ON THREE VARIABLE NOR/NAND LOGIC WHEN COMPLEMENTED LITERALS ARE NOT AVAILABLE

A Dissertation
submitted in partial fulfilment
of the requirements for the Degree
of
MASTER OF ENGINEERING
in
ELECTRONICS & COMMUNICATIONS ENGINEERING
(APPLIED ELECTRONICS & SERVOMECHANISMS)

By
P. SRIRAMARAO

DEPTT. OF ELECTRONICS & COMMUNICATIONS ENGINEERING
UNIVERSITY OF ROORKEE
ROORKEE
1966

C E R T I F I C A T E

Certified that the dissertationentitled " THREE VARIABLE NOR/ NAND LOGIC WHEN COMPLEMENTED LITERALS ARE NOT AVAILABLE" which is being submitted by Shri P.Sri Rama Rao in partial fulfilment for the award of the Degree of Master of Engineering in Applied Electronics and Servomechanism of the University of Roorkee is a record of student's own work carried out by him under my supervision and guidance. The matter embodied in this dissertation has not been submitted for the award of any other Degree or Diploma.

This is to further certify that he has worked for a period of 7 months from 1.1.66 to 5.8.66 for preparing this thesis for Mas ter of Engineering at the University.

(N.N.Biswas)
Professor
Electronics and Communi cation
Engineering Department
University of Roorkee
Roorkee.

August , 1966.

A C K N O W L E D G E M E N T S

The author is highly indebted to Dr. N.N.Biswas, Professor , Electronics and Communication Engineering Department , University of Roorkee, Roorkee for the inspiration and valuable guidance given by him throughout the course of this work.

S Y N O P S I S

The implementation of NOR and NAND Logics in Digital computers has assumed great importance because it implies the use of single type of circuit in logic part of the computer. Economical use of the number of logic elements and to minimise the connections is the main problem that confronts the designer. Dr. Lee Hellerman of IBM Corporation, has given a catalogue of minimal three variables NOR and NAND logic circuits, obtained by programming on Digital computer. Here in this work these circuits have been studied and an attempt is made to find out a systematic method by which we can arrive at these minimal NOR and NAND circuits of three variables assuming complemented literals are not available. Some rules have been suggested which help in implementing a given three variable function by minimum NAND circuits. Possibilities of extension of the same methods to four variable functions are discussed.

C O N T E N T S

CHAPTER I

INTRODUCTION

INTRODUCTION

1.1. GENERAL

Implementation of NOR and NAND logics has gained much importance due to their increasing use in Digital circuits. The main concern of the designer is how to achieve minimal implementation of these logics. Methods have been developed for implementing a function by NOR and NAND logics. Besides economy, additional advantages of using NOR and NAND logics are :

1. Considerable less test equipment is needed to test one device than several.
2. Production of equipment is eased.
3. Fault location and repair of equipment is eased.

Use of NOR as well as NAND logic implies use of only one type of circuit. Thus with the use of these type of logics above mentioned advantages are exploited to full extent.

Justification behing the use of NOR and NAND function lies in the fact that each may be individually used to give all other Boolean functions. Hence NOR

and NAND logics are called universal logics. Either NOR ir NAND logics could be used to represent all the operations.

HOw , OR , AND, and NOT are implemented by only NOR or NAND logics is shown below :

a. Using NOR operation only.

NOT — $A \downarrow A = \overline{A + A} = \bar{A}$

OR — $(A \downarrow B) \downarrow = \overline{\overline{A+B}} = A+B$

AND — $(A\downarrow)\downarrow(B\downarrow) = \overline{\bar{A} + \bar{B}} = AB$

b. Using NAND logic only.

NOT — $A|A = \overline{A.A} = \bar{A}$

OR — $(A|)|(B|) = \overline{\bar{A}\,\bar{B}} = A+B$

AND — $(A|B) = \overline{\overline{AB}} = AB$

This shows the universality of NOR and NAND logics.

1.2. <u>PRACTICAL IMPORTANCE OF NOR AND NAND LOGICS</u>

Transistor circuits of NOR and NAND have the following characteristics in addition to their being universal logics.

1. Each circuit has current as well as voltage gain and also level setting ability.

1.4 In Chapter II a review of the methods so far available for the implementation of NOR and NAND logics has been presented. I

Maley and Earle's transform method, Map method have been discussed and examples have been given in each case.

In Chapter III new approach for the implementation of NOR and NAND logics suggested by Arjun Godwani in his M.E. thesis 1965 has been reviewed.

In Chapter IV Lee Hellerman's work of obtaining catalogue of minimal NOR and NAND circuits has been reviewed. After programming on a digital computer he has given 80 minimal circuits for NOR and Nand three variable logics assuming that complented literals are not available. So far no definit methods are available to implement the given function by minimal NOR or NAND logics. Now that catalogue of minimal circuits is available, this can is studied and some systematic methods are developed in this work and these have been given in Chapter V.

—

and NAND logics are called universal logics. Either NOR ir NAND logics could be used to represent all the operations.

HOw , OR , AND, and NOT are implemented by only NOR or NAND logics is shown below :

a. Using NOR operation only.

NOT — $A \downarrow A = \overline{A + A} = \bar{A}$

OR — $(A \downarrow B) \downarrow = \overline{\overline{A+B}} = A+B$

AND — $(A\downarrow)\downarrow(B\downarrow) = \bar{A} \downarrow \bar{B} = AB$

b. Using NAND logic only.

NOT — $A|A = \overline{A.A} = \bar{A}$

OR — $(A|)|(B|) = \overline{\bar{A}\,\bar{B}} = A+B$

AND — $(A|B) = \overline{\overline{AB}} = AB$

This shows the universality of NOR and NAND logics.

1.2. PRACTICAL IMPORTANCE OF NOR AND NAND LOGICS

Transistor circuits of NOR and NAND have the following characteristics in addition to their being universal logics.

1. Each circuit has current as well as voltage gain and also level setting ability.

2 Number of inputs to a block can be increased easily by parallelling the circuits which in most of the cases amounts to connecting the collectors.

3 There are practically no circuit constraints except for the fan in an fan out.

4 Sequential circuit design becomes easy because of the fact that no amplifiers need be put in the feedback loops.

1.3. DESIGNING CONSIDERATIONS

The main consideration while implementing NOR and NAND logics is to minimise the number of logic blocks and reduce the connections. At the outset it may appear that more elemental blocks would be necessary to perform a given logic function through NOR and NAND logic than by using English logic (AND , OR NOT) . This is the outcome of the idea that large number of elements will be used solely as inverters. But this is not so and on the otherhand if we can exploit the basic logic power of the NAND (and NOR) function beyond that of Boolean connectives, there will be reduction in number of logic elements and connections. In general an extensive logical net can be found using NOR and NAND logic employing no more elements than would be necessary in English logic system.

1.4 In Chapter II a review of the methods so far available for the implementation of NOR and NAND logics has been presented. I

Maley and Earle's transform method, Map method have been discussed and examples have been given in each case.

In Chapter III new approach for the implementation of NOR and NAND logics suggested by Arjun Godwani in his M.E. thesis 1965 has been reviewed.

In Chapter IV Lee Hellerman's work of obtaining catalogue of minimal NOR and NAND circuits has been reviewed. After programming on a digital computer he has given 80 minimal circuits for NOR and Nand three variable logics assuming that complented literals are not available. So far no definit methods are available to implement the given function by minimal NOR or NAND logics. Now that catalogue of minimal circuits is available, this is studied and some systematic methods are developed in this work and these have been given in Chapter V.

—

CHAPTER II

IMPLEMENTATION OF NOR AND NAND LOGICS - TRANSFORM METHOD, MAP METHOD

IMPLEMENTATION OF NOR AND NAND LOGICS

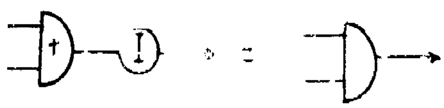
2.1. DEFINITION OF NOR AND NAND LOGICS

NOR : Output is present if and only if all the inputs are zero or low.

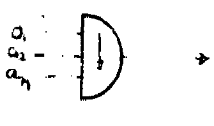
Interpretation in Boolean Algebra

One variable	$\bar{A}$	
Two variable	$\overline{A+B}$	$= \bar{A}\bar{B}$
Three variable	$\overline{A+B+C}$	$= \bar{A}\bar{B}\bar{C}$
n variables	$\overline{A_1+A_2+..A_n}$	$= \bar{A}_1\bar{A}_2....\bar{A}_n$

NOR is equivalent to Boolean function OR followed by an inverter



Symbol of NOR logic is as shown here



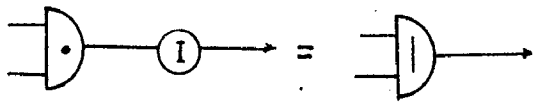
We can also find out from the truth table the property of NOR logic

a	b	$\overline{a+b}$
0	0	1
0	1	0
1	0	0
1	1	0

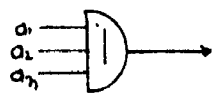
$\overline{a+b} = \overline{a}\,\overline{b}$

As we see here above, the output is present if and only if all the inputs are zero.

NAND FUNCTION: This is equivalent to Boolean network of AND followed by Inverter



Definition: For NAND network output is present if and only if atleast one of the inputs is zero. The symbol for NAND logic is shown below :



$\overline{a_1 a_2 \cdots a_n}$

It can also be explained by the truth table :

a	b	$\overline{ab}$
0	0	1
0	1	1
1	0	1
1	1	0

Output $\overline{ab} = \bar{a} + \bar{b}$

As is evident from the truth table output is present if and only if atleast one of the inputs is zero.

To put in mathematical language, for n variables

$A_1 \downarrow A_2 \downarrow \ldots\ldots \downarrow A_n = 1$ if and only if

$$\sum_{i=1}^{n} A_i = 0$$

For NAND logic for n variable case it amounts to

$A_1 \mid A_2 \ldots\ldots \mid A_n = 1$ if and only if

$$\prod_{i=1}^{n} A_i = 0 \quad \text{for } n = 1,$$ that is the case of a single input the gate acts as an inverter.

2.2. NAND/NOR CONVERSION : TRANSFORM METHOD[4,5]

The transform method makes the logic design of logic circuits with the NAND or NOR blocks as easy and as rapid as designing circuits of AND, OR, and NOT blocks. Essentially it allows us to work in the familiar and simpler Boolean algebra, and during the last step of drawing the block diagram to apply a simple set of transform rules that map these Boolean equation or diagrams into NAND or NOR diagrams. To achieve a sense of minimality preservation in the transform method, it requires that a few simple constraints be added to the design done in Boolean algebra.

Given a Boolean function in sum of products form it can be directly transformed from the form (involving AND, OR, NOT function) to a form in NAND or NOR functions only. For example,

$$f = abc + d\bar{e} \quad \text{can be put as}$$

$$f = (a|b|c)|(d|\bar{e})$$

This is obtained as shown below:

$$f = \overline{\overline{abc}} + \overline{\overline{d\bar{e}}} \qquad = abc + \overline{\overline{d\bar{e}}}$$

$$= \overline{\overline{(abc)}} + \overline{\overline{(d\bar{e})}} \qquad = \overline{(abc)}\ \overline{(d\bar{e})}$$

$$= (a|b|c)|(d|\bar{e})$$

Figures written in parenthesis denotes the serial number of references given at the end.

For inverting 'c' we can make use of NAND logic element, in order to avoid the use of NOT function.

Similarly any function in products of sum form can be directly put in terms of NOR symbols

$$f = (a+b+c) \quad (a+\bar{c})$$

$$= (a \downarrow b \downarrow c) \downarrow (a \downarrow \bar{c})$$

It is seen that putting a logic function in terms of NAND or NOR function is a straight job since a function in sum of products form can be put in terms of NAND symbols and product of sum form can be put in NOR symbols. Also a general relationship exists between sum of products and product of sum form. Hence examples of implementation have been given for only one kind of functional form.

In the transform method proposed by Maley G.A. and Earle John a set of basic rules have been given along with some transform tricks. In the first step under this method functional interpretation of block diagram is made. This means inverse transform. This gives an insight into the general rules and the circuit.

2.9. INVERSE TRANSFORM (9)

For simplicity NAND function is considered here

f = ab + cd is implemented with NAND blocks employing two levels of gating as shown in Fig. 2.1. (a)

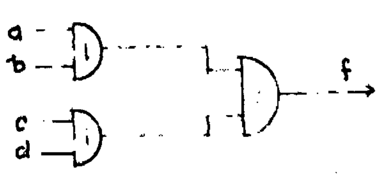


FIG 2a

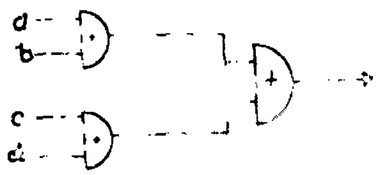


FIG 2 b

To implement the same function in English logic (i.e. AND, OR, NOT) same number of blocks are required as shown in Fig. 2(b). Comparing Fig. 2.1a and 2.1.(b) a set of rules can be formulated which when supplemented with few exceptions, is a generalisation. These rules form the basis of inverse transform technique for interpreting in algebraic form the circuit made up of either only NOR or NAND blocks.

Rules of Inverse Transform:

A list of inverse transform rules for the NAND circuit is given below:

Nand Networks:

1. Write e inside each even level gate and an 'O' inside each odd level gate ignoring all inverters (single input blocks). If a gate appears both as an even and odd level, leave it blank and all gates into it blank as well.

2. Transform all O gates to OR's transform all e gates to AND's . Transform all blank gates split in two as an AND (writing an e in it) driving the odd level loads and as an OR putting an O in it driving the even level loads . Alternatively treat the blank gates as though the output drove only odd levels, transform to AND's and insert an inverter in the output driving even levels.

3. Complement all variables entering OR's or O blocks. Any inverters to AND's remain the same.

NOR networks:

1. Same as mentioned for NAND .

2. Transform all e gates to OR's and all O gates to AND's Transform all blank gates split in two as an OR (putting an e in it) driving the odd level loads, and an AND (putting O in it) driving the even level loads. Alternatively treat the blank gates as though the output drove only odd levels, transform to OR's and

insert an inverter in the output driving even levels.

3. Complement all variables entering AND's or O blocks any inverters to OR's (o levels) remains the same.

The transform rules (from Boolean notation to NAND or NOR)

1. NAND:

Factor the Boolean equation to a form where the output is an OR (or and or etc) Try to get complemented variables on odd levels uncomplemented on even levels.

NOR: Factor to the Boolean equation to a form where the output is an AND (and or and etc) Try to get complemented variables on odd levels, uncomplemented on even levels.

2. Lay out the gating from the equation exactly as though they were in AND's and OR's instead of NAND's and NOR's except that both NAND and NOR variables coming in at odd levels of gating should be complemented

2.4. TRANSFORM TRICKS:

The following list of tricks will supplement the transform rules.

1. Partial multiplication:

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

This is useful for separating complemented variables to the odd levels to be complemented in the function, from the uncomplemented variables to the even levels where they will not be complemented in the function.

2. Adding or Multiplying a Constant $(A+B)(C+D)+0$

This puts an equation in the wrong form for the function into a correct form for the NAND with the output C.

3. Associativity: $D(\bar{A}+\bar{B}+C) = D\left[(\bar{A}+\bar{B})+C\right]$

This separates complemented variables (A,B) from the uncomplemented C ready ready for partial multiplication 1 to separate them to the levels that will eliminate inverters.

4. Doubling: $\overline{\overline{\overline{ABD}}} = \overline{ABD}$ (in NANDS)

Letting the complement of a signal be represented by the bundle of wires which are inputs to that block is

as $\bar{B}$ is redundant all complemented literals now on odd level.

b. Redundant terms may make factoring possible:

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

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

Redundancy allows the possibility of some gates being shared between even and odd levels replacing two gates.

An example of implementing a function with the NAND logic is illustrated below

$$(A+C)(A+B)(A+\bar{D})(\bar{B}+\bar{C}+D)$$

$$\text{Factoring} = (A+\bar{B}CD)(\bar{B}+\bar{C}+D)$$

In the second term the variables B, C are complemented D is not. It would be desirable to separate the complemented ones from the uncomplemented and put them on odd and even levels of gating respectively. We can do this by associativity followed by partial multiplication.

$$(A+\bar{B}CD)\left[(\bar{B}+\bar{C})+D\right] = (A+\bar{B}CD)(\bar{B}+\bar{C})+(A+\bar{B}CD)D$$

The $\bar{B}+\bar{C}$ term is odd as the complements get dropped but we still have the complement in $\bar{B}CD$. This can be gotten from the o on existing term however

called bundling. It eliminates gates.

5. <u>Distributive Law : $AB+\bar{C}\bar{D} = (AB+\bar{C})(AB+\bar{D})$</u>

This factoring rule (and its dual) can serve to separate complemented variables to all levels, and also to factor out common terms leaving correct operation form.

$$AB+CD\bar{D}+C\bar{D} = \underline{(AB+CD)}(AB+\bar{D})+(AB+CD)(AV+\bar{D})$$

(The underlined terms are common and are shared). It can be used to factor to a function form for which the solution is well known, like the exclusive or for example.

6. <u>Add Redundant terms, literals</u>

a. Redundant literals may make gates identical

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

Adding redundant $\bar{C}$ to the first term, and redundant $\bar{A}$ to the second term makes the gates identical and can be shared.

Redundant literals may eliminate gates:

$$E(C\bar{D}+G) = E(E\bar{C}+G+D) = E(E+G+D)$$

Ed redundant in eliminates the $E\bar{C}$ gate. Redundant literals may reduce complemented literals to all levels

$$E(\bar{D}D+\bar{F}) = E(\bar{D}E+\bar{F}+\bar{E}) = E(\bar{D}+\bar{F}+\bar{E})$$

If $\bar{B}$ is redundant all complemented literals now on odd level.

b. Redundant terms may make factoring possible:

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

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

Redundancy allows the possibility of some gates being shared between even and odd levels replacing two gates.

An example of implementing a function with the NAND logic is illustrated below

$$(A+C)(A+B)(A+\bar{D})(\bar{B}+\bar{C}+D)$$

Factoring $= (A+BC\bar{D})(\bar{B}+\bar{C}+D)$

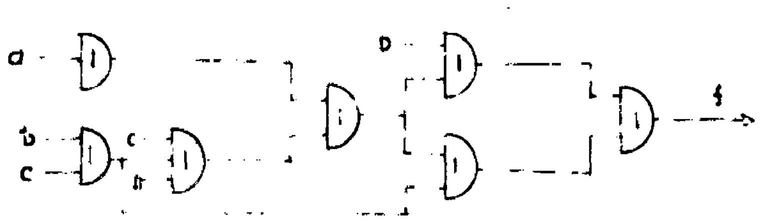
In the second term the variables B, C are complemented D is not. It would be desirable to separate the complemented ones from the uncomplemented and put them on odd and even levels of gating respectively. We can do this by associativity followed by partial multiplication.

$$(A+BC\bar{D})\left[(\bar{B}+\bar{C})+D\right] = (A+BC\bar{D})(\bar{B}+\bar{C})+(A+BC\bar{D})D$$

The $\bar{B}+\bar{C}$ term is odd so the complements get dropped but we still have the complement in $BC\bar{D}$. This can be gotten from the D on existing term however

$$\left[A+(\bar{B}+\bar{C})CD\right](\bar{B}+\bar{C})+\left[A+(\bar{B}+\bar{C})CD\right]D$$

This is implemented in Fig. below :



$f = (a+c)(c+d)(a+\bar{b})(\bar{b}+\bar{c}+d)$

The transform method merely gives a comparatively simpler ways of implementation of NOR and NAND logics and cases of minimality preservation exists to a certain extent. But we cannot say that the resultant circuits are absolutely minimal. Even if by chance we obtain a minimum circuit there is no way to recognise that the circuit is minimal.

2.5. DESIGNING WITH KARNAUGH MAP[5]

This method is proposed by Maley GA and Earle John.

For functions of only a few variables the map method is developed, and it makes use of considerable

retaining these advantageous, and uses the full logic power of NOR or NAND. Instead of Karnaugh map, here Veitch Karnaugh Map proposed by M.W. Dixon[6] is used to review this method.

Analysis of map method is dealt here for only NAND function to one. The method makes use of Veitch Karnaugh map in association with inhibition principle.

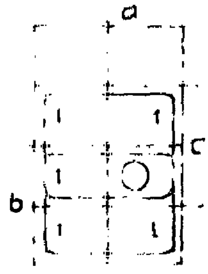
Principle of Inhibition (NAND) Function)

The previous method of reduction of the Boolean function into convenient form is carried out algebraically. Now a Graphical techniques are used in this map method and in most of the cases they may be found to be advantageous.

For example let us consider the function

$B\bar{C} + \bar{A}C + \bar{B}C$

The hatched one represents ABC



Now Inspecting the V-K map, we require the loop B AND NOT the circled loop ABC, or loop C AND NOT the circled loop ABC.

