M. Tech. (Computer Science) Dissertation Series

Studies on Fault Diagnosis in Clos Network

a dissertation submitted in partial fulfilment of the requirements for the *M. Tech. (Computer Science)* degree of the **Indian Statistical Institute**

By

Sambhu Nath Chakrabarty

under the supervision of

Professor Bhabani P. Sinha

INDIAN STATISTICAL INSTITUTE
203 Barrackpore Trunk Road
Calcutta - 700 035

July 1993

ACKNOWLEDGEMENT

I am deeply indebted to Prof. B.P. Sinha whose classic guidance has helped me to achieve the goal. His valuable advice and critical analysis of the problem is the key to sucess.
I am grateful to Dr S. C. Bagchi, The Dean of studies, for providing necessary facilities to carry out the project.

I am also grateful to Mr. Debasish Das for giving his good advices and valuable discussions about the problem.

I thank to all.

Sambhu Nath Chakrabarty

ABSTRACT

Use of crossarray is becoming more and more important nowadays. Crossarray can be implemented by crossbar and Clos network.
Our telephone switching system even now uses crossbar network. If we allow some delay Clos network can replace crossbar network with sufficient amount of reduction in number of switches. But a major problem with the implementation of Clos network for telephone switching system is the lack of fault diagnosis procedures. So, this project has been taken as a dissertation work which would enable us to diagnose different types of faults in Clos network.

CONTENTS

1. INTRODUCTION

1.1. Multistage Interconnection Network (MIN) 5
1.2. Classification of MINs 5
1.3. Scope of the Work 6
1.4. Organization of the Report 6

2. DEFINITION OF NETWORK STRUCTURE AND FAULT MODEL

2.1. Introduction to Clos Network 7
2.2. Structure Definition 7
2.3. Fault Model 7

3. FAULT DIAGNOSIS

3.1. Problem Definition and Motivation 9
3.2. Some Definitions and requirements 9
3.3. Test Vector Design and Algorithms for Switching Faults 9
3.4. Diagnosis of Single Switching Fault 12
3.5. Link Short Circuit Faults 13
3.6. Upper Bound on the Number of Tests and Test Vector Design 13
3.6.1. Upper Bound on the Number of Tests 14
3.6.2. Test Vector Design 16
3.7. Detection of Short Circuit Faults in a 3-stage Clos Network 16
3.8. Diagnosis of Link Short Circuit Faults 17
3.9. Multiple Switching Fault Diagnosis 17
3.10. Single Multiple Link Short Circuit Fault Diagnosis 18

4. CONCLUSION 19

5. BIBLIOGRAPHY 20

6. APPENDIX

6.1. Simmulation Program for ~~Fault~~ Diagnosis of Single Switching Fault 21
6.2. Simmulation Program for ~~Fault~~ Diagnosis of Single Two-link Short Circuit Fault 27

CHAPTER 1

INTRODUCTION

1.1. Multistage Interconnection Networks (MIN)

A recent trend in computing is to distribute the concurrently executable portions among a set of processing elements. There are two basic approaches to this - one is to build a *loosely-coupled system* and the other is to form a *tightly-coupled system.*

In loosely-coupled systems, the processors do not share any common clock or any common memory, but share the important resources like data files, softwares, special hardware components etc., is possible without duplicating the resources themselves. They may even be seperated from each other and connected through databuses, radio/telephone links, satellite, etc. These systems commonly reffered to as *distributed system.*

In tightly coupled systems the processors share one common clock or common memory or both, resulting a parallel processing environment. Such a system enables one to meet the requirement for enormous amount of fast real-time computations in many practical applications, e.g., weather forecasting, image processing, etc. Examples of parallel processing systems include pipelined computers, array processors and multiprocessors. A distributed/ parallel processing system involving array processors require an interconnection network for communication among processing elements.

Interconnection networks may be broadly classified as:

(a) *static interconnection network*
and (b) *dynamic interconnection network.*

A static interconnection network is a collection of processors and links among them, e.g., linear array, ring, systolic array, hypercube, barrel shifter, cube connected cycles. A dynamic interconnection network has a set of sources and a set of destinations (may be same as set of sources) which are connected through switches and links. Different input-output connection patterns may be achieved by changing switch-settings. Dynamic interconnection networks can be classified as :

i) *single stage interconnection network*
and ii) *multistage interconnection network* (MIN).

