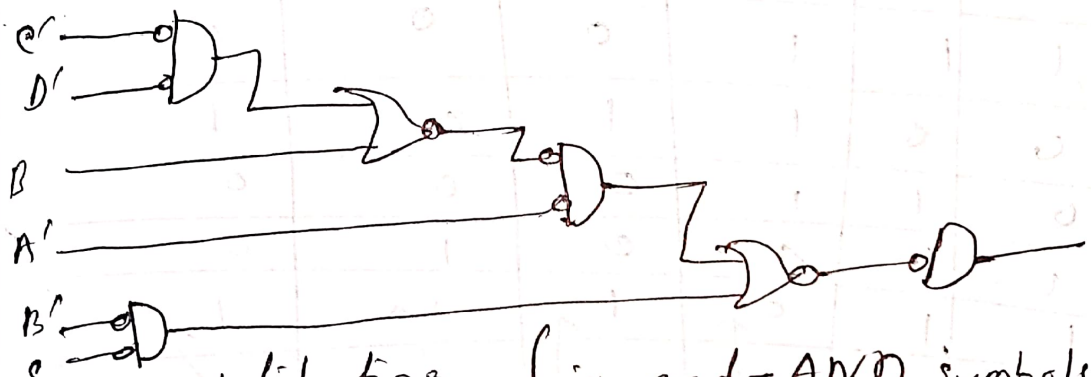


Block diagram Transformations: The conversion of a NOR logic diagram to an AND-OR diagram is achieved through a change in symbols from OR-invert to invert-AND starting from the last level and in alternative levels.

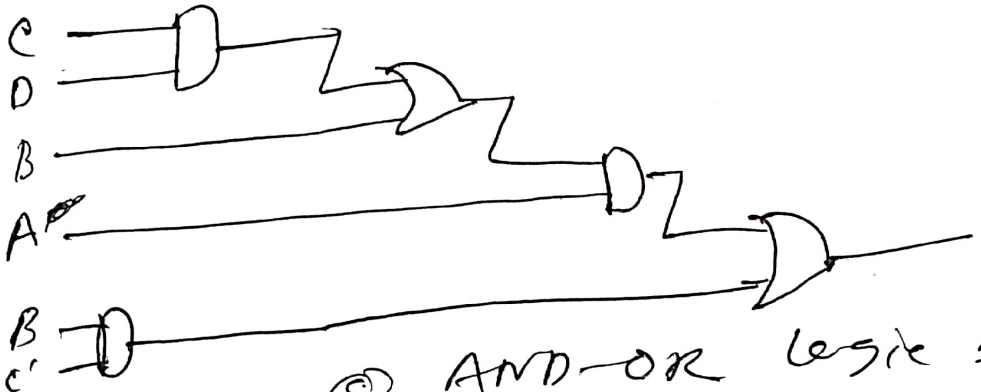
A one input AND or OR gate is removed but if it has a small circle at the input or output, it is converted into an inverter.



(a)



(b) Substituting of invert-AND symbols in alternative levels



(c) AND-OR logic sat-

4.9] Exclusive-OR and ~~Equivalence~~ Equivalence function:

Exclusive-OR and Equivalence denoted by $\oplus$ and $\odot$ respectively.

Binary operation, $x \oplus y = xy' + x'y$

$x \odot y = xy + x'y'$

$(x \oplus y)$ and $(x \odot y)$ is inverse of each other.

But, $(x \oplus y \oplus z)$ and $(x \odot y \odot z)$

x	y	z	$x \oplus y$	$x \oplus y \oplus z$	$x \odot y$	$x \odot y \odot z$
0	0	0	0	0	1	0
0	0	1	0	1	1	1
0	1	0	1	1	0	1
0	1	1	1	0	0	0
1	0	0	1	1	0	1
1	0	1	1	0	0	0
1	1	0	0	0	1	0
1	1	1	0	1	1	1

Parity Bit: A parity bit is an extra bit included with a binary message to make the number of 1's either even or odd. The message including the parity bit, is transmitted and then checked at the receiving end for errors.

Parity generation: The circuit that checks the parity in the receiver is called a **parity checker.**

The circuit that generates the parity bit in the transmitter is called a parity generator.

Example: consider a three bit message to be transmitted with an odd parity bit. Shown in table bellow:

Table: Odd-parity generation

Three-bit message			Parity bit generated
x	y	z	P
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

equation of P,

$= x'y'z' + x'yz + xy'z + xyz$

$= x'(y \odot z) + x(y \oplus z)$

$= x'(y \odot z) + x(y \odot z)'$

$= x \oplus (y \odot z)$

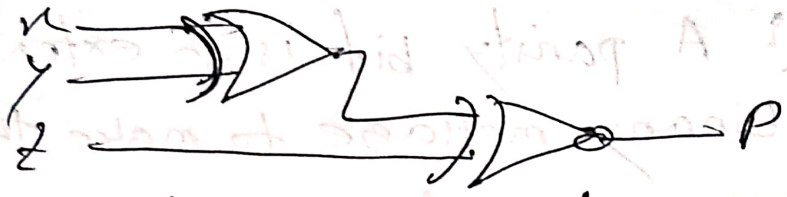
OR,

$= z'(x'y' + xy) + z(x'y + xy')$

$= z'(x \odot y) + z(x \oplus y)$

$= z'(x \odot y) + z(x \odot y)'$

$= (x \odot y) \oplus z$



3 bit parity generation.

Parity checker: Three bit message and the parity bit are transmitted to their destination, where they are applied to a parity-checker circuit. An error occurs during ~~transf~~ transmission if the parity of four bit received is even, since the binary information transmitted was originally odd. The output C of parity checker should be a 1 when an error occurs, Otherwise '0'

Table: ~~Odd~~ Truth table for odd parity checker circuit.

Four bit receives x y z P	Parity error check C
0 0 0 0	1
0 0 0 1	0
0 0 1 0	0
0 0 1 1	1
0 1 0 0	0
0 1 0 1	1
0 1 1 0	1
0 1 1 1	0
1 0 0 0	0
1 0 0 1	1
1 0 1 0	1
1 0 1 1	0
1 1 0 0	1
1 1 0 1	0
1 1 1 0	0
1 1 1 1	1

From the truth table we see that the function for C consists of the eight minterms with numerical values having an even number of 0's. So the function can be expressed as, $C = x \odot y \odot z \odot p$

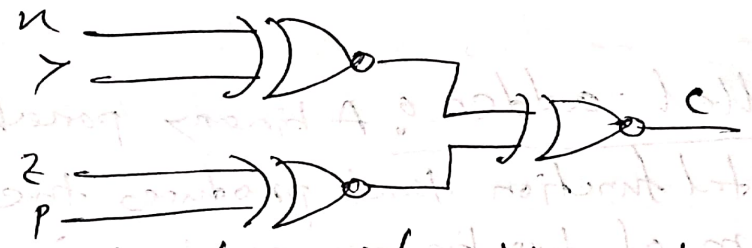


figure: 4 bit ~~parity~~ odd-parity checker