$B(\overline{ABC}) + C(\overline{ABC})$

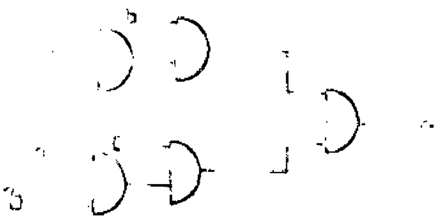
$= B(\bar{A}+\bar{B}+\bar{C}) + C(\bar{A}+\bar{B}+\bar{C})$

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

By simplifying ordinarily excluding the checked loop we get $B\bar{C} + \bar{A}C + \bar{B}C$

factoring this we get $B\bar{C} + C(\bar{A}+\bar{B})$

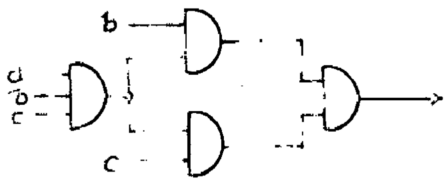
which requires five gates.



Whereas after including the checked loop and applying principle of inhibition on V-K map we get

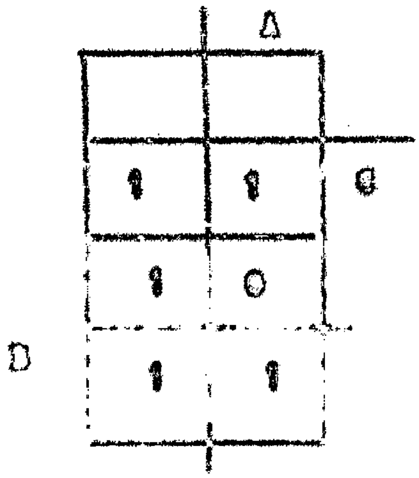
$B(\overline{ABC}) + C(\overline{ABC})$

which requires only four gates as shown



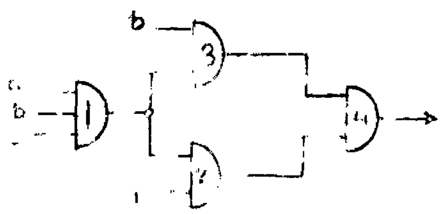
Loops on the V-K map representing the input literals to NAND blocks that get inverted on even levels of gating

act as ones. Loops representing an odd level gates act as zeros. A loop of zeros (gate on odd level) will inhibit any area of intersection with a loop of ones(even level gates), if the zero loop goes into the input of the one loop gate.



Take loop ABC (gate 1 ABC), take loop C and inhibit with loop ABC. This yields (gate 2 — $\overline{C.1}$). Take loop D and inhibit it with loop ABC. This yields gate 3 - $\overline{D.1}$

Take the inhibited loops, 2, 3 and inhibit from the unitary loop (the loop of the whole map) with them. This will yield all zeros of the map. This is the output gate 4. This argument will yield the circuit below:



which is same as previous one.

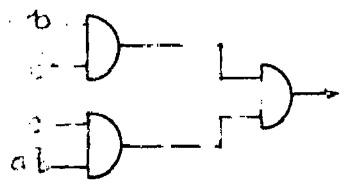
2.6. Here is suggested a new Approach.

After simplifying on V-K map, let the expression be

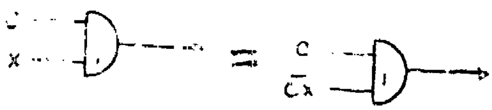
$$bc + c\bar{a} + c\bar{b}$$

$$bc + c(\bar{a} + \bar{b})$$

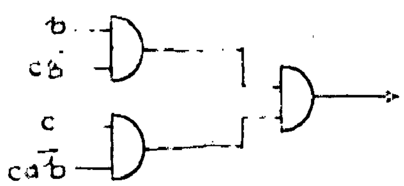
Now implement the same directly by Nand Blocks



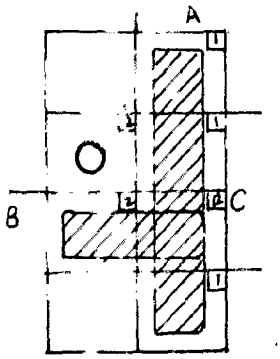
Output is not changed by making the following change as it only amounts multiplying c with c.

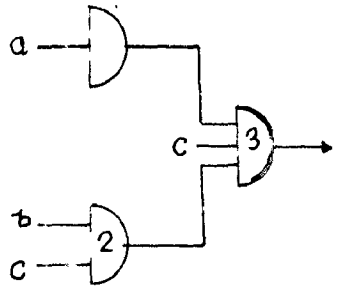


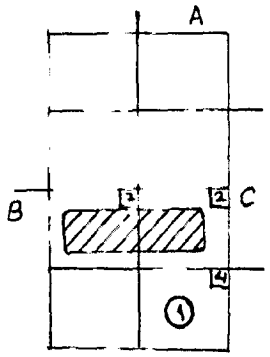
Using this principle, i.e. by including redundant inputs in the block diagram itself we can achieve reduction. Applying this principle and figure above can be modified to the following change.

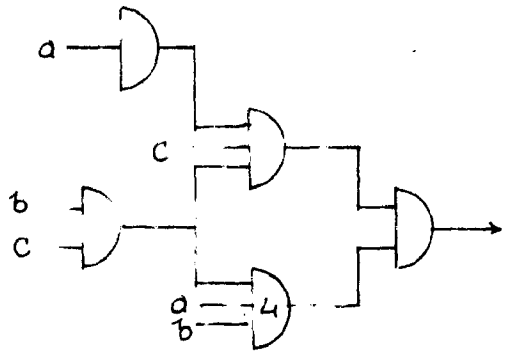


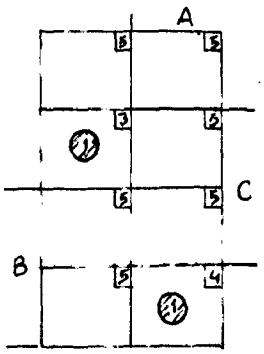
$f = \bar{a}\bar{b}c + a\bar{b}c$











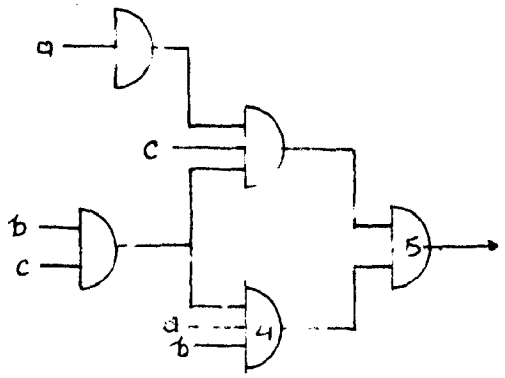


FIG-

Simplification on V-K map by using principle of inhibition, is a graphical technique. The technique is cumbersome when applied to more than four variables case as the map itself becomes complicated for five variable case. For each step we have to use a different map. There is no assurance of minimality in this method also.

CHAPTER III

IMPLEMENTATION OF NOR and NAND LOGICS NEW APPROACH SUGGESTED BY GODVANI - A REVIEW

IMPLEMENTATION OF NOR AND NAND LOGIC - NEW APPROACH (AS STATED BY COWAN[4]) A REVIEW.

Cowan in his M.S. thesis 1959 - An Implementation of NOR and NAND logic has suggested a new approach on implementation.

Method:

Given a logic function (involving AND, OR, NOT) the first step is to bring to the form involving NAND or NOR function only as per requirements. This is done with no regard for incorporating minimality.

In the second step relation which are given below are used to bring the function to the requisite form.

No block as far as possible be used specially for purpose of inversion and this serves as a principle and guide in the implementation problem using this technique.

$$f = a\bar{b}c + ab\bar{c}$$

The following relations in case of NAND logic are made use of in the above method.

1. $A|B|C \neq A|[B|C] = [(A|B)]|C$

2. $(A|B|C)|(A|B|\bar{C}) = (A|B)|$

3. $(A|B|\bar{C})|(A|B|\bar{B}) = [A|B(C|B)]$

5. $(A|B|C)|(A|B|D) = [A|[(B|C)|(B|D)]]|$

6. $(A|B)|(A|B|C) = (A|B)|$

7. $A|C = A|(\bar{C}|A)$

8. $(A|B)|(A|\bar{B}) = A$

9. $A|(\bar{A}|B) = \bar{A}$

How the method is applied in the implementation of function by NAND logic is described below

$$\text{Let } f = \bar{a}\bar{b}c + ab\bar{c}$$

$$= (\bar{a}|\bar{b}|c)|(a|b|\bar{c})$$

This is true for any function in sum of products form. Above function needs six blocks.

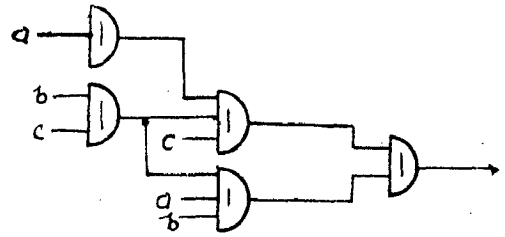
No. of blocks 6.

Applying one of the relations available

$$f = (\bar{A}|\bar{B}|C) \mid (A|B|\bar{C})$$

$$= \left[A \mid (B|C) \mid C\right] \mid \left[A|B|(B|C)\right]$$

This needs five blocks.



For NOR function

Let the function given be

$$f = (A + C + \bar{D})\ (\bar{A} + B + \bar{C})(\bar{A} + B + \bar{D})$$

$$= (A \downarrow C \downarrow \bar{D}) \downarrow (\bar{A} \downarrow B \downarrow \bar{C}) \downarrow (\bar{A} \downarrow B \downarrow \bar{D})$$

Applying relations

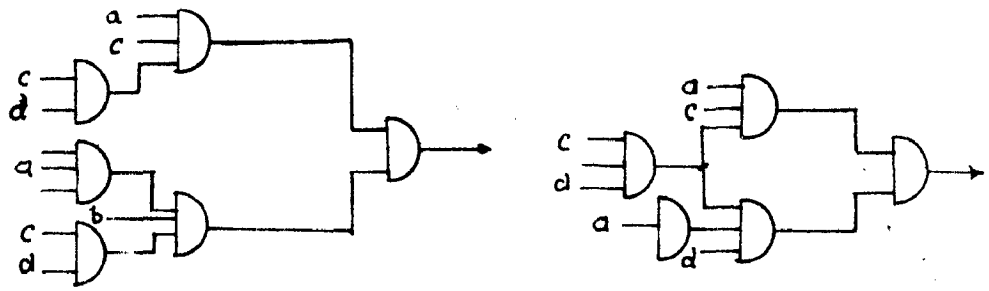
$$f = (A+C+\bar{D})+(\bar{A}+B+\bar{C})+(\bar{A}+B+\bar{D})$$

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

$$= \Big\{\big[(A+C+\bar{D})+(A+C+\bar{C})\big]+\big[(\bar{A}+B+\bar{D})+(\bar{A}+B+\bar{C})\big]+\Big\}$$

$$= \big[A+C+(C+D)\big]+\big[\bar{A}+B+(C+D)\big]$$

$$= \big[A+C+(C+D)\big]+\big[\bar{A}+B+(C+D)\big]$$



which requires five blocks only.

Another example which uses this method is illustrated here

$$f = A\bar{B} + \bar{A}C + A\bar{C} + \bar{B}C$$

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

This requires 8 blocks.

Applying relations given previously

$$S = \left[A|(D|C)\right] | \left[C|(A|D)\right]$$

$$= \left[A|(A|D|C)\right] | \left[C|(A|D|C)\right]$$

This requires four blocks.

This method uses only the relations in all for both the NOR and NAND implementation. Capability of selection of appropriate relation at right place is expected with practice.

The method suggested above by Gilman is nothing but expressing as many as possible in the function in NAND form. The relations given are only the properties of NAND or NOR logics. This method does not however take into consideration minimality at all. The implementation results in the use of comparatively large number of blocks.

The method is cumbersome if it is applied on four or more variable functions is not systematic. The method is not systematic in that we do not know how to apply the relations appropriately.

CHAPTER IV

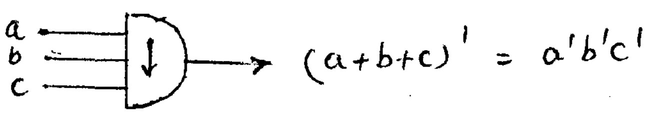
REVIEW OF LEO HELLERMAN'S WORK

is that there is no way to recognize that is certain configuration is minimal ; we could only rely on the fact that no one has done better. The Chapter is devoted to the review of Hellerman's work.

4.1. OUTLINE OF THE METHOD

The method is exhaustive in that looks at every possible combinational network of NOR invert blocks and find out the desired minimum for each three variable logic function. Each of the three variable function can be implemented by seven blocks or fewer. There are 2^{42} 4×10^{12} possible combinational networks with seven or fewer blocks. By programming on the digital computer the problem is that all these circuits are tested and minimum circuits are arrived at . A list of 256 logic functions of three variable is input data for the programme.

1. The procedure starts by considering, only possible network with three inputs and one NOR block.



This is the minimum circuit for performing the NOR function of three variables. This function may now be struck off from the list of functions to be found.

each input to each block (we say that block i feeds block j if the output i is an input of j) The points labelled (ij) are points of possible interconnection of vertical and horizontal lines, with each point ij we associate a value (ij) which may be 0 or 1.

(ij) = 1 if ij is a connection point

= 0 if ij is not a connection point.

In this each circuit in the universal network corresponds to a single matrix of the form

$$
\begin{bmatrix}
(11) & (12) & (13) & x & (15) \cdots\cdots & 1(9+n) \\
(21) & (22) & (23) & (24) & (25) \cdots\cdots & 2(9+n) \\
\\
(n1) & (n2) & (n3) & (n4) & (n5) & x
\end{bmatrix}
$$

As we are dealing with circuits of single output without loss of generality we assume block n as the out-put block in Fig. 1.

4.2. ELIMINATION OF SEQUENTIAL CIRCUITS

In the combinational circuits we are using feedback is not allowed, and keeping this in view we can further simplify the network in Fig. 1 to the Fig. 2 below:

2. Next we consider all network with two blocks in some order. For each circuit we evaluate the function performed and check against the list. If it is on the list we cut off the function found if it is not on the list we discard the circuit.

3. After evaluating all the two block networks we go to the three block networks and so on.

In this procedure all the accepted circuits are minimum.

To understand the programming for this particular problem in the digital computer we shall now deal with "Universal" network, showing possible interconnections between inputs and blocks as shown.

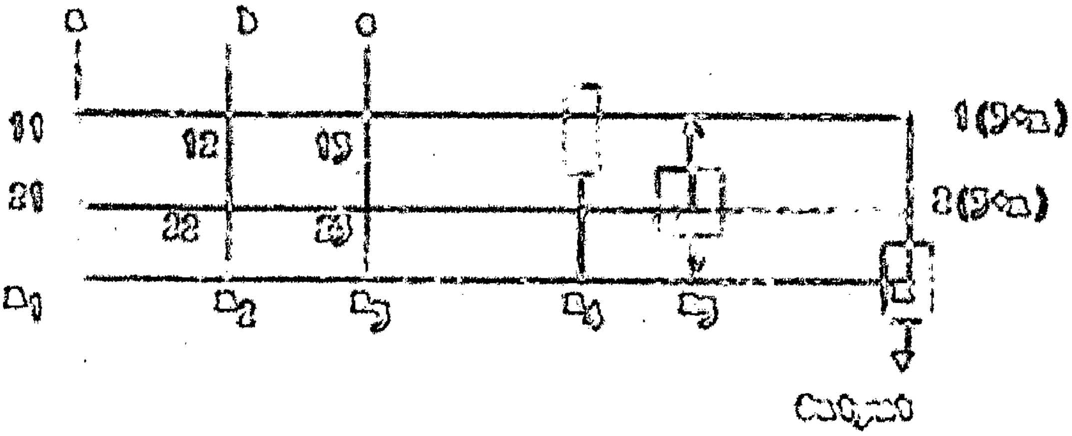


Fig. 1

A arrow is indicated the direction of outputs from the blocks. In the universal network shown above we can see that it is possible that any block can feed each of the other block and that there is a possible path from

each input to each block (we say that block i feeds block j if the output i is an input of j) The points labelled (ij) are points of possible interconnection of vertical and horizontal lines, with each point ij we associate a value (ij) which may be 0 or 1.

(ij) = 1 if ij is a connection point
      = 0 if ij is not a connection point.

In this each circuit in the universal network corresponds to a single matrix of the form

(11)	(12)	(13)	x	(15)	......	1(5+n)
(21)	(22)	(23)	(24)	(25)	....	2(5+n)
(n-1)	(n-2)	(n-3)	(n-4)	(n 5)		x

As we are dealing with circuits of single output without loss of generality we assume block n as the out-block in Fig. 1.

4.2. ELIMINATION OF SEQUENTIAL CIRCUITS

In the combinational circuits we are using feed-back is not allowed, and keeping this in view we can further simplify the network in Fig. 1 to the Fig. 2 below:

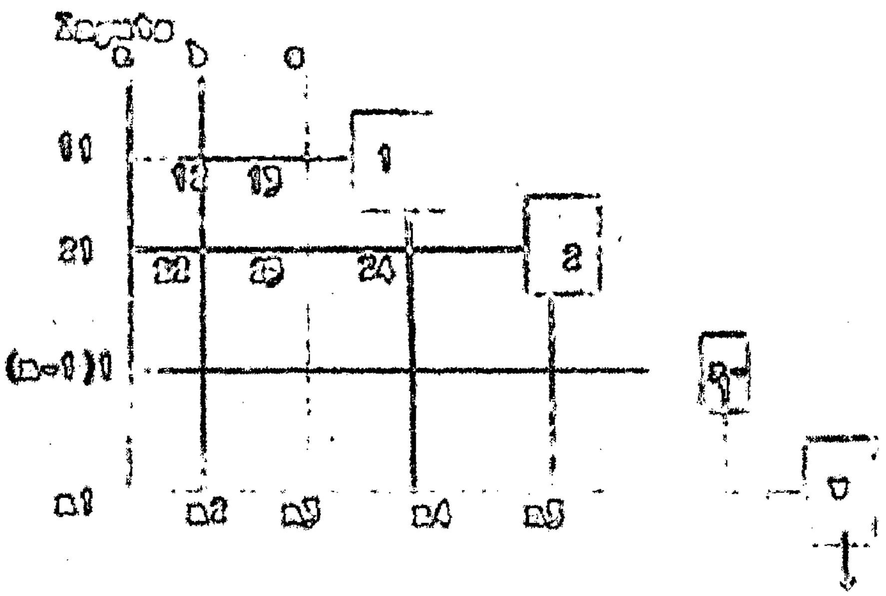


Fig. 2.

In combinational network at least one block is fed by no other one. We may choose block 1 of the network as that which is not fed by any other block. The remaining blocks form a subnetwork which is also combinational. Now block 2 can be chosen as that which is not fed by no other block. Continuing the same we get the network in Fig. 2. The connection in such a network may be specified by the combinational connection matrix.

$$\begin{bmatrix} (11) & (12) & (13) & & \\ (21) & (22) & (23) & (24) & \\ \vdots & & & & \\ (n1) & (n2) & (n3) & (n4) & \cdots\ n(n+2) \end{bmatrix}$$

For $n = 3$, the matrix has 48 elements each of which may take the value 0 or 1 . The elements of the first three columns specify the connection of inputs to the network the remaining elements specify the interconnection of the blocks.

4.3. DESIGNATION OF EQUIVALENT FUNCTION CLASSES

The 256 logical functions of three variables can be partitioned into 80 equivalent classes. If one function can be obtained by the from the other by the permutation of input variables, they are said to be equivalent and belong to the same class. Similarly the networks can be partitioned into equivalent classes with respect to input line permutation. Since implementation of any member of a function class serves to implement all members of the same class, it is necessary to evaluate only one network from each equivalent class.

jth column of vector in the connection vector in the connection matrix is written as a row vector

(1j) (2j) (3j) ......(nj) where j may be 1,2,3, and ij) 1 or 0, depending on whether the network has a connection at the point (ij) or not. The correspondence of network classes to function classes is many to one. So even though there are only 80 classes of functions, there are many more network classes.

6

For networks with seven logic blocks the input matrix contains 21 elements giving rise to 2^{21} input networks. Eliminating the networks that do not satisfy the relation $D_1 \leq D_2 \leq D_3$ where D_j is notation for a row vector which is a binary number, the input networks evaluated become approximately $1/6 \times 2^{21}$.

To further simplify the magnitude of the problem and render the programme easy, we can eliminate some impossible cases.

1. The first block must be fed by atleast one variable for it is not fed by any other block. This means at least one of the points (1,1) (1,2) (1,3) of the network in Fig. 2 must be a connection.

Since $D_1 \leq D_2 \leq D_3$ we can let point 1,3 be connected for all networks. The programme will set up and evaluate no circuit for which (1,3) = 0

2. Block (n-1) must always feed block block n as seen from Fig. 2. This means that point n(n+2) must always be connected. The programme will set up and evaluate no circuit for which n(n+2) = 0.

3. Each row of the connection matrix must contain atleast one 1 and each column contains atleast one 1. So

Since each block except the last should serve as an input to some subsequent block. A fan limit of 3 is imposed, and hence connection matrix has another restriction that each row and each column should contain no more than three 1's.

By applying these constraints in the programme, the programme is simplified and Hellerman arrived at 80 minimal NOR/NAND circuits which are given in Appendix II. In the following Chapter an alternative approach is indicated.