1.2. Classification of MINs

The number of stages, interstage connection topology, and switching capability of the switching elements (SE) characterize a MIN. The MIN can be considered as a permutation function which permutes the input lines to the output lines according to the control setting such that any input line may be connected to any one of the output lines. It is important to note that not all MINs are capable of providing all possible permutations,i.e.,not all requests can be serviced at all times. On this basis the MINs can be classified as follows :

(i) Blocking or non-rearrangeable type: These MINs can not provide all possible permutations because there is genarally a unique path between a given input output pair in the network and it blocks certain other input output paths from being established, e.g., baseline, omega, flip and indirect hypercube networks.

(ii) Loosely nonblocking or rearrangeable type: These MINs can provide all possible permutations of the input lines. However it is necessary to establish an alternate path for some other pair. This is called rearranging the network to achieve the required permutation, e.g., Benes, (3 $\log_2 N$ - 4) stage omega network.

(iii) Strictly nonblocking type: These MINs have multiple paths for any input output pair and it is always possible to establish a path, independent of existing paths, e.g., Clos network.

1.3. Scope of the Work

The area of fault diagnosis of MINs is assuming more and more importance because with the use of the parallel systems expanding rapidly, system reliability measures are becoming prominent parameters for system performance.

Our telephone switching system in which it is always be possible to establish a connection from an idle input to an idle output regardless of the number of calls served by the system is yet implemented by crossbar network. If we allow some delay Clos network can replace crossbar network with sufficient number of reduction in number of switches. The Clos network was developed by C. Clos [C52]. No work on diagnosis of faults in Clos network has been published till now.

Our problem is to diagnose any single fault in a 3-stage Clos network. A 3-stage Clos network defined as N(n,m,r) [B62], where r is number of input stage switches, m is number of middle stage switches and n is number of inputs per input switch. Fault in a clos network may be switching fault, intrastage link stuck-fault, and short circuit fault. Intrastage link stuck-fault can not be distinguished from switching fault without extra hardware. So, in this work we treat intrastage link stuck-fault as switching fault. We have obtained the following results:

(i) For diagnosis of single faults at most $4r\overset{+2}{\wedge}$ test vectors are required.
(ii) For visibility of short circuit faults at most $(2\lceil\log_2(2n-1)\rceil - 1)\lceil\log_2 N\rceil + 1$ test vectors are required.
(iii) For diagnosis of single short circuit faults at most $2\lceil\log_2 N\rceil + 1$ extra test vectors are required.

1.4. Organization of the Report:

The report consists of the motivation of the work, fault model definition, detection and diagnosis procedures and conclusion. Single switching faults and short circuit faults has been discussed separately. Distinguishing procedures for multiple faults has also been discussed. Simmulation programs for both short circuit fault diagnosis and single switching fault diagnosis have been developed and run. The results of those simulation programs support the results obtained in this work.

CHAPTER 2

DEFINITION OF NETWORK STRUCTURE AND FAULT MODEL

2.1. Introduction to Clos Network:

The impact of recent discoveries and developments in the electronic art is being felt in the telephone switching field. This is evidence by the fact that many laboratories here and abroad have research and development programs for arriving at economic electronic switching system. In some of these systems, such as the ECASS system, the role of switching crossnet array becomes much more important than in present day commercial telephone systems. In that system the common control equipment is less expensive, but the cross points are more expensive. Requirements for such a system are that number of cross points should be minimum. Due to these opposing requirements some compromise is required. To design a crossnet arrays where it is always possible to establish a connection from an idle input to an idle output regardless of the amount of traffic on the system, a simple square array with N inputs, N outputs and N^2 crosspoints are sufficient (fig. 1). It can be taken as an upper design limit. In case of clos network less number of crosspoints is required. The number of crosspoints, required for 3-stage Clos network, where $n = N^{1/2}$, is equal to ($6 * N^{3/2} - 3 * N$). This is less than N^2 for $N \geq 36$. In a 3-stage Clos network, the number of input stage switches are greater than or equal to number of inputs per input switch (C52).