The circuits obtained by Hellerman are minimal according to the definition of minimality indicated in the beginning of this Chapter. The method is exhaustive and it has tested all the possibilities . This is the only listed circuits which are known to be minimal.

CHAPTER V

AN ALTERNATIVE APPROACH TO A MINIMAL NETWORK.

AN ALTERNATIVE APPROACH TO MINIMAL DESIGN

Dr. Leo Hellerman of IBM corporation has given a complete catalogue of minimal NOR and NAND circuits of 3 variables functions, assuming complemented literals are not available. Daley and Earle have shown that list of 256 functions of 3 variables reduces to 78 circuit configurations which has been discussed in the previous Chapter. Hellerman's catalogue gives the minimal NOR and NAND implementation of these 78 functions.

In this work Leo Hellerman's catalogue of functions have been studied and an attempt is made to systematise the minimal implementation by certain rules.

Given any logic we can first implement it by AND- OR blocks, using inverters to get the complemented literals. If we replace the AND, OR blocks by NAND blocks the output function will be the same. So one method of implementation of a logic function by NAND logic , is to first implement by AND- OR blocks and then replacing them by NAND blocks.

For some functions this type of implementation may be minimal. This has been found by the study of the circuits given by Hellerman. Since we know these

circuits are minimal.

51 out of the 80 circuits could be designed by this method. Now how to identify each function out of the 80 functions is another problem. So some rules have been suggested below to ascertain which function is to be implemented in which way to achieve minimality in implementation. The definition of minimality has already been explained in the previous Chapter.

The first type of implementation is to implement the logic function AND OR blocks, using inverters to obtain complemented literals, and then to replace AND, OR by NAND blocks.

1. Expressions which consist of only one term can be implemented by method 1.

Function	Circuit No.
abc	2
a'b'c	12
a'bc	6
a'b'c'	26

2. Functions of the form x + y + z where x, y, z can be complemented or uncomplemented literals

Circuit No.	Function
1	a'+b'+c
4	a'+b'+c
10	a'+b +c
17	a+b+c

3. Function which contain no complemented literal at all

Circuit No.	Function.
8	ab+c
9	(a+b)c
18	ab+ac+abc

4. Functions in Which each of literals complemented or uncomplemented occure only once.

Circuit No.	Function
3	a'+bc
7	a'+b'c
14	a'+b'c'
15	c+a'b
30	c+a'b'

9. Functions in which we cannot take any complemented literal as common (in which ring sum of uncomplemented literals is not implied).

31	a'b' ⊕ bc	54 X	a'bc ⊕ b'c'
32	a' ⊕ bc ⊕ b'c'	56	a'b'c' ⊕ bc
33	abc ⊕ a'b'	57	abc ⊕ a'b'c'
34a	ab'c' ⊕ bc	61	c⊕ab⊕a'b'
35	abc⊕a'b'c		

Method II

Method. Implementation by ring sum logic. Before this method has been discussed the properties of ring sum are given in Appendix I.

21	a'b'c ⊕ bc'	45	a'(b'⊕c')⊕cbc
22a	a'b ⊕ cb'c	47a	c(a'⊕b')⊕a'cb'
23	a'⊕ab'⊕abc'	48	(b'⊕ab')a'
30	(ab⊕a'b')c	53	a'(b'c' ⊕bc)
40a	a'b⊕c ⊕cb'	57	a'b⊕a'⊕cb'c'
42	a'b⊕cb' ⊕c	60	ab⊕a⊕a'b'c'
43a	a'c ⊕cb'c'	24	(a'b⊕cb')c
44a	ab'c⊕a'bc	63a	a'(b⊕c')⊕cb'c'

65. a'b'c ⊕ a'bc'⊕cbc

280 a'b cb'c

57 a'(b'⊕c')⊕cbc

49 a'b ⊕cb'⊕c

73 c'b'c' ⊕a'bc⊕cb'c

Method 3.

Express the function of the form $x'+y'$ as $\overline{xy}$ and implement by method 1.

5	$c(a'+b')$	35x	$bc+b'(a'+c')$
11.	$(a'+b)c$	57	$a'(b'+c')+abc$
13	$a'(b'+c')$	59	$a'b'+c(a'+b')$
20	$a'(b'+c)$	62x	$a'b+ab'+a'c'$
64x	$a'(b+c')+ab'c$		

Method 4

If the complement of the given function has no complemented literal present in it then implement the complement of the function by method 1 and add an inverter. circuits (27)

2	$a'b'+a'c'+b'c'$	30	$a'b'+a'c'+b'c'+abc$
		29x	$a'c+bc'$

The list of circuits given by Hellerman have been given in Appendix II.

In all total 53 out of 80 NAND circuits have been grouped under the following category.

1. AND OR implementation, all blocks being replaced by NAND blocks.

2 Ring Sum implementation of the function after expressing the function in Ring sum form or Ring sum with zero addition etc.

3. Expressing function of the type $x'+y$ as xy

Still there is a possibility to find a systematic method to design the remaining 27 circuits also. The methods suggested for the design of 53 circuits shown in the following pages, reasonably hold good but there is scope for making them exhaustive.

In implementing NAND logics we can make use of these two following theorems

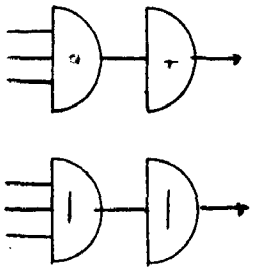
(i) $x'y + xy' = (x+y)(x'+y')$

This may result in useful simplification.
Basically the implementation of NAND logic seems to be first implementing by AND OR circuit and then replacing them by NAND blocks. It is necessary that the number of blocks in the AND OR circuit itself are minimum.

In this method 2 ring sum circuits have been used . Certain properties of ring sum are investigated and they have been given in Appendix III.

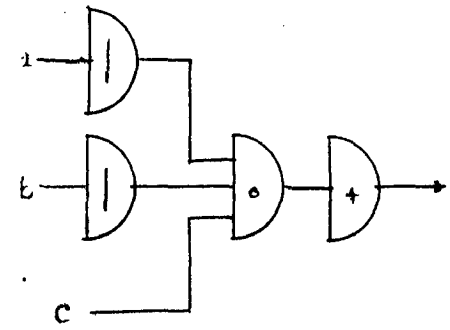
Symbols nf used in the preceeding Chapters have been given in the beginning .

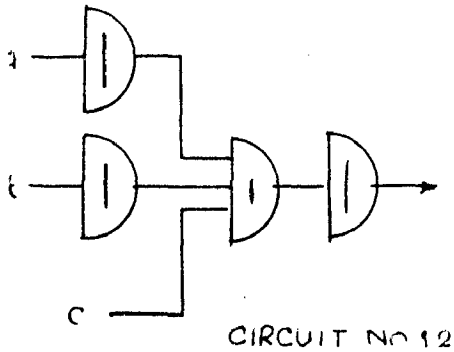
EXPRESSION



CIRCUIT NO 2

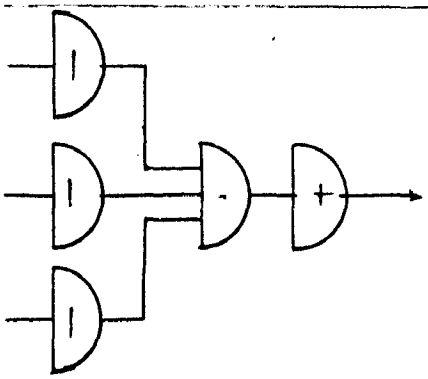
EXPRESSION a' b + c'

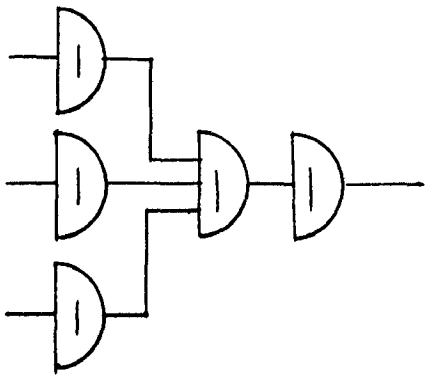




CIRCUIT NO 12

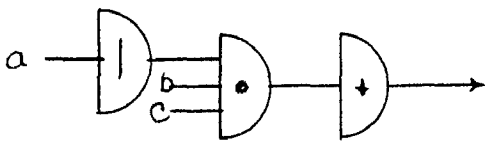
XPRESSION a' b' c'

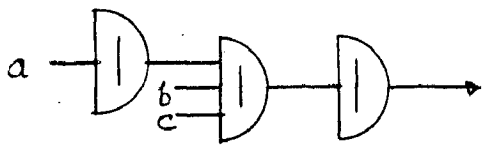




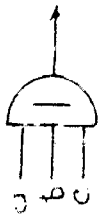
CIRCUIT -26

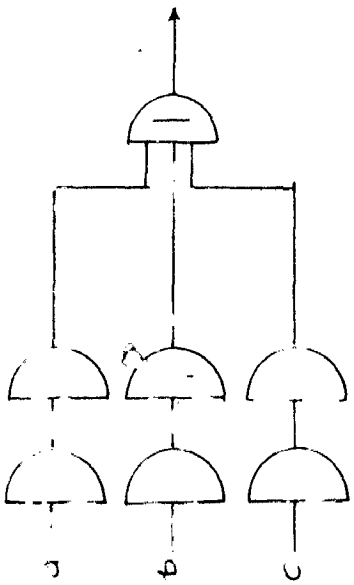
EXPRESSION a' b c

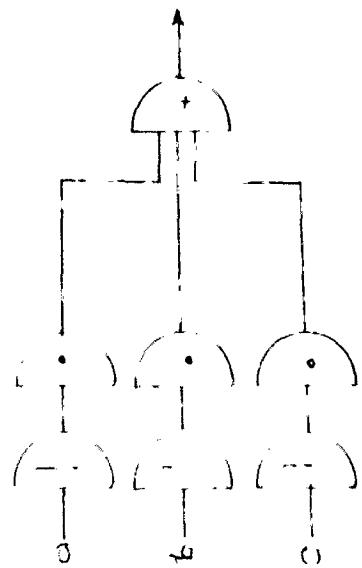




CIRCUIT -6

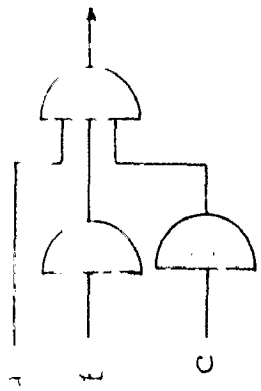


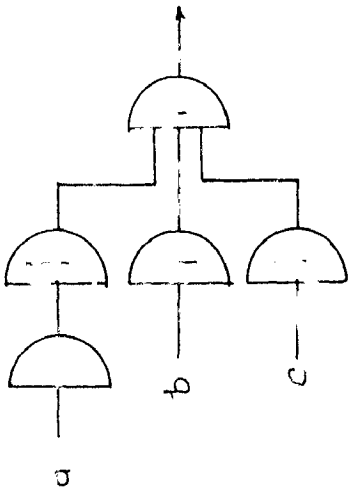


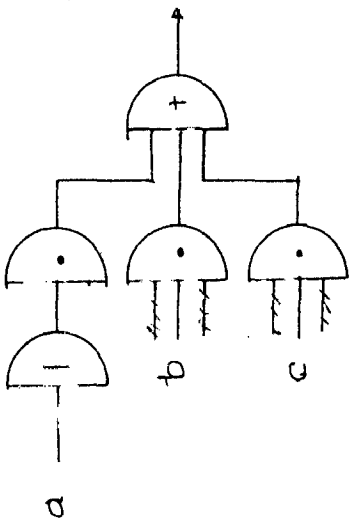


CIRCUIT 1

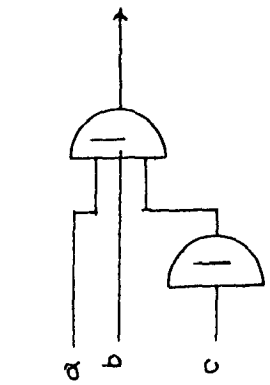
EXPRESSION a'+b+c

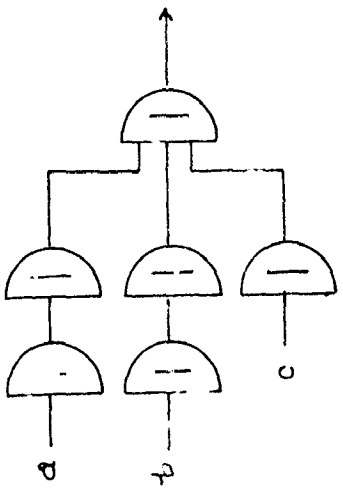




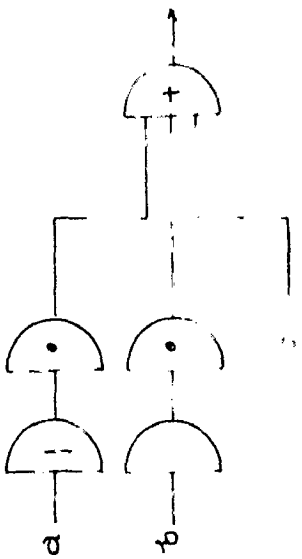


CIRCUIT 10

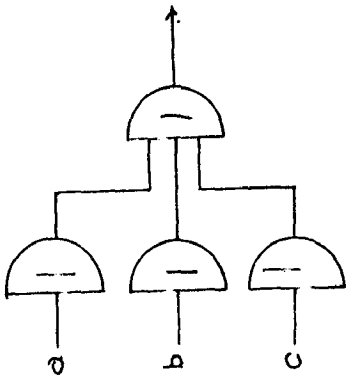




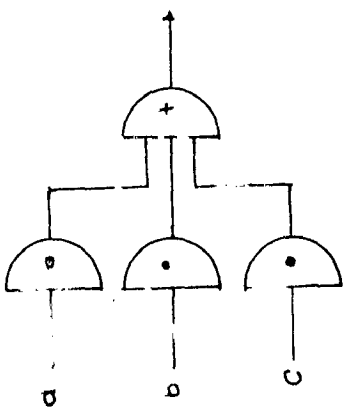
CIRCUIT 4



EXPRESSION a + b + c



CIRCUIT - 17



III

EXPRESSION $ab + bc + ca$

CIRCUIT 18

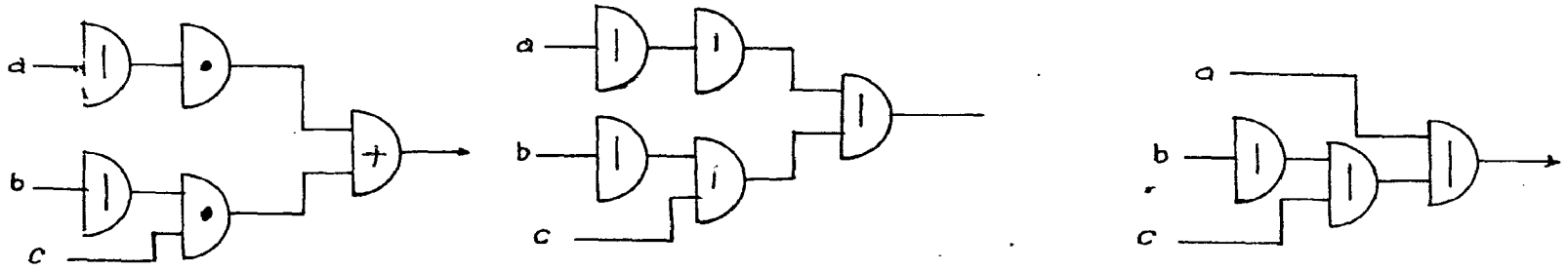
EXPRESSION $ab + c$

CIRCUIT – 8

EXPRESSION $(a + b)c$

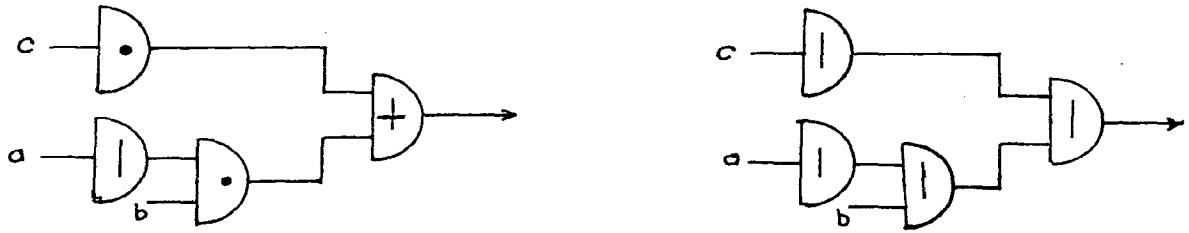
CIRCUIT - 9

EXPRESSION $a' + b'c$



CIRCUIT-7

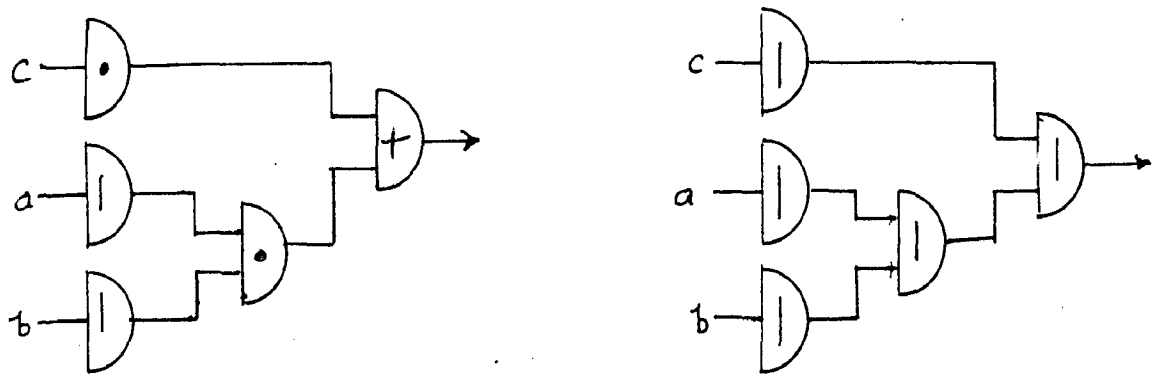
EXPRESSION $c + a'b$



CIRCUIT – 15

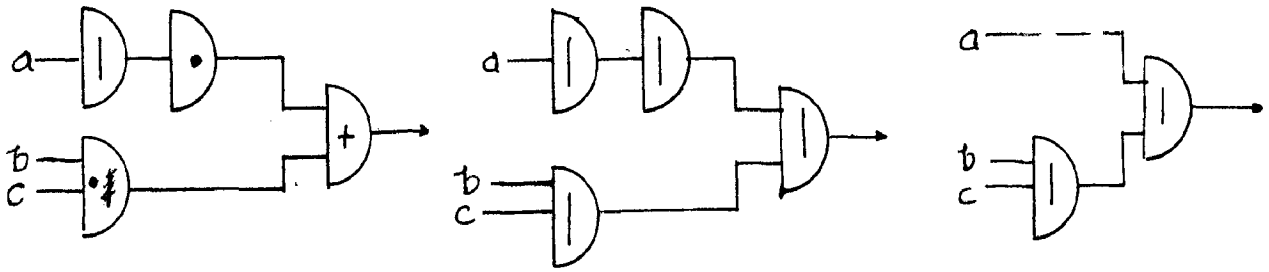
IV

EXPRESSION c +a' b'



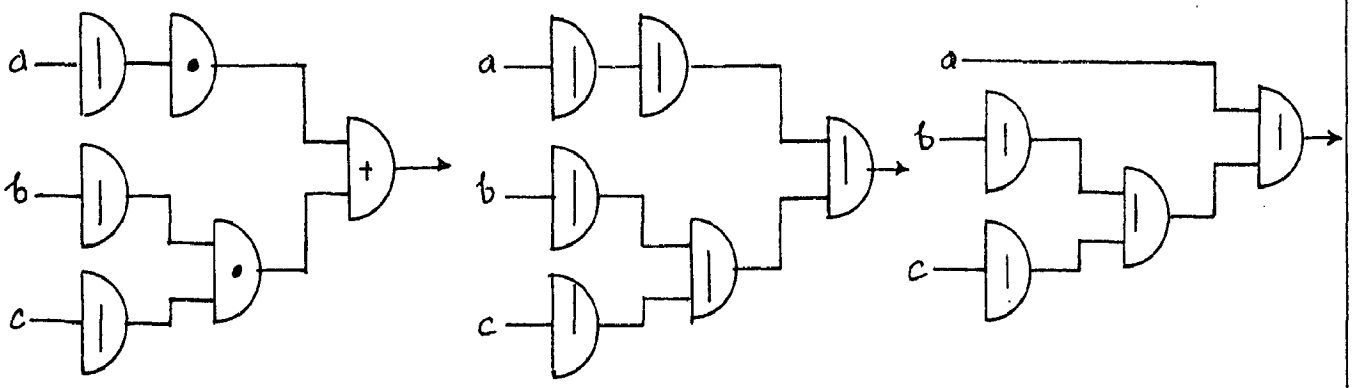
CIRCUIT 30

EXPRESSION a'+bc



CIRCUIT 3

EXPRESSION a' + b'#c'

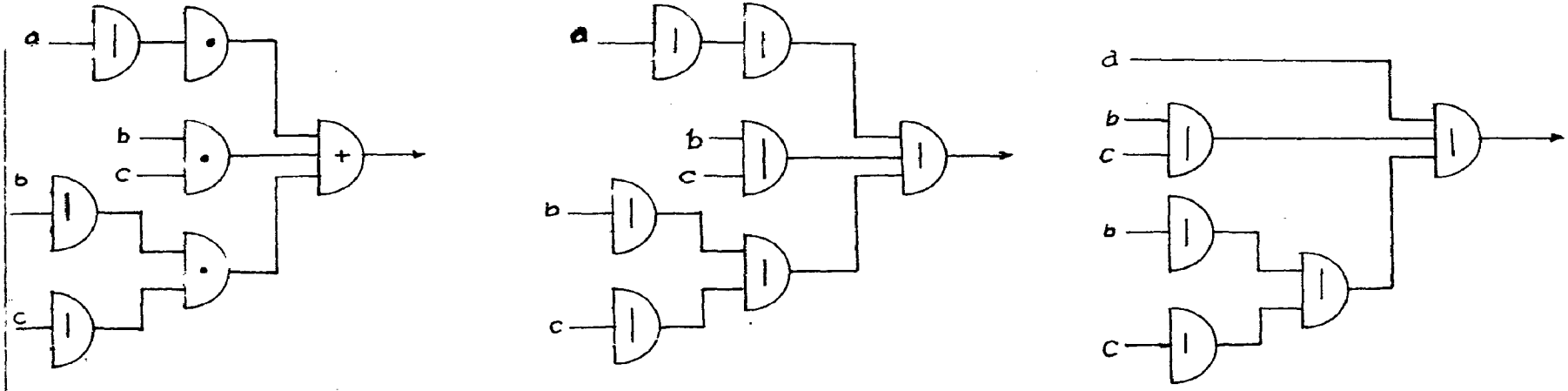


CIRCUIT 14

EXPRESSION V $a'b' + bc$

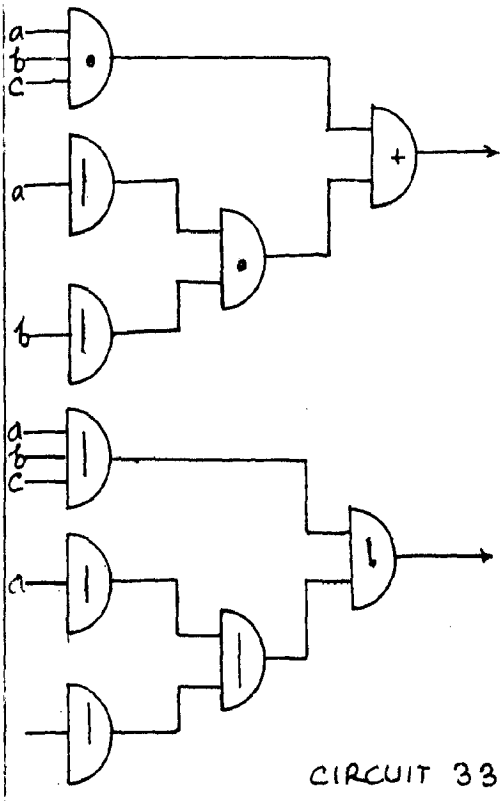
CIRCUIT-31

EXPRESSION $a' + bc + b'c'$



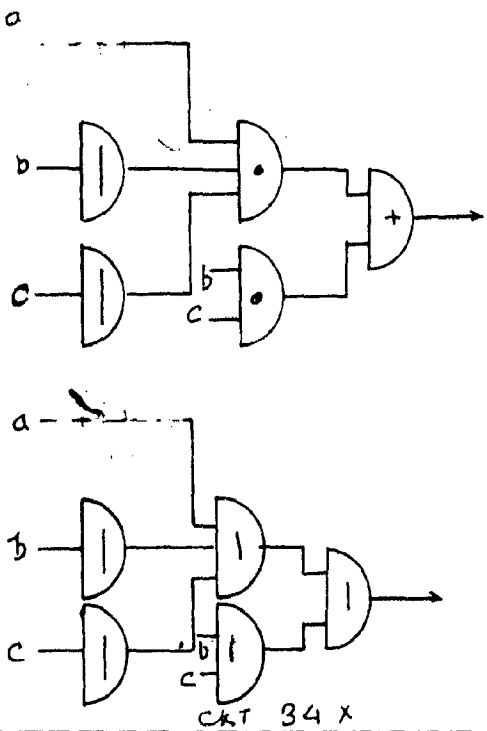
CIRCUIT NO- 32

EXPRESSION abc + a' b'



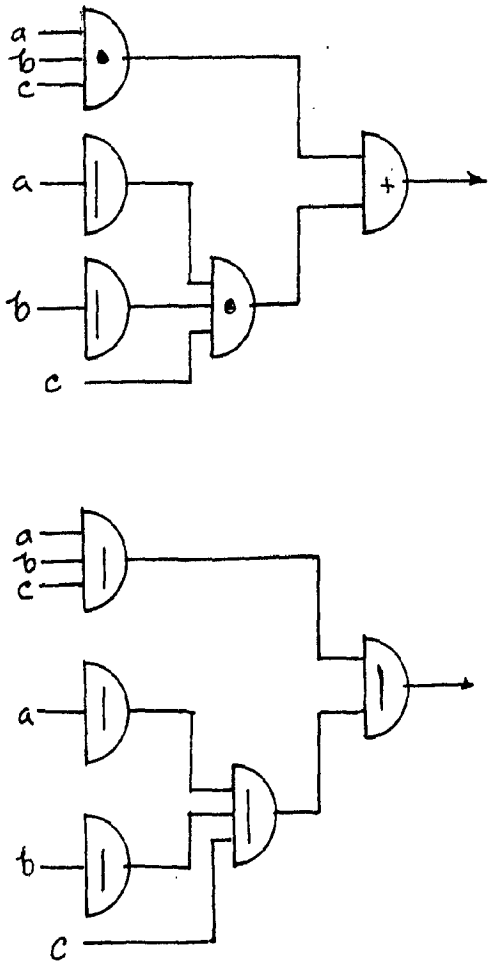
CIRCUIT 33

EXPRESSION a b'c' + bc



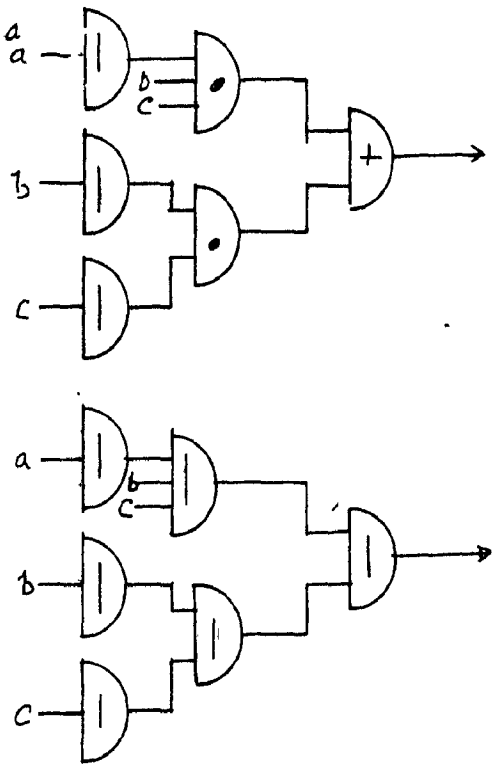
CKT 34 X

EXPRESSION a bc + a' b'c



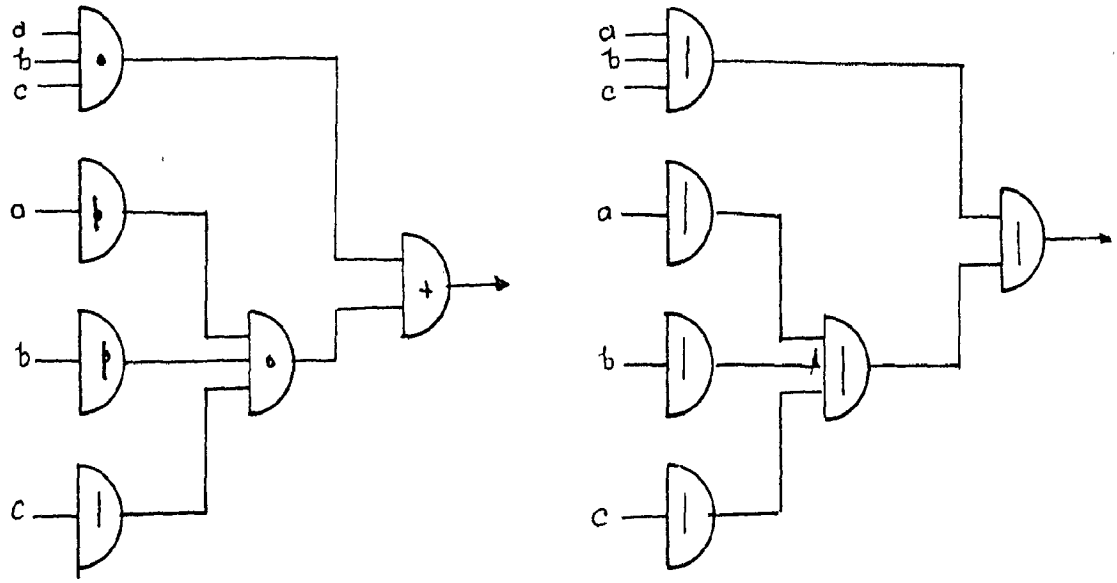
CIRCUIT 35

EXPRESSION a' bc + b'c'



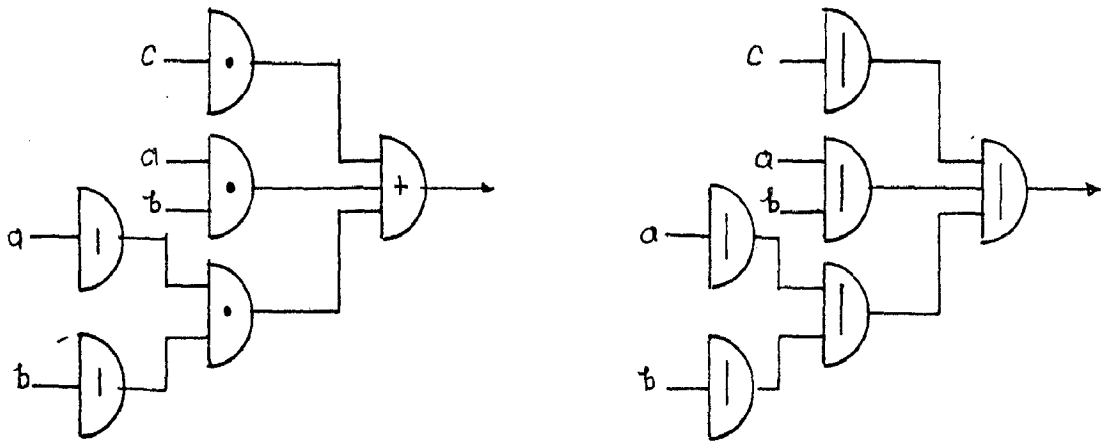
CIRCUIT - 56

EXPRESSION $a^{*}b\,c + a'b'c'$



Circuit No. 57

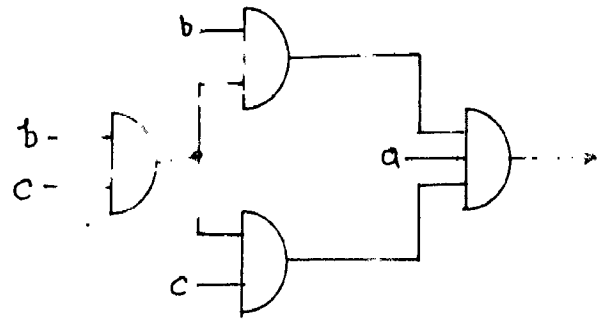
EXPRESSION $c + a\,b + a'\,b'$



Circuit No 61

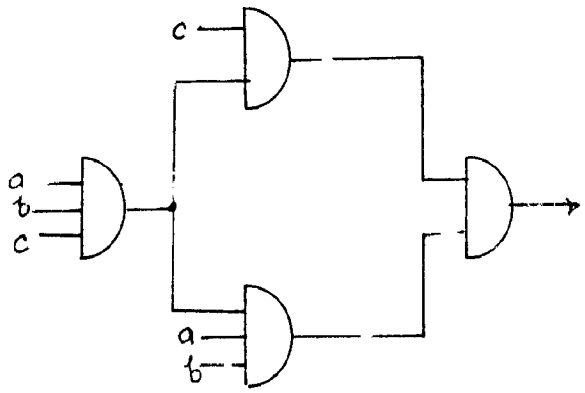
RING SUM

EXPRESSION $a' + b'c + bc'$
$= a' + b \oplus c$



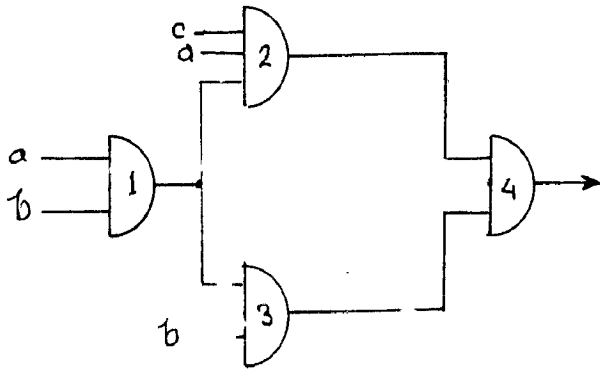
CIRCUIT No 21

EXPRESSION $a'c + bc + abc' = c(a'+b') + abc'$
$= c\overline{ab} + abc' = c(a \oplus b)' + abc'$
$= c \oplus ab$



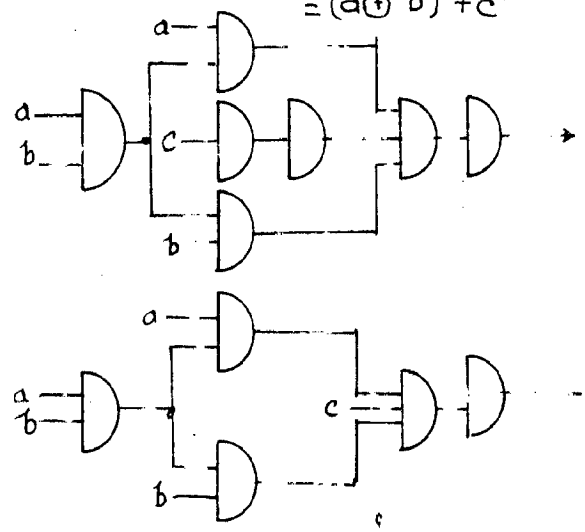
CIRCUIT 25

EXPRESSION $a'b + ab'c$



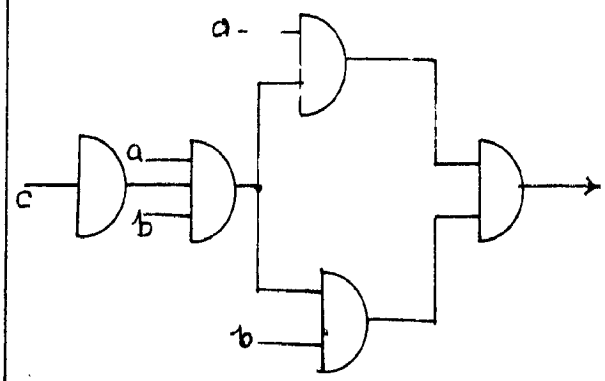
CIRCUIT No 22x

EXPRESSION $(ab + a'b')c = \overline{(a \oplus b)}c$
$= \overline{(a \oplus b)} + c'$



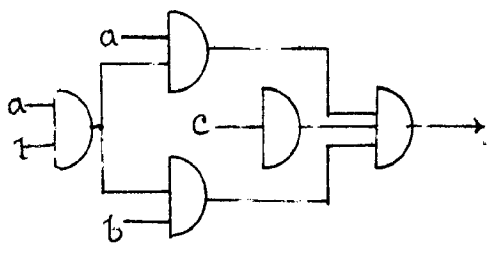
CIRCUIT 38

EXPRESSION $ab' + ac + a'b$
$= a \oplus b + ac$



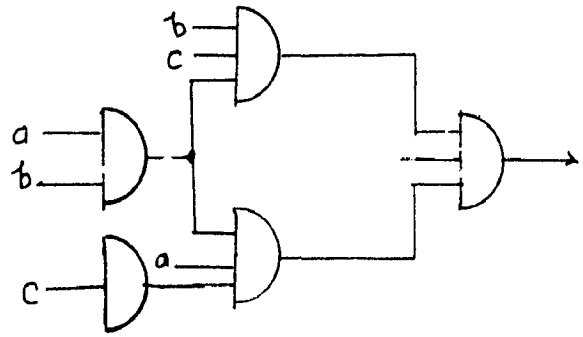
CIRCUIT 40x

EXPRESSION $a'b + ab' + c$
$= a \oplus b + c$



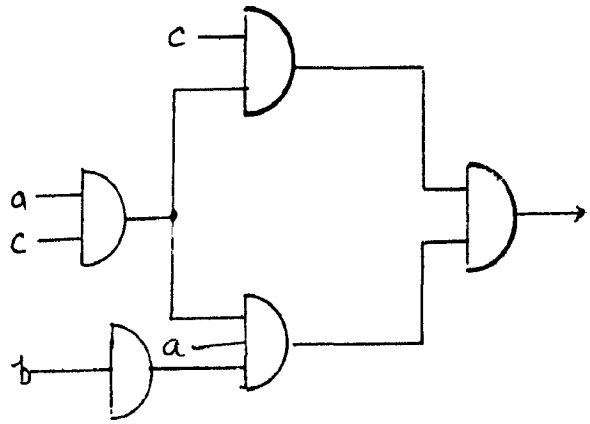
CIRCUIT 42

EXPRESSION $a'b'c + a'bc$



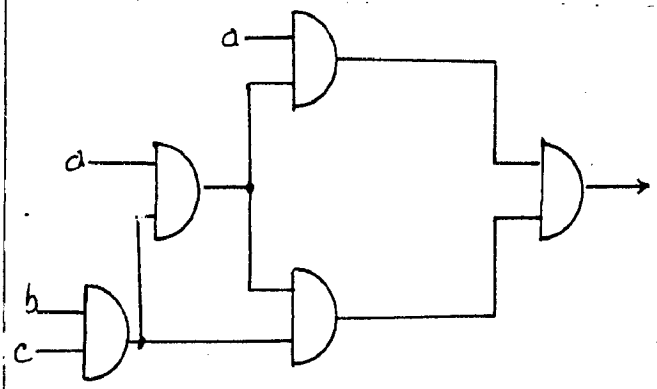
CIRCUIT NO 44X

EXPRESSION $a'c + a\,b'c'$



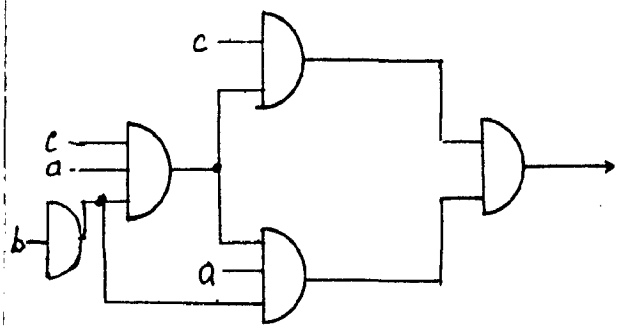
CIRCUIT - 43 X

EXPRESSION $a'(b'+c') + a\,bc$
$= a'\overline{bc} + abc = a \oplus \overline{bc}$



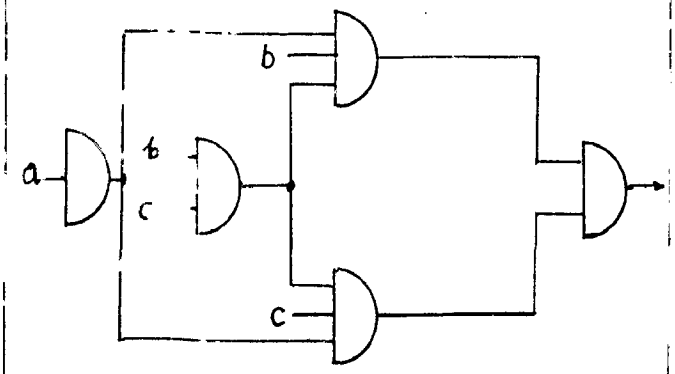
CIRCUIT 46

EXPRESSION $c(a'+b) + c'a\,b'$
$= c(\overline{a\,b}) + c'\,a\,b'$
$= c \oplus a\,b'$