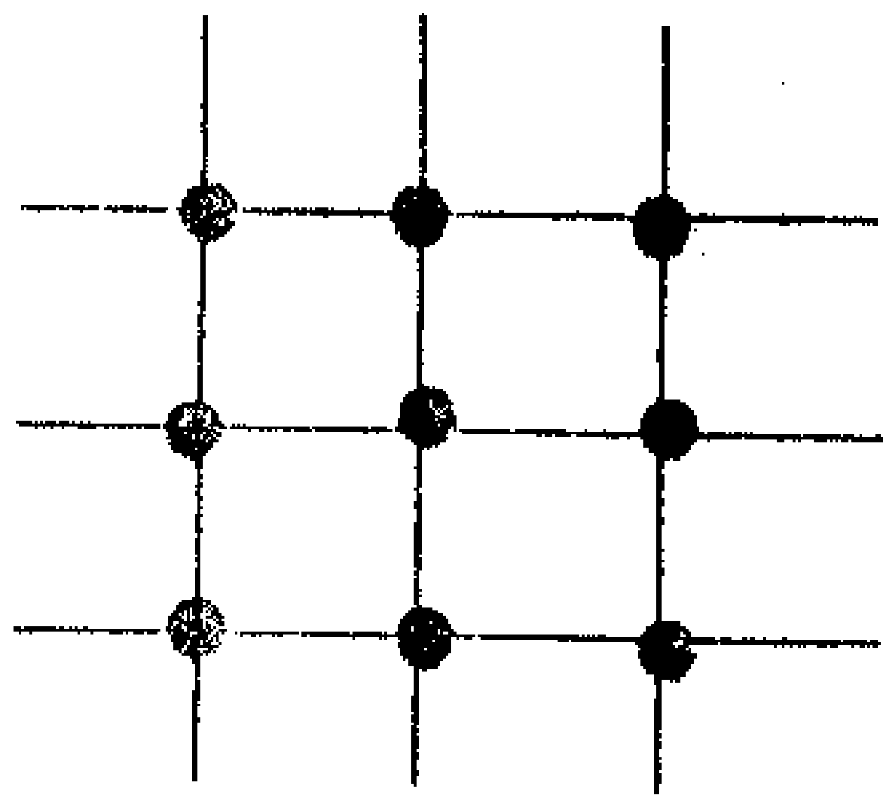


Fig. 1
3 × 3 Crossarray

2.3. Structure Definition

Each switch is basically a multiplexer demultiplexer circuit. For an **m × n** switch each input is demultiplexed by 1 to n demultiplexer and then outputs are multiplexed by m to 1 multiplexer. Here we are maintaining seperate control structure for each input line of each switch. For an m × n switch, the number of control lines for each input is equal to $\lceil \log_2 n \rceil$. Thus for an m × n switch, $m * \lceil \log_2 n \rceil$ control lines are required. Only for the output switches of size **m × n** we are maintaining $\lceil \log_2 m \rceil$ control lines for each output and in total $n * \lceil \log_2 m \rceil$ control lines are required. For all switches other than the output switches each input is demultiplexed by its control lines and for each output switch the outputs are multiplexed by its control lines.

2.4. Fault model

Fault diagnosis is concerned with detecting and locating faults in interconnection networks. As time-shared buses could introduce catastrophe to an entire computer system, systems using complex interconnection networks would be even more vulnerable to hardware or shoftware faults of connection components. To conduct the fault diagnosis, a fault model must be defined so that one knows what fault need to be tested and located.

We want to diagnosis any type of single switching fault or short circuit fault between any two links in a Clos network. According to our model the faults in a Clos network can be classified as:

(1) Switching faults: Switching faults in a Clos network can be classified as:

(a)Control stuck-fault: If any control line of any switch is stuck at 0 (respective 1), it is called control stuck-at-0 (respective stuck-at-1) fault.

(b)Input stuck-fault: If any input of any switch is stuck at 0 (respective 1), it is called input stuck-at-0 (respective stuck-at-1) fault. If any input of any switch is open, it is also modelled as input stuck-fault.