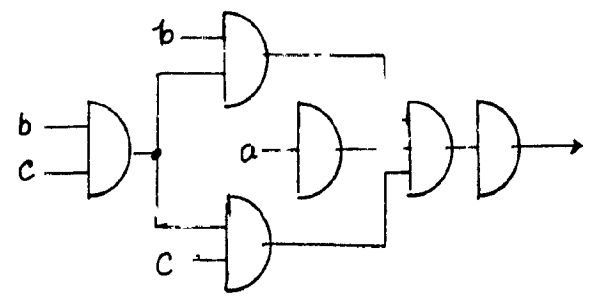
CIRCUIT 47 a

EXPRESSION $a'(b'c + bc')$
$= a'(b \oplus c)$



CKT NO 48

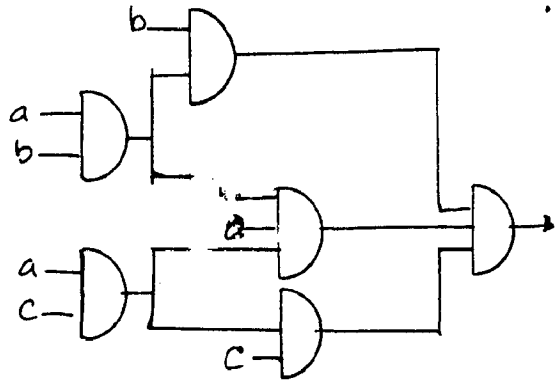
EXPRESSION $a'(b'c' + bc)$
$a'b \oplus c = \overline{a + (b \oplus c)}$



CKT NO 53

EXPRESSION $a'b + a'c + ab'c'$

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



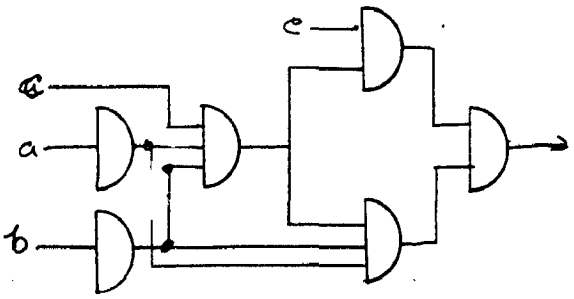
CIRCUIT 67

EXPRESSION - $ac + bc + a'b'c'$

$c(a+b) + c'a'b'$

$c\,\overline{a'b'} + c\overline{a'b'}$

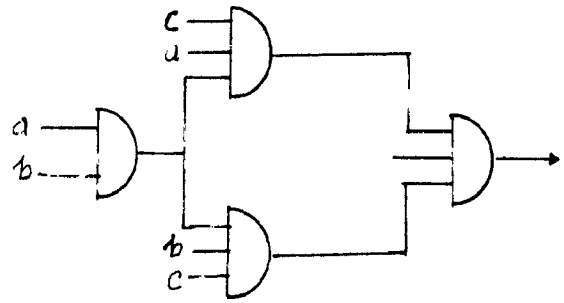
$c \oplus a'b'$



CIRCUIT 68

EXPRESSION $(a'b + ab')c$

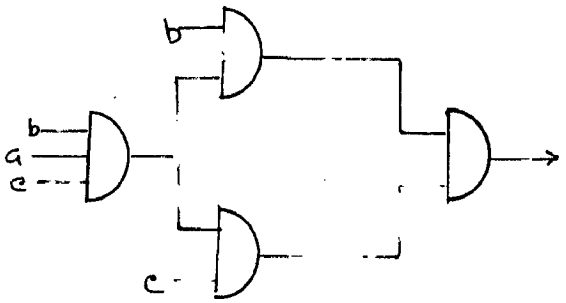
$(a \oplus b)c$



CIRCUIT 24

EXPRESSION $b'c + bc' + a'(b+c)$

$b + c + a'(b+c)$

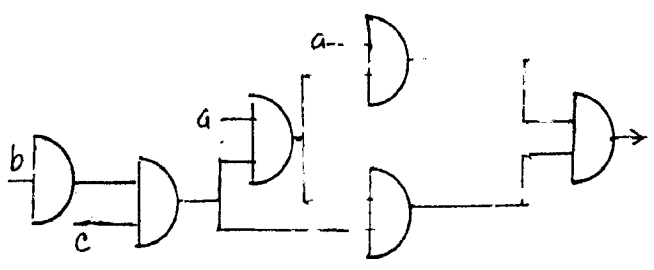


CIRCUIT 23

EXPRESSION $a'(b+c') + ab'c$

$= a'\overline{b'c} + ab'c$

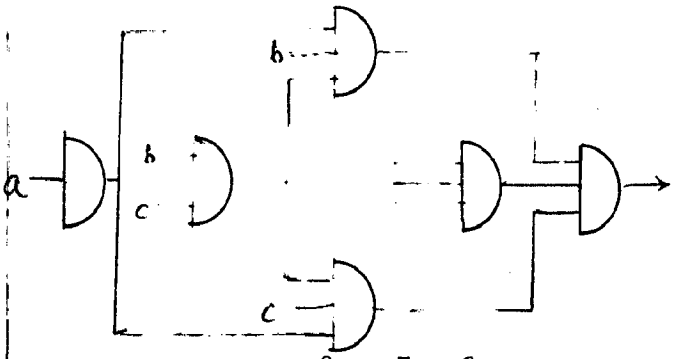
$= a \oplus b'c$



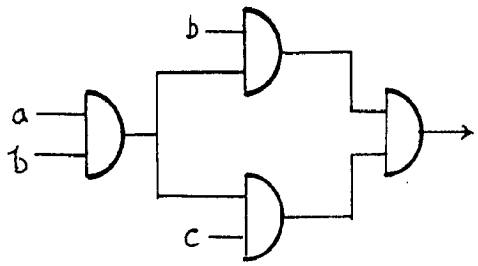
CIRCUIT C5

EXPRESSION $a'b'c + a'bc' + abc$

$a'(b \oplus c) + abc$



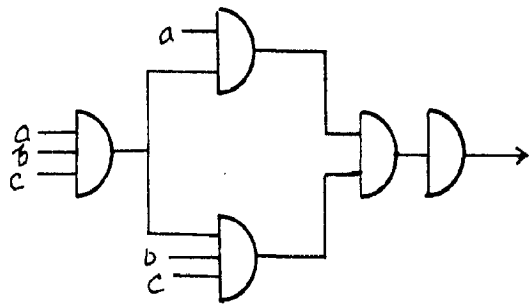
CIRCUIT - 66



EXPRESSION

$a'b + b'c$

CIRCUIT 20

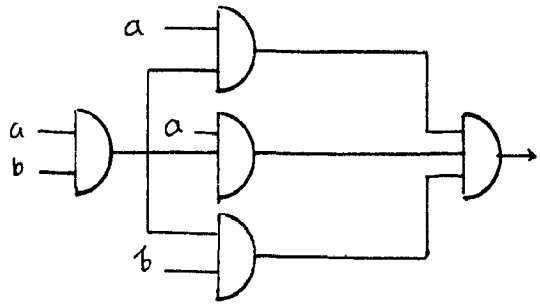


EXPRESSION

$a'(b'+c') + a\,b\,c$

$\overline{a \oplus bc}$

CIRCUIT 37



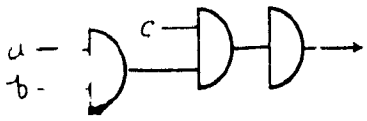
EXPRESSION

$a'b + ab' + c(a'+b')$

$= a'b + ab' + a'c$

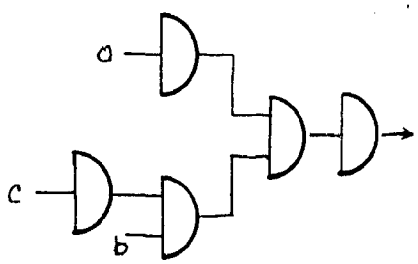
CIRCUIT 49

EXPRESSION $(a' + b')$
$= c\ \overline{a\,b}$



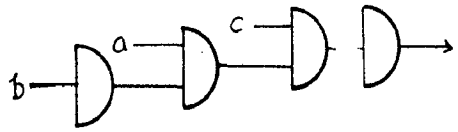
CIRCUIT 5

EXPRESSION $a'(b'+c)$
$= a'\ b\ \bar{c}'$



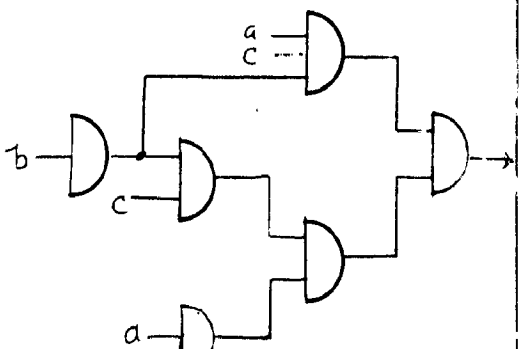
CIRCUIT NO 28

EXPRESSION $(a' + b)\,c$
$= \overline{a\,b'}\ c$



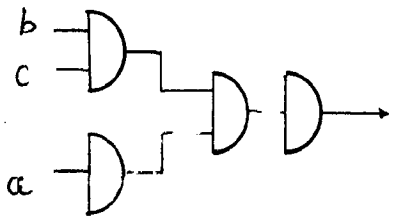
CIRCUIT 11

EXPRESSION $a'(b+c') + a\,b'c$
$= a'\overline{b'c} + a\,b'c$



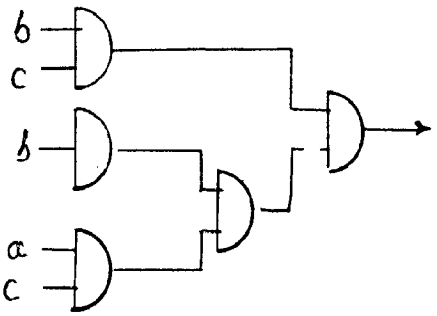
CIRCUIT NO 64X

EXPRESSION $a'(b'+c$
$= a'\ b\,c$

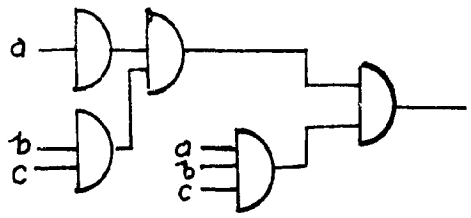


CIRCUIT 13

EXPRESSION
$bc + b'(a'+c')$
$= bc + b'\ \overline{a\,c}$



CIRCUIT 36X

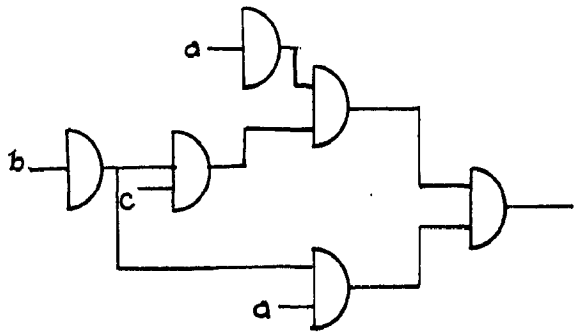


EXPRESSION

$a'(b'+c) + abc$

$a'\overline{bc} + abc$

CIRCUIT 37



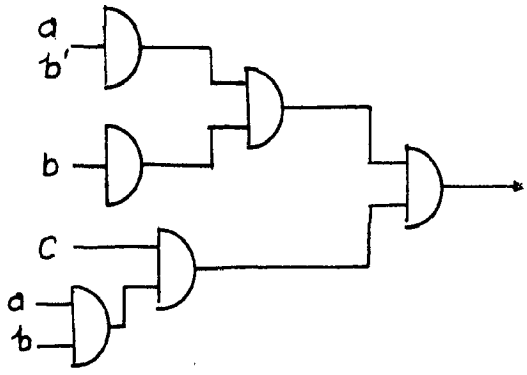
EXPRESSION

$a'b + ab' + a'c'$

$= a'(b+c') + ab'$

$= a'\overline{b'c} + ab'$

CIRCUIT 63X

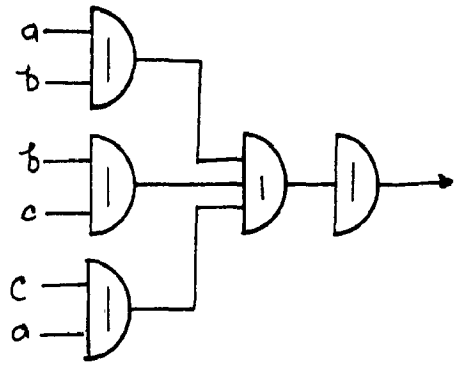


EXPRESSION

$a'b' + c(a'+b')$

$= a'b' + c\overline{(ab)}$

CIRCUIT 55

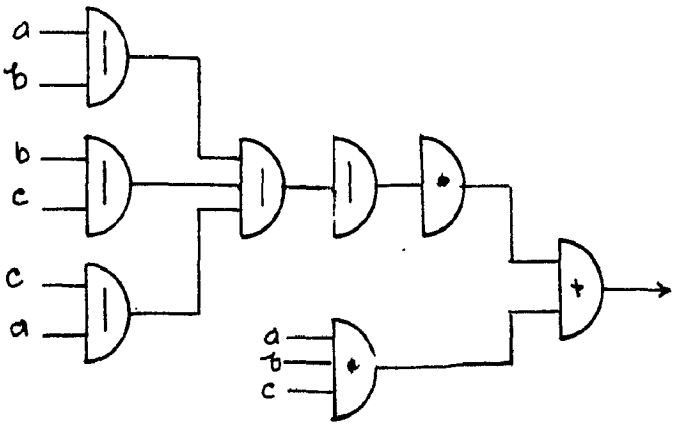


EXPRESSION

$f = a'b' + a'c' + b'c'$

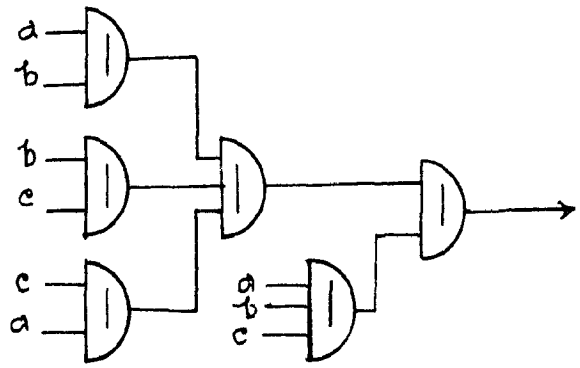
$\bar{f} = ab + bc + ca$

CIRCUIT - 27

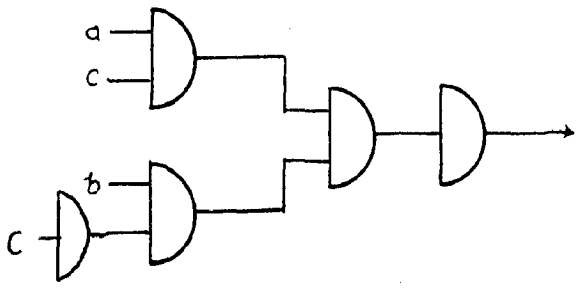


EXPRESSION

$a'b' + a'c' + b'c' + abc$



CIRCUIT - 54



EXPRESSION

$f = a'c + b'c'$

$f = ac + bc'$

CIRCUIT 29X

CONCLUSIONS

Group		Circuit	Property
I	a	2, 12, 6, 23	Expression consist of only term
	b	1, 4, 16, 17	These are of the form xy+z these xy+z can be complemented or uncomplemented.
	c	8, 9, 10	These do not contain any comple- mented literal.
	d	3, 7, 14, 15, 30	Each of the literals complemented or uncomplemented occur only once.
	e	31, 32, 33, 34a, 35, 36a, 36, 37, 61	We cannot join any complemented literal as common (since one of uncomplemented literals is not implied)
II		21, 22a, 29, 30, 40a 42, 43a, 44a, 45, 47a 48, 53, 67, 68, 24, 29 69a, 6., 20, 31, 40, 73.	The functions can be expressed in xy+z form with some additions or modifications.
III		5, 11, 13, 20, 64a, 36a 37, 35, 69a.	Functions of the form x̄ + ȳ are replaced by xy .

IV   27,28,29:   Complement of these functions contain less number of complemented literals.

The remaining 22 circuits, 16, 39, 51, 52, 72, 73,74,76,77,79,60,69, 70, 71, 59, 66, 62, 49, 19 did not come under any group, and it might be possible to find another method to design them also. The rules given in Group I (1,2,3) may be applicable in the case of four variable function also. Considerable labour is saved by these methods, which are easy and simple in application.

REFERENCES

1. Caldwell S.H. "Switching Circuits and Logical Design" John Wiley & Sons. Inc. 1958.

2. Phistor M. Jr. "Logical Design of Digital computers" John Wiley and Sons Inc. 1958.

3. Leo Hellerman "Catalogue of Three variable OR and AND Invert networks" IEE Trans. EC12 1963.

4. Arjun Godhwani "On Implementation of NOR and NAND Logic in Digital Computers." M.E. Thesis 1965 University of Roorkee

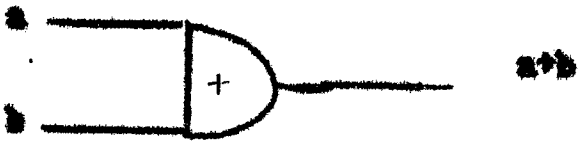
5. Maley G.A. & Earle John " Logic Design of Transistor Digital Computers " Prentice Hall Inc. 1965.

6. Biswas N.N., "Veitch Karnaugh Map" Letter to the Editor Control April 1965 P. 185.

A P P E N D I CES

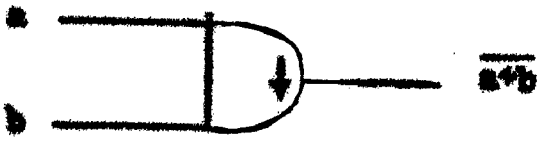
APPENDIX I

AND

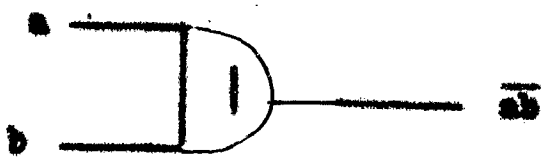


OR

Inverter (NOT)



NOR



NAND

⊕ Ring sum

↓ NOR

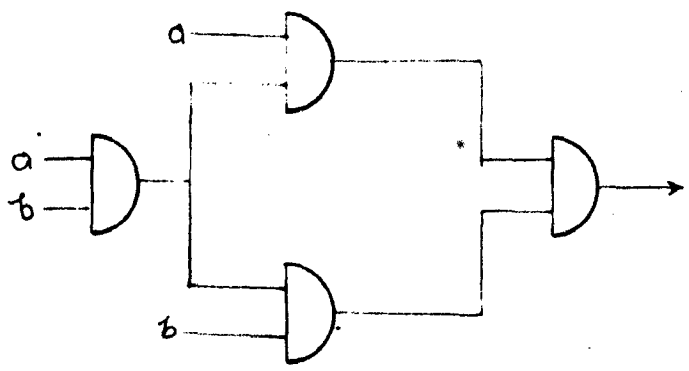
| NAND

APPENDIX II

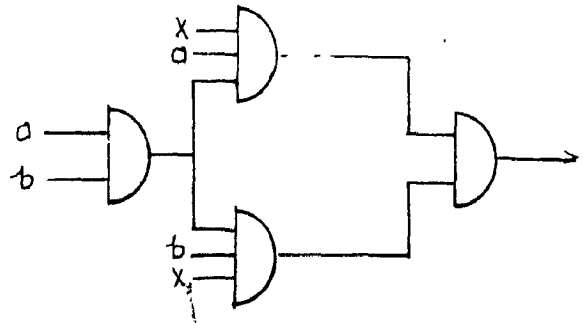
Some properties of Ring sum implementation circuits are given which are of use in the minimal implementation of NAND logics.

1. $a + b = ab' + a'b$
2. $\overline{a + b} = a + \bar{b} = a + b$
3. $\bar{a} + \bar{b} = a + b$
4. $(a\ b)\ c = ac + bc$

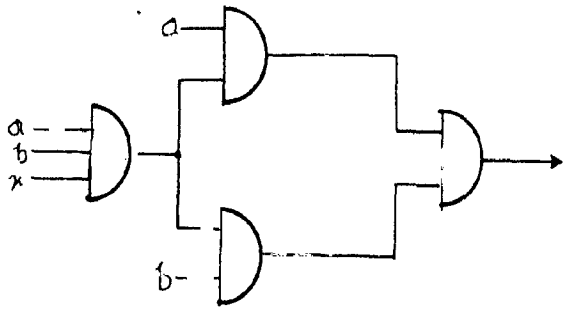
This is minimal implementation for $a'b + ab'$ Now by adding some inputs some blocks , how the output varies is illustrated in the figures given in the following pages.



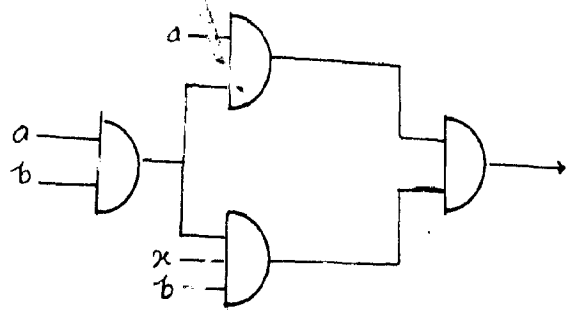
EXPRESSION $x(a \oplus b)$



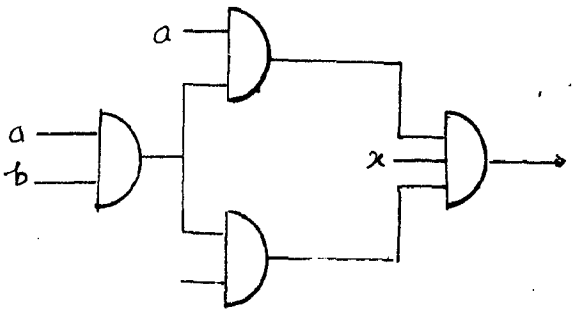
EXPRESSION $a \oplus b + \bar{x}(a+b)$



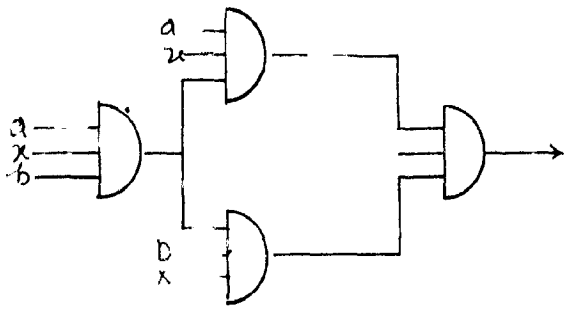
EXPRESSION $a'b + ab'x$



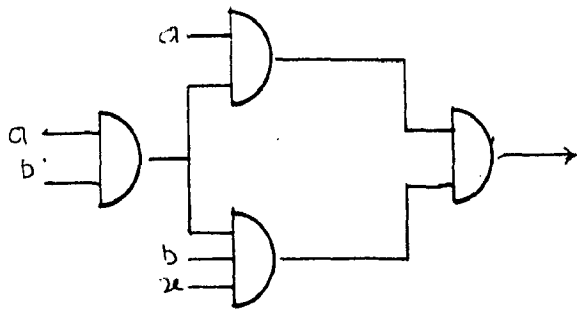
EXPRESSION $a \oplus b + \bar{x}$



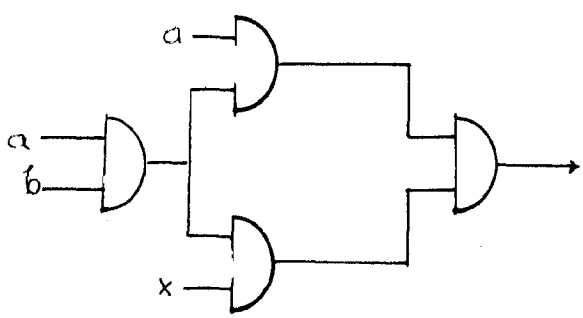
EXPRESSION $x(a \oplus b)$



EXPRESSION $x(a \oplus b)$

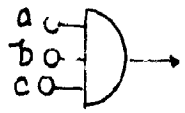


EXPRESSION $\bar{a}\,b'(x+a)$



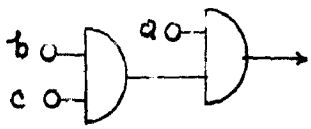
HELLERMAN'S CATALOGUE OF NOR and NAND CIRCUITS OF THREE VARIABLE FUNCTIONS

	NUMBER	EXPRESSION
NOR	1	$a'b'c'$
NAND	177	$a + b' + c'$



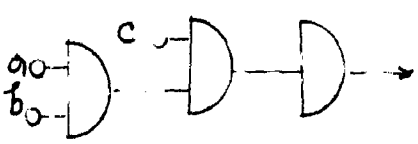
CIRCUIT NO 1

NOR	16	$a'(b+c)$
NAND	217	$a' + bc$



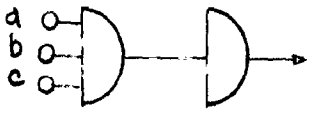
CIRCUIT NO 3

NOR	259	$c + a'b'$
NAND	52	$c(a'+b')$



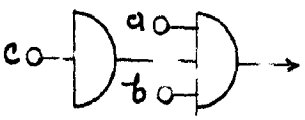
CIRCUIT N 5

	NUMBER	EXPRESSION
NOR	376	$a + b + c$
NAND	200	$a\ b\ c$



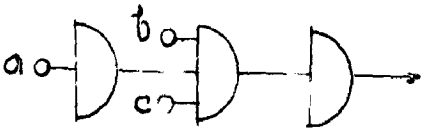
CIRCUIT NO 2

NOR	2	$a'b'c$
NAND	277	$a' + b' + c$



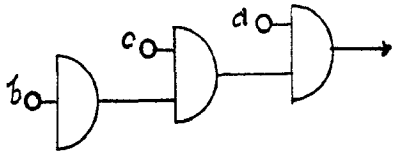
CIR CUIT NO 4

NOR	357	$a'(+b+c$
NAND	10	$a'bc$



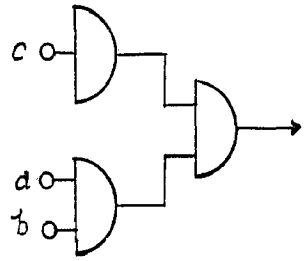
CIRCUIT NO 6

	NUMBER	EXPRESSION
NOR	19	$a'(b'+c)$
NAND	57	$a'+b'c$



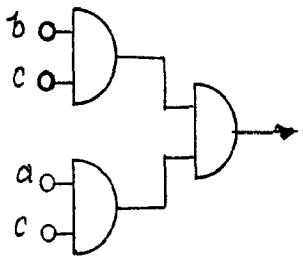
CIRCUIT NO. 7

	NUMBER	EXPRESSION
NOR	250	$(a+b)c$
NAND	352	$ab+c$



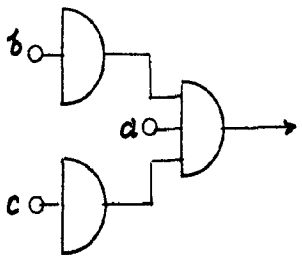
CIRCUIT NO 8

NOR	352	$ab+c$
NAND	250	$(a+b)c$



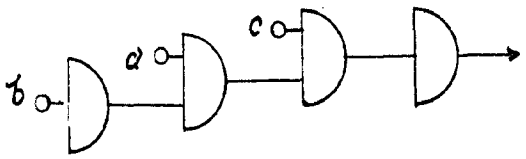
CIRCUIT NO. 9

NOR	10	$a'bc$
NAND	357	$a'+b+c$



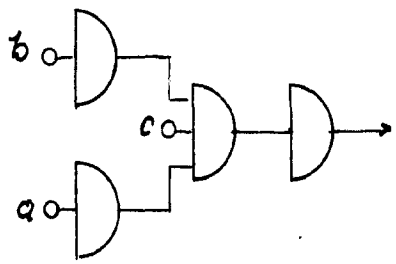
CIRCUIT NO. 10

NOR	256	$a'b+c$
NAND	212	$(a'+b)c$



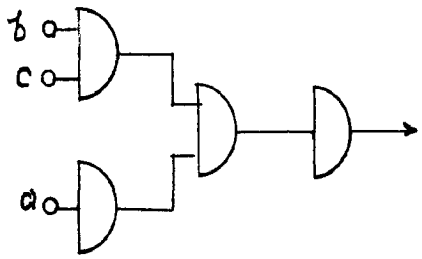
CIRCUIT NO 11

NOR	277	$a'+b'+c$
NAND	2	$a'b'c$



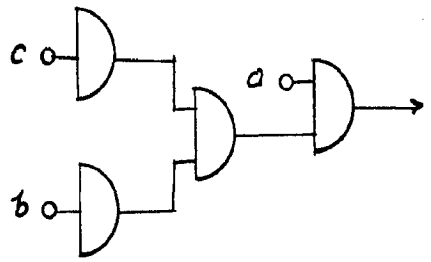
CIRCUIT NO. 12

	NUMBER	EXPRESSION
NOR	37	$a'+bc'$
NAND	7	$a'(b'+c')$



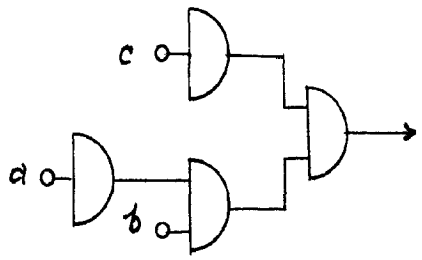
CIRCUIT NO 13

	NUMBER	EXPRESSION
NOR	7	$a'(b'+c')$
NAND	37	$a'+b'c'$



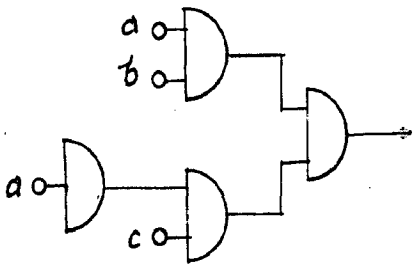
CIRCUIT NO 14

NOR	212	$(a'+b)c$
NAND	256	$c+a'b$



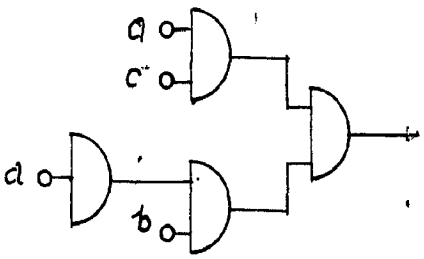
CIRCUIT NO 15

NOR	254	$a'b+ac$
NAND	—	—

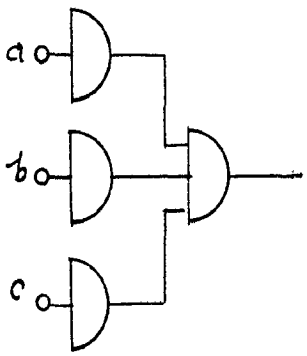


CIRCUIT NO - 16

NOR		
NAND		

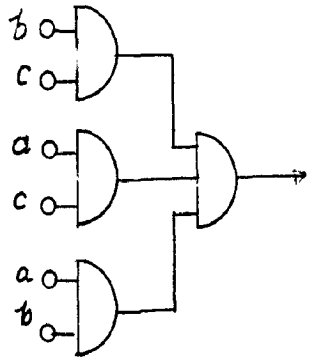


CIRCUIT NO 16X



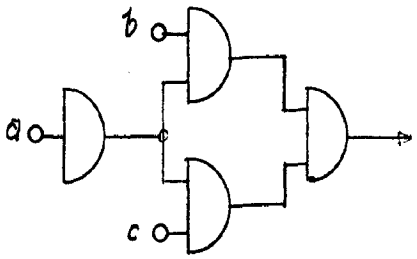
CIRCUIT NO - 17

	NUMBE	EXPRESSION /
NOR		
NAND		



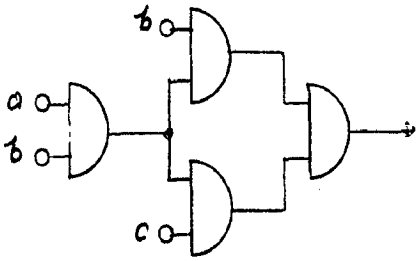
IT N 18

	NUMBE	EXPRESSION.
NOR		
NAND		



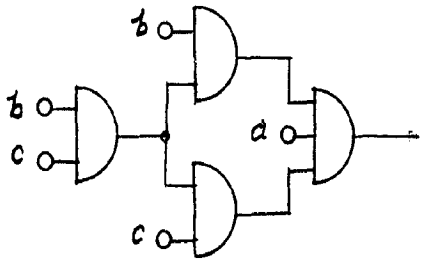
IT NO 19

NOR		
NAND		



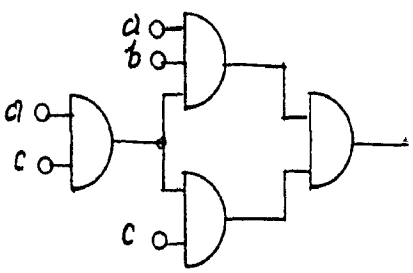
20

NOR		
NAND		

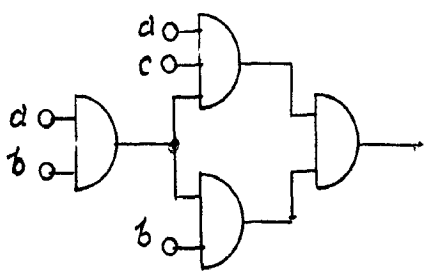


21

NOR		
NAND		

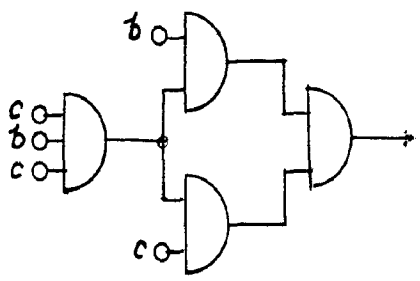


22



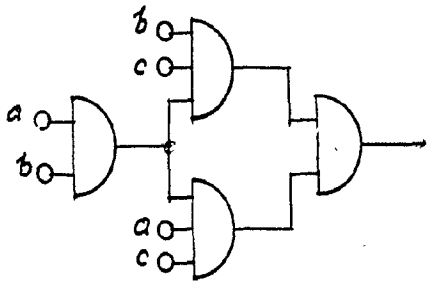
22X

	NUMBE	EXPRESSION
NOR		
NAND		



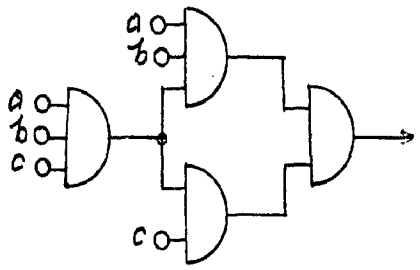
23

		EXPRESSION
NOR		
NAND		



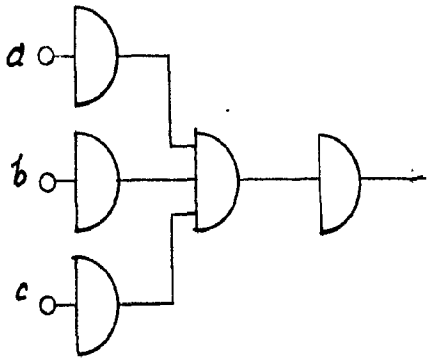
24

NOR		
NAND		



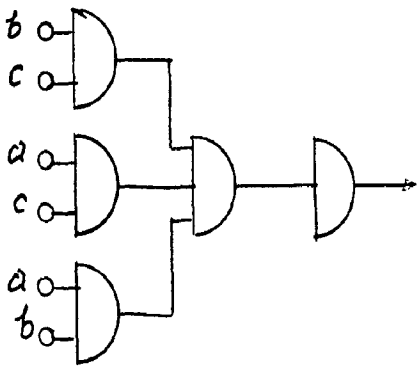
25

NOR	177	
NAND	1	



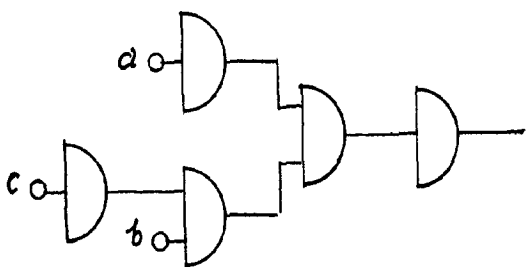
26

NOR		
NAND		



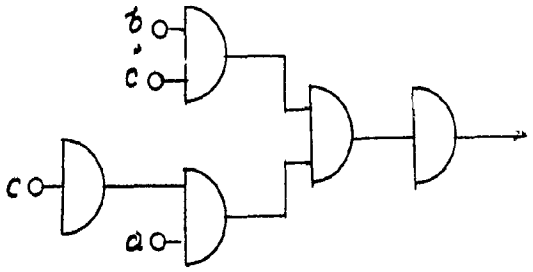
27

NOR		
NAND		



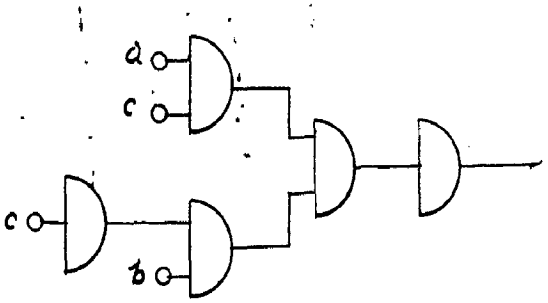
28

	NUMBER	EXPRESSION
NOR		
NAND		



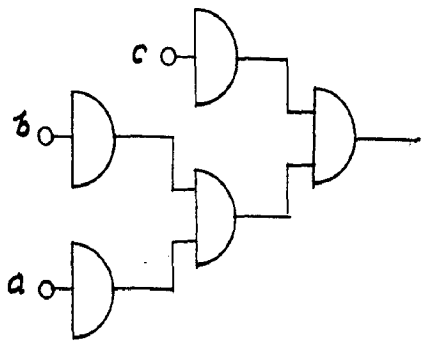
CIRCUIT NO 29

	NUMBE	EXPRESSION
NOR		
NAND		



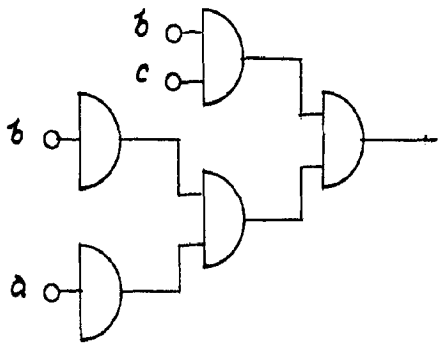
CIRCUIT NO 29X

NOR		
NAND		



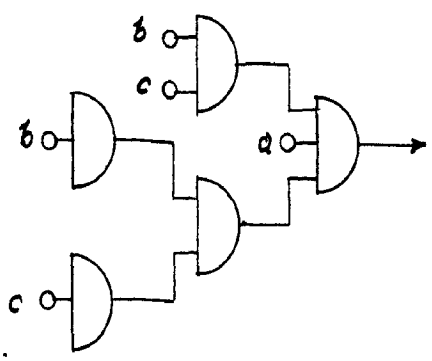
CIRCUIT NO 30

NOR	56	a'b + b'c
NAND	213	a' b'+ bc



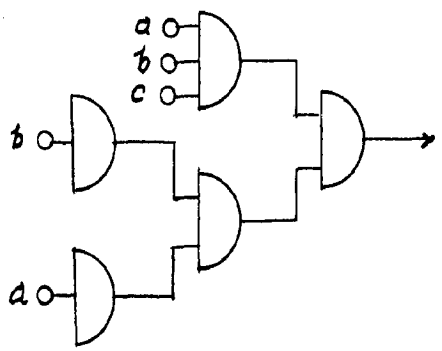
CIRCUIT NO- 31

NOR	6	a'(b'c+bc')
NAND	237	a'(bc + b'c')



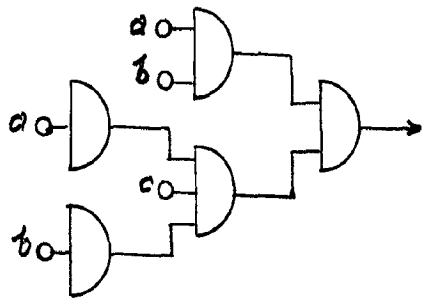
CIRCUIT NO 32

NOR	76	a' b + ab' +a'c
NAND	203	abc +a' b'