(c)Output stuck-fault: If any output of a switch is stuck at 0 (respective 1), it is called output stuck-at-0 (respective stuck-at-1) fault. If any output of any switch is open, it is also modelled as output stuck-fault.

(d)Interstage link stuck-fault: In multiplexer or demultiplexer circuit of any switch input or output of some gate has been opened, i.e., some input of a switch has lost connection with some output of that switch.

(e)Control short circuit fault: Two control lines corresponding to a specific input of a switch may be shorted together.

(2) Link short circuit fault:

A second type of fault which are not switching faults is known as link short circuit faults, in which any two links of the network may be shorted together.

(3) Intrastage link stuck-fault:

Any link in between two stages is stuck at 0 or 1. This can not be distinguished from switching faults. So throughout the report it will be considered as switching fault.

In fig. 2 if a or b is opened, it will be treated as inter stage link stuck-fault and if c is open i.e. c is high, it will be treated as input and output link short circuit fault. If number of output of a switch is equal to $2^k + 1$, then the stuck at 0 fault at the most significant position of the control lines of any of the input lines is equivalent to the interstage link stuck-fault between that input and last output line.

In case of switching faults the faulty output will remain stuck at 0 or 1 for any type of test vector during a particular switch setting. In case of link short circuit fault the faulty output will set to the lower value of them who are involved in short circuit.

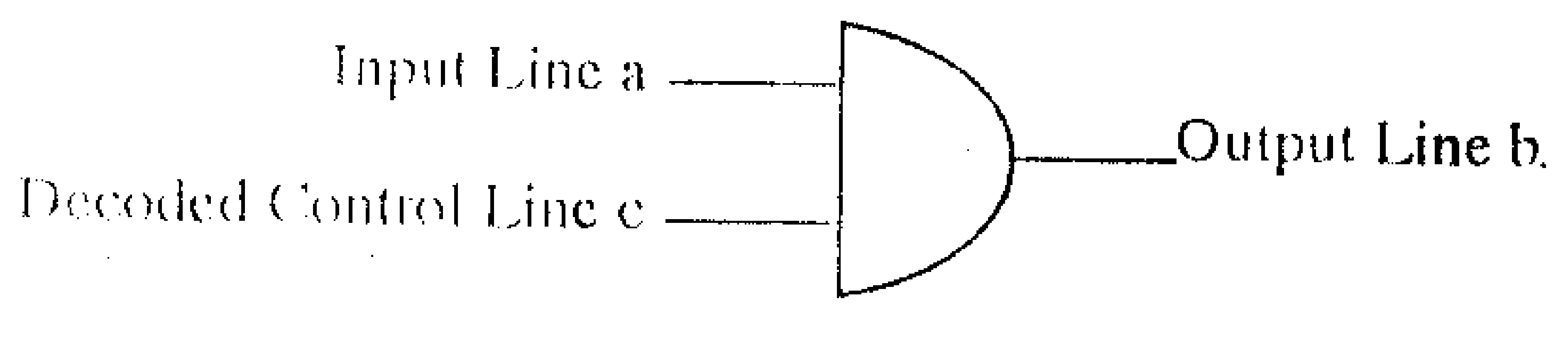


Fig. 2

CHAPTER 3

FAULT DIAGNOSIS

3.1. Problem Definition and Motivation

Our problem is, ''To study and develop diagnostic procedures for any type of single switching faults and link short circuit fault between any two links in a Clos network''. The area of fault diagnosis of MINs is assuming more and more importance because with the use parallel systems expanding rapidly, system reliability measures are becoming prominent parameters for system performance. To guarantee fault free performance it is therefore necessary to be able to test system components. It is desirable that such test procedures be algorithmic in nature so that automated implementation is possible and in view of widespread LSI/VLSI implementation of most system components, such test procedures should access only the input/output terminals.

Fault diagnosis of Clos network is necessary because Clos network is cost effective in the telephone switching system with respect to crossbar. This has been motivated us to choose this work as my dissertation work.

3.2. Some Definitions and Requirements

Definition 1 : A set of inputs designed for testing purpose is known as ***test vector.*** A particular choice of set of links which will remain on at a time defines a *phase*. In a phase a particular control setting of the input, output and middle stage switches defines a ***subphase.