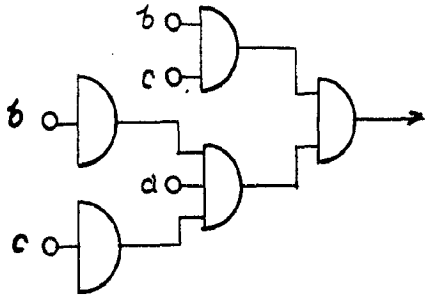
CIRCUIT NO 33

	NUMBER	EXPRESSION
NOR	274	$a(b'+c)+a'b$
NAND	–	–



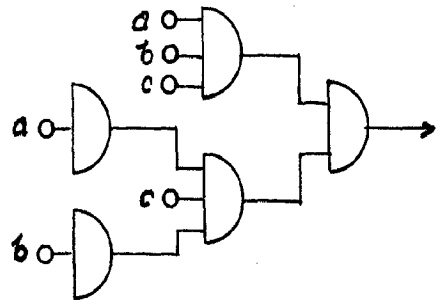
CIRCUIT NO 34

	NUMBER	EXPRESSION
NOR	–	–
NAND	230	$ab'c'+bc$



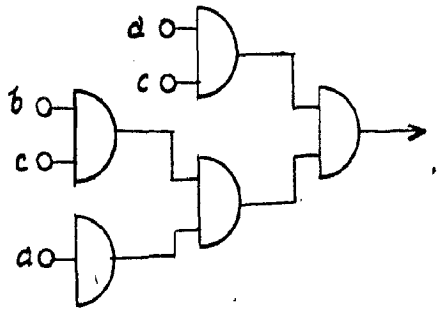
CIRCUIT NO 34X

NOR	276	$a'b+ab'+c$
NAND	202	$(ab+a'b')c$



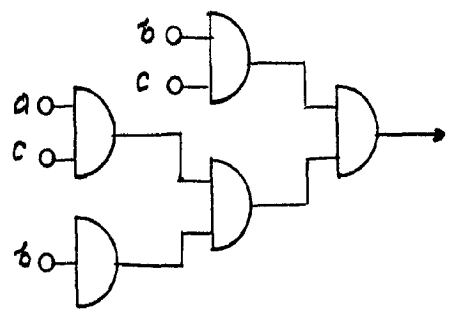
CIRCUIT NO- 35

NOR	32	$a'c+ab'c'$
NAND	–	–



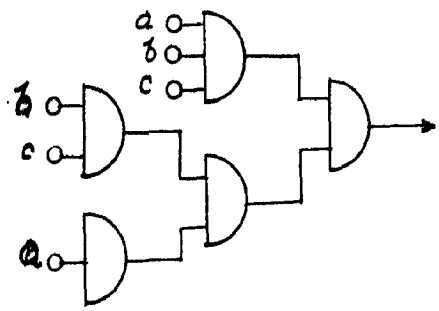
CIRCUIT NO 36

NOR	–	–
NAND	233	$bc+b'(a'+c)$



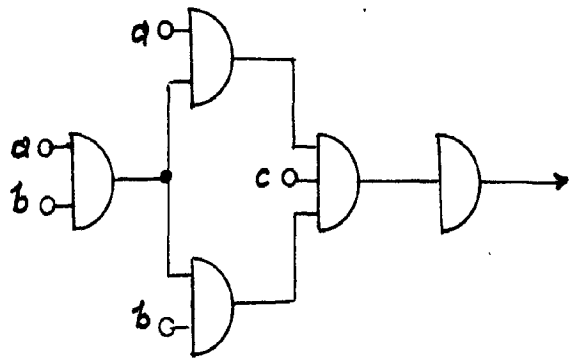
CIRCUIT NO 36X

NOR	36	$a'(b+c)+a'bc'$
NAND	207	$a'(b'+c')+abc$



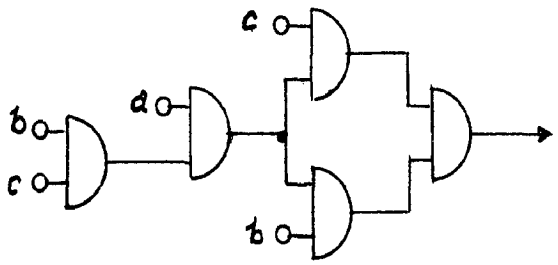
CIRCUIT NO 37

	NUMBER	EXPRESSION
NOR	276	$a'b+ab'+c$
NAND	202	$(ab+a'b')c$



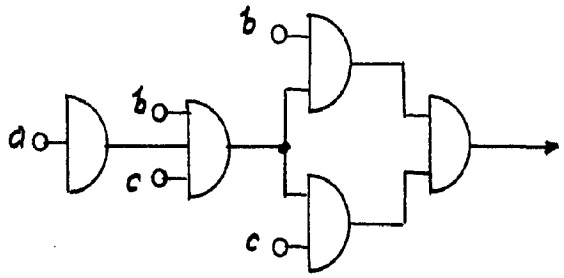
CIRCUIT NO 38

	NUMBER	EXPRESSION
NOR	216	$a'b+a'c+bc$
NAND	216	$a'b+a'c+bc$



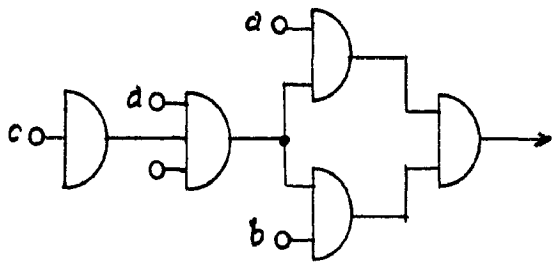
CIRCUIT NO 39

	NUMBER	EXPRESSION
NOR	230	$ab'c'+bc$
NAND	—	—



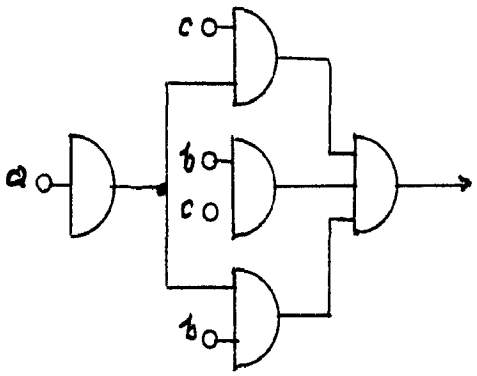
CIRCUIT NO 40

	NUMBER	EXPRESSION
NOR	—	—
NAND	274	$ab'+ac+a'b$



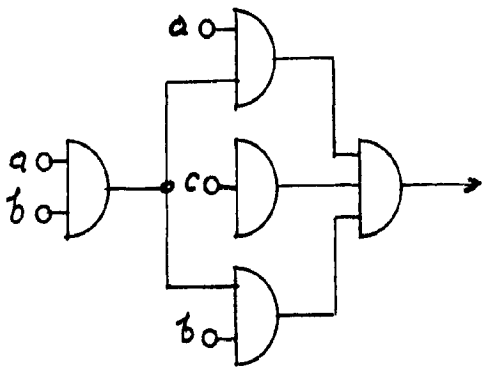
CIRCUIT NO. 40X

	NUMBER	EXPRESSION
NOR	216	$a'b+a'c+bc$
NAND	216	$a'b+a'c+bc$



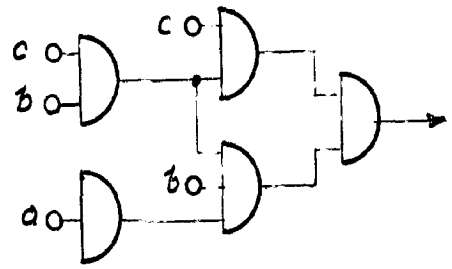
CIRCUIT NO 41

	NUMBER	EXPRESSION
NOR	202	$(ab+a'b')c$
NAND	276	$a'b+ab'+c$



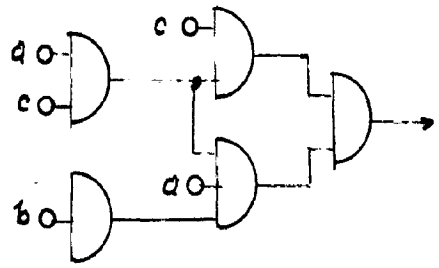
CIRCUIT NO 42

	NUMBER	EXPRESSION
NOR	233	$a'b'+b'c+bc$
NAND	—	—



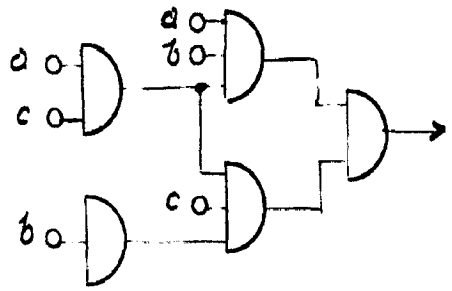
CIRCUIT NO 43

	NUMBE	EXPRESSION
NOR	—	—
NAND	32	$a'c+ab'c'$



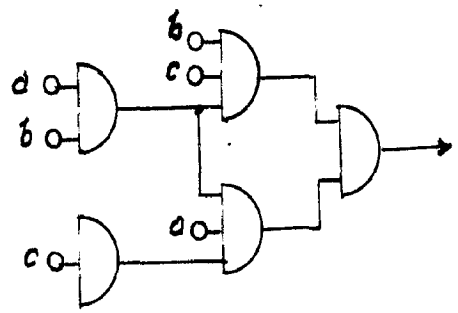
CIRCUIT NO 43X

NOR	275	$a(b'+c)+a'(b+c')$
NAND	—	—



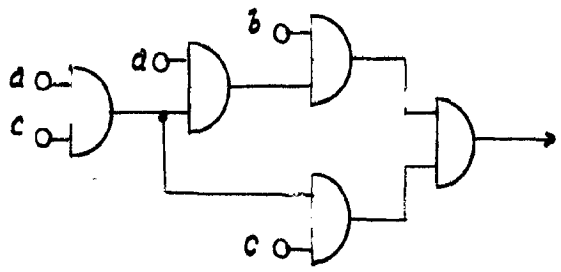
CIRCUIT NO 44

NOR	—	—
NAND	30	$ab'c'+a'bc$



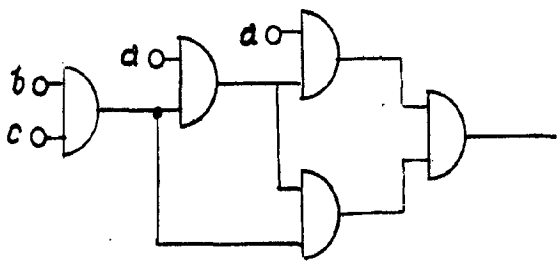
CIRCUIT NO 44X

NOR	216	$a'b+a'c+bc$
NAND	216	$a'b+a'c+bc$



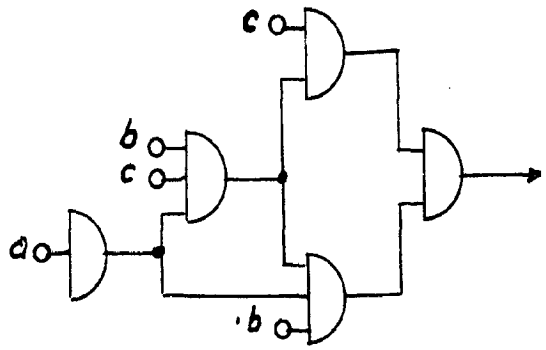
CIRCUIT NO 45

NOR	36	$a'(b+c)\ ab'c'$
NAND	207	$a'(b\|c')+abc$



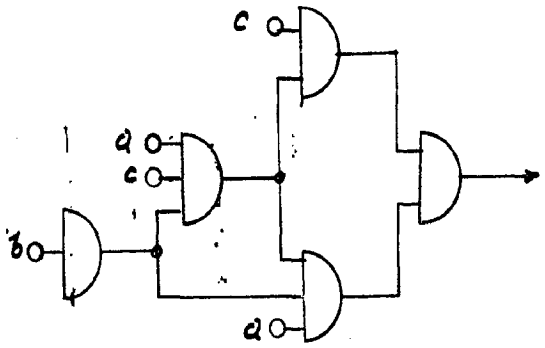
CIRCUIT NO 46

	NUMBER	EXPRESSION
NOR	232	c(a'+b) + c'ab'
NAND	—	—



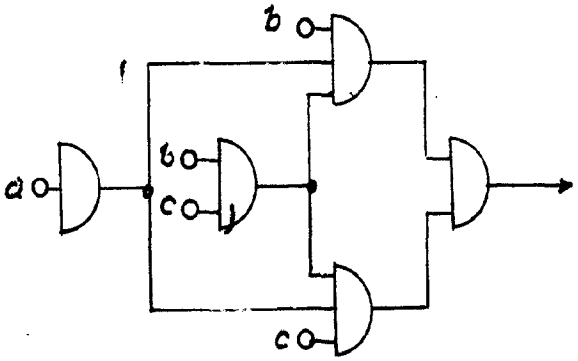
CIRCUIT NO 47

	NUMBER	EXPRESSION
NOR	—	—
NAND	232	c(a'+b) + c'ab'



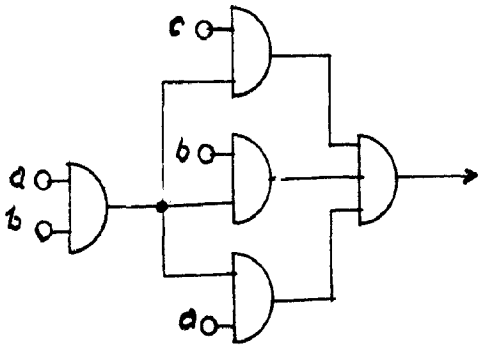
CIRCUIT NO 4X

NOR	237	bc + b'c' + a'
NAND	6	b'c + bc')a'



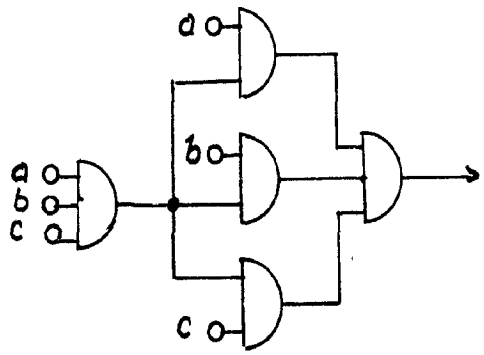
CIRCUIT NO 48

NOR	203	abc + a'b'
NAND	76	a'b + ab' + a'c



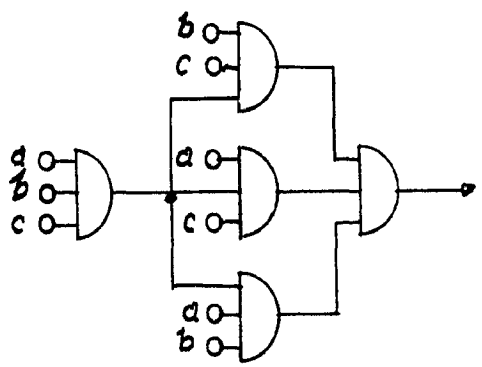
CIRCUIT NO 49

NOR	201	abc + a'b'c'
NAND	176	ab' + bc' + a'c



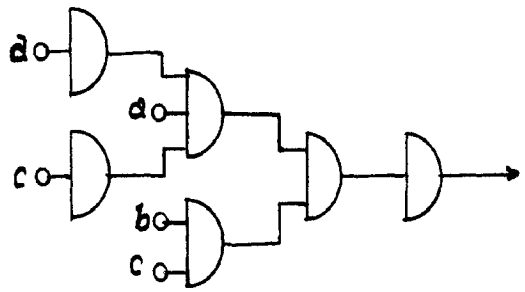
CIRCUIT NO 50

NOR	351	ab + ac + bc + a'b'c'
NAND	150	a'bc + ab'c + abc'



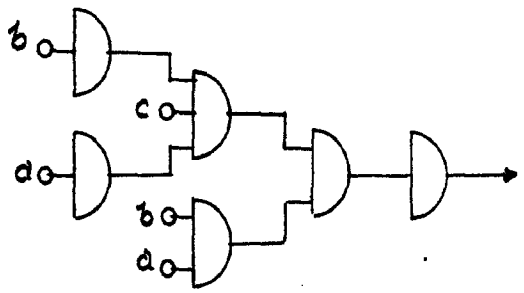
CIRCUIT NO. 51

	NUMBER	EXPRESSION
NOR	31	$a'bc + b'c'$
NAND	–	—



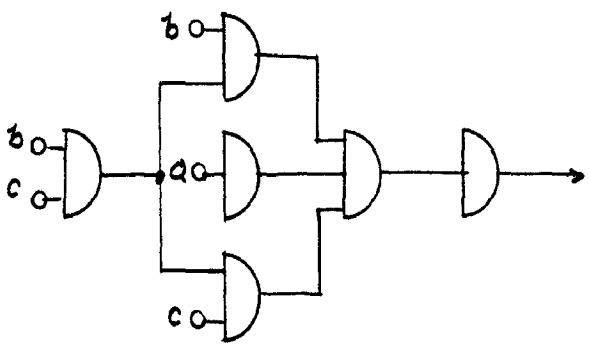
CIRCUIT NO 52

	NUMBER	EXPRESSION
NOR	—	—
NAND	75	$a'b + ab' + a'c'$



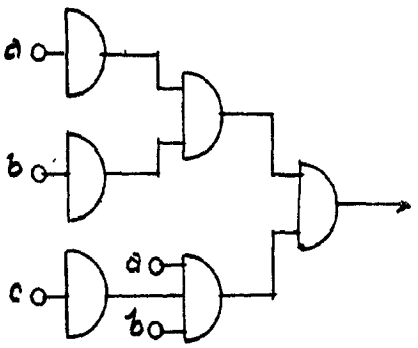
CIRCUIT NO 52 x

NOR	757	$a' + b'c + bc'$
NAND	11	$a'(b'c' + bc)$



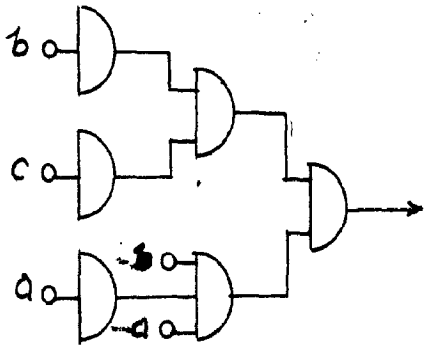
CIRCUIT NO 53

NOR	75	$a'b + ab' + a'c'$
NAND	–	—



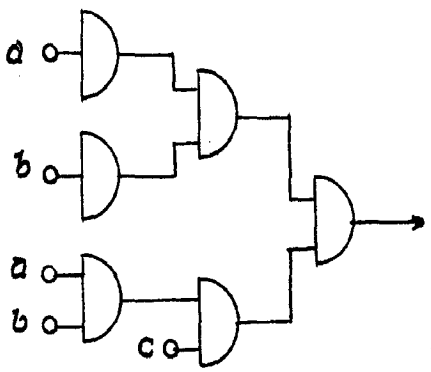
CIRCUIT NO 54

NOR	–	—
NAND	31	$a'bc + b'c'$



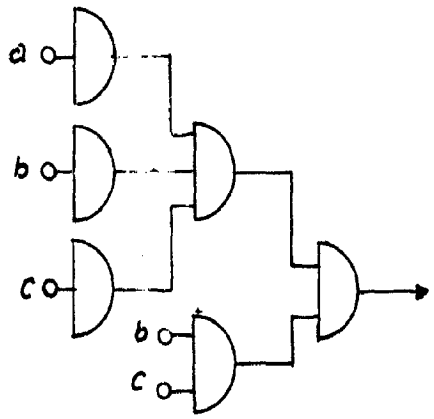
CIRCUIT NO 54 x

NOR	53	$a'b' + c(a' + b')$
NAND	53	$a'b' + c(a' + b')$



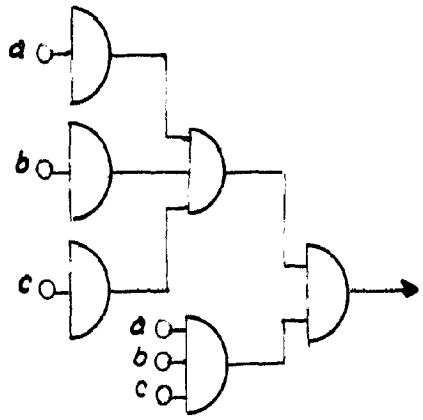
CIRCUIT NO 55

	NUMBER	EXPRESSION
NOR	256	$a'b + a'c + b'c + bc'$
NAND	211	$a'b'c' + bc$



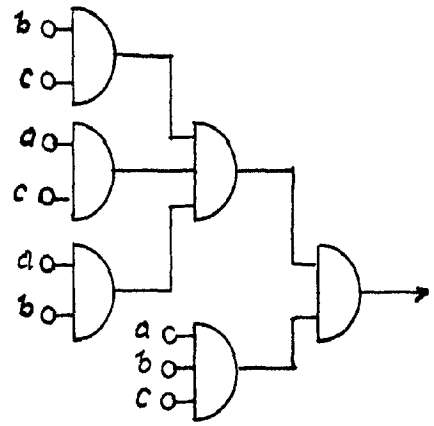
CIRCUIT NO 56

	NUMB	EXPRESSION
NOR	176	$ab' + bc' + a'c$
NAND	201	$abc + a'b'c'$



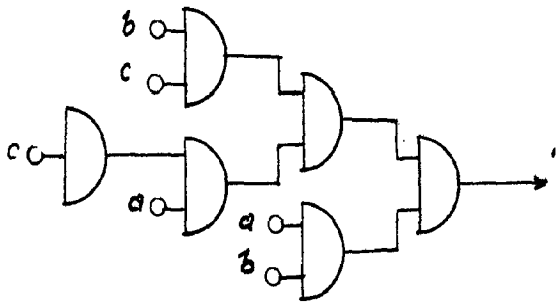
CIRCUIT NO 57

NOR	26	$a'b'c + a'bc' + ab'c'$
NAND	227	$a'b' + ab'c' + b'c' + abc$



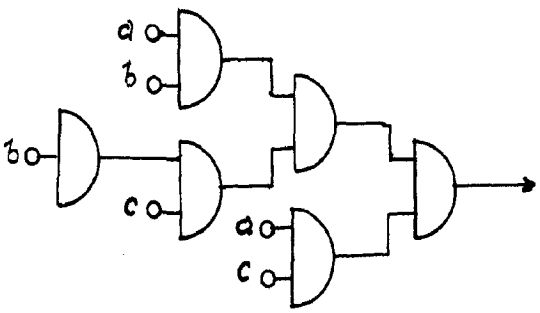
CIRCUIT 58

NOR	30	$ab'c' + a'bc$
NAND	—	—



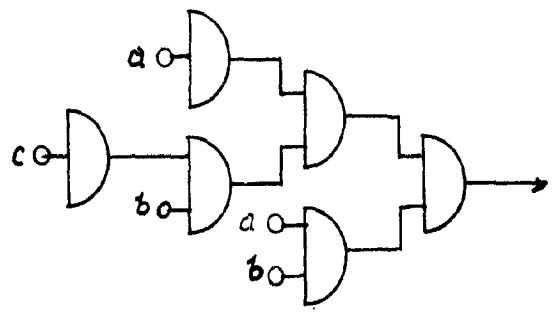
CIRCUIT NO 59

NOR	—	—
NAND	215	$ab' + a'b + ac + a'c'$



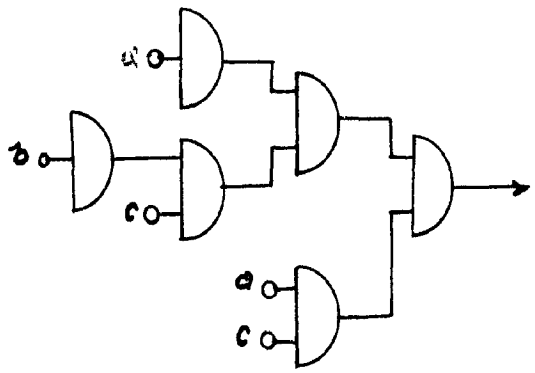
CIRCUIT NO 59X

NOR	54	$a'b + ab'c$
NAND	—	—



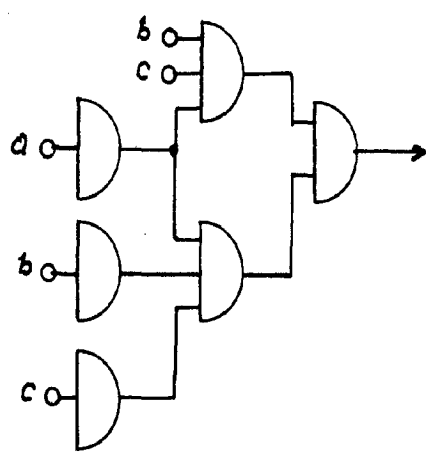
CIRCUIT NO 60

	NUMBER	EXPRESSION
NOR	—	—
NAND	255	$bc + a'c' + ac$



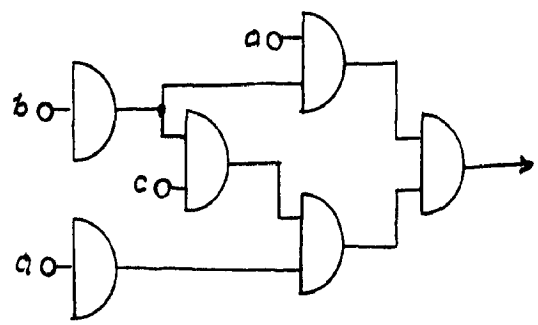
CIRCUIT NO 60X

	NUMBER	EXPRESSION
NOR	157	$a' + b'c + bc'$
NAND	11	$a' + (b'c' + bc)$



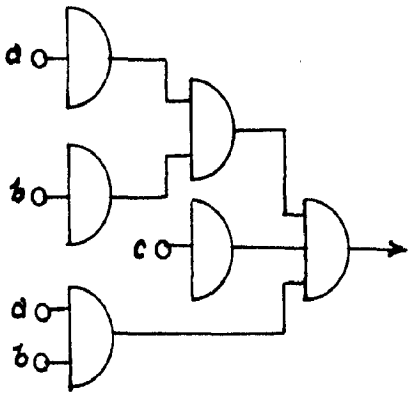
CIRCUIT NO 62

	NUMBER	EXPRESSION
NOR	—	—
NAND	75	$a'b + ab' + a'c'$



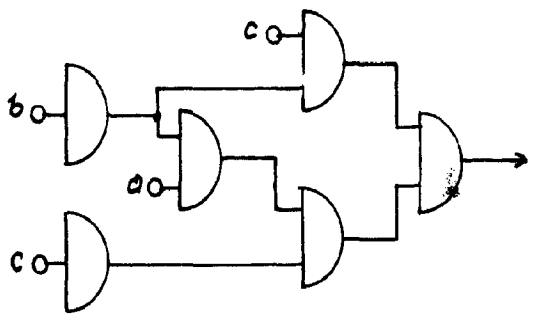
CIRCUIT NO 63X

	NUMBER	EXPRESSION
NOR	50	$a'bc + ab'c$
NAND	353	$c + ab + a'b'$



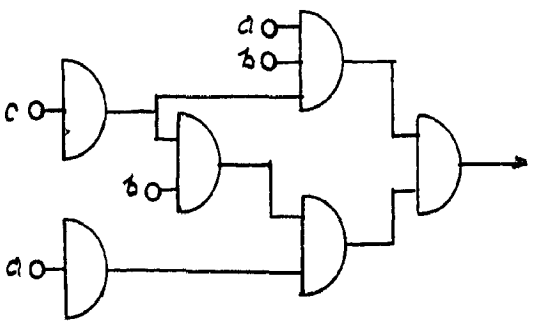
CIRCUIT NO 61

	NUMBER	EXPRESSION
NOR	31	$a'bc + b'c'$
NAND	—	—



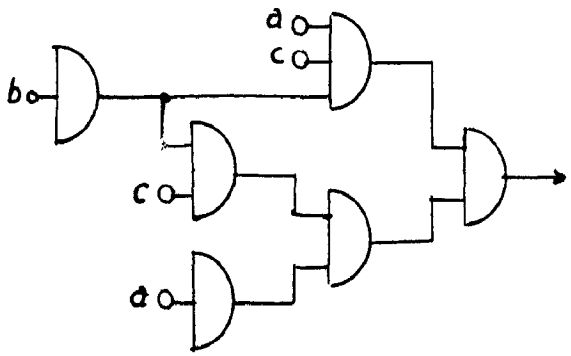
CIRCUIT NO 63

	NUMBER	EXPRESSION
NOR	55	$a'(b + c') + ab'c$
NAND	—	—



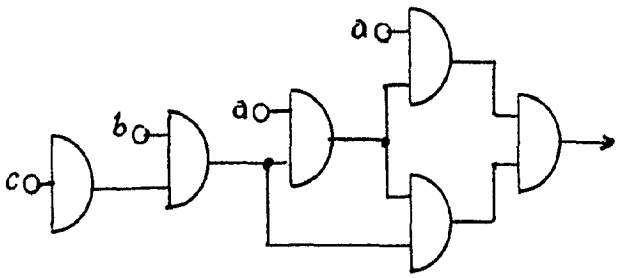
CIRCUIT NO 64

	NUMBER	EXPRESSION
NOR	—	—
NAND	55	$a'(b+c')+ab'c$



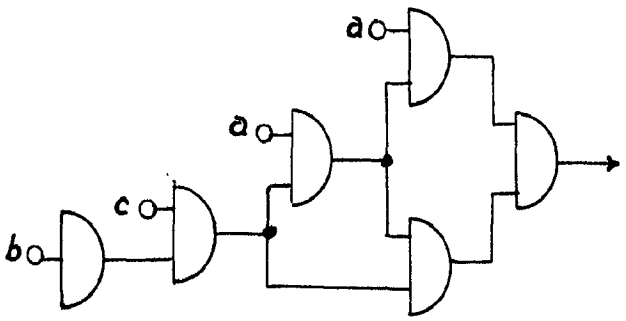
CIRCUIT NO 64x

	NUMBER	EXPRESSION
NOR	55	$a'(b+c')+ab'c$
NAND	—	——



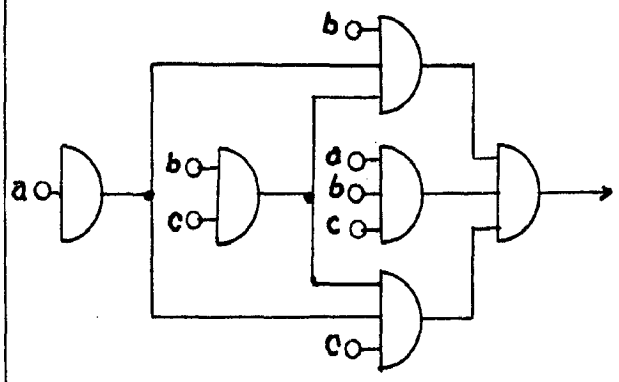
CIRCUIT NO 65

	NUMBER	EXPRESSION
NOR	—	—
NAND	55	$a'(b+c')+ab'c$



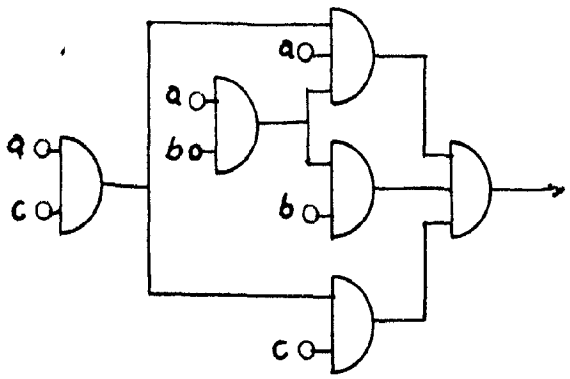
CIRCUIT NO 65x

	NUMBER	EXPRESSION
NOR	236	$a'c+bc+a'b+ab'c'$
NAND	206	$a'b'c+a'bc'+abc$



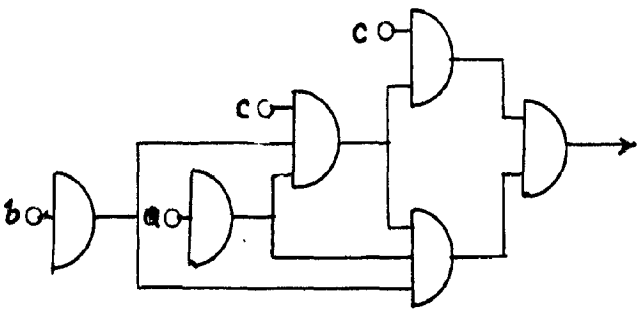
CIRCUIT NO 66

	NUMBER	EXPRESSION
NOR	207	$a'b'+a'c'+abc$
NAND	36	$a'b+a'c+ab'c'$



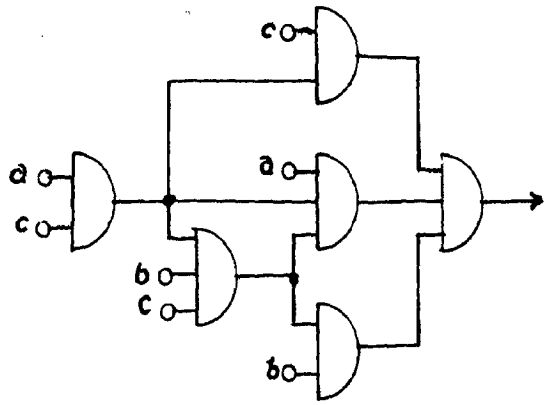
CIRCUIT NO 67

	NUMBER	EXPRESSION
NOR	152	$a'c+b'c+abc'$
NAND	251	$ac+bc+a'b'c'$



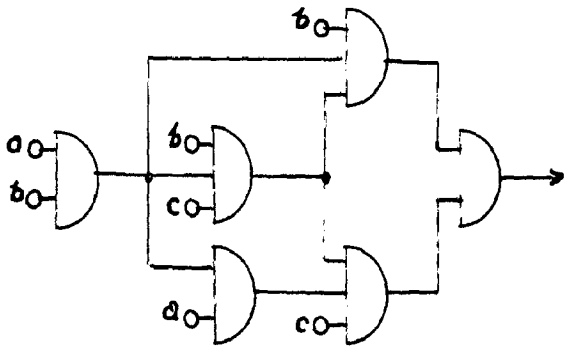
CIRCUIT NO 68

	NUMBER	EXPRESSION
NOR	206	$a'b'c + a\,bc' + abc$
NAND	236	$a'c + bc + a'b + ab'c'$



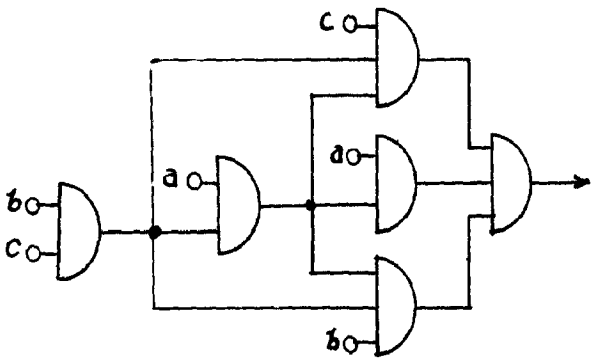
CIRCUIT NO 69

	NUMBER	EXPRESSION
NOR	236	$a'c + bc + a'b + ab'c'$
NAND	206	$a'b'c + a'bc' + abc$



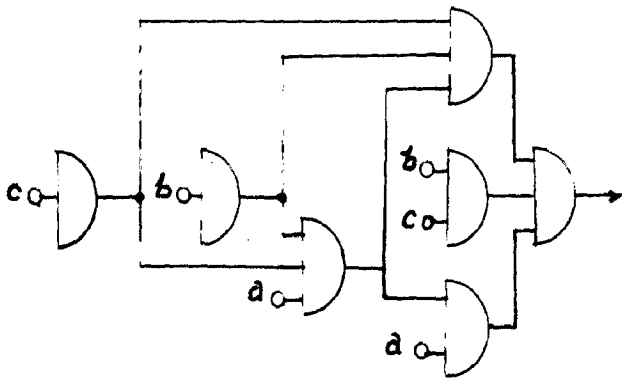
CIRCUIT NO 70

	NUMBER	EXPRESSION
NOR	236	$a'c + bc + a'b + ab'c'$
NAND	206	$a\,bc + a'b'c + a'bc'$



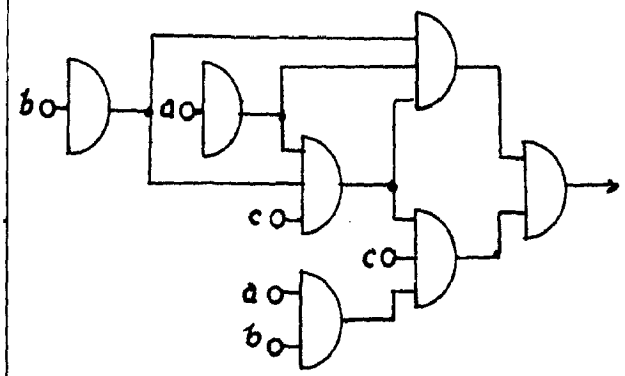
CIRCUIT NO 71

	NUMBER	EXPRESSION
NOR	150	$a'bc + ab'c + abc'$
NAND	351	$ab + ac + bc + a'b'c'$



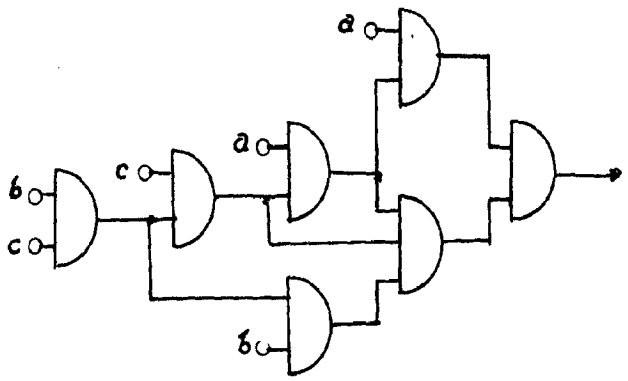
CIRCUIT NO 72

	NUMBER	EXPRESSION
NOR	153	$a'c + a'b' + b'c + abc'$
NAND	51	$a'b'c' + a'bc + ab'c$



CIRCUIT NO 73

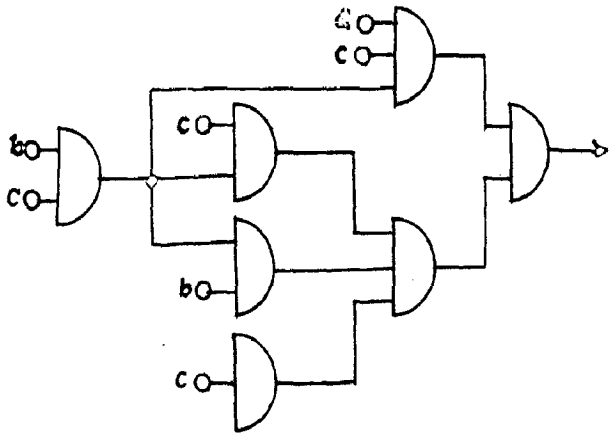
	NUMBER	EXPRESSION
NOR	153	$a'c + a'b' + b'c + abc'$
NAND	51	$a'b'c' + a'bc + ab'c$



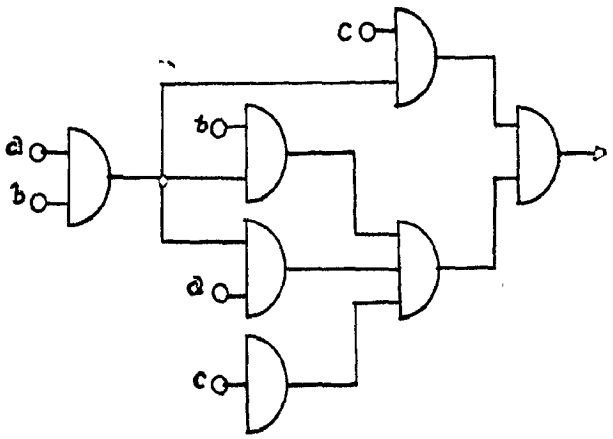
CIRCUIT NO. 74

	NUMBER	EXPRESSION
NOR	103	a'c + a'b' + bc + abc'
NAND	51	a'bc' + a'bc + ab'c

	NUMBER	EXPRESSION
	51	a'b'c' + abc + abc
NAND	153	a'c + a'b' + bc + abc'



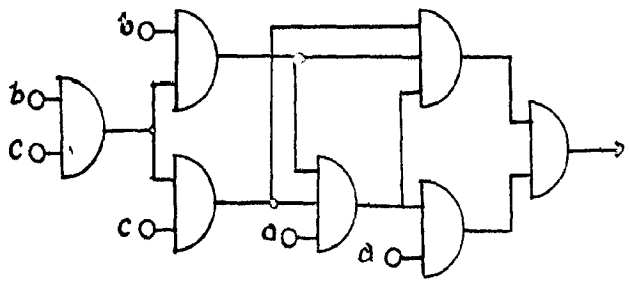
CIRCUIT NO 75



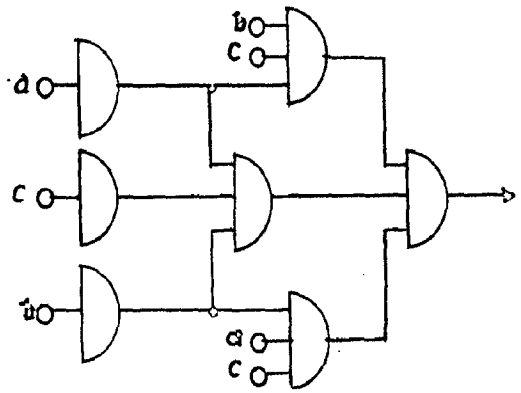
CIRCUIT NO 76

NOR	151	a'bc + a'bc + abc' + a'b'c'
NAND	151	a'bc + abc + abc' + a'b'c

NOR	153	a'c + a'b' + b'c + abc'
NAND	51	a'b'c' + abc + a'bc



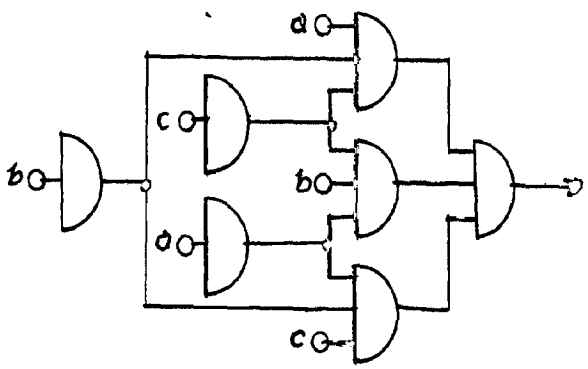
CIRCUIT NO 77



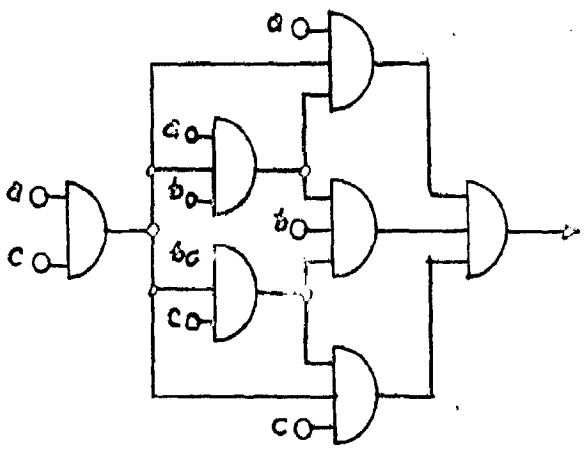
CIRCUIT NO 78

NOR	227	c'b' + a'c' + b'c' + abc
NAND	26	a'b'c + a'bc' + ab'c

NOR	226	abc + ab'c' + a'bc' + a'b'c
NAND	226	abc + ab'c' + a'bc' + a'b'c



CIRCUIT NO 79



CIRCUIT NO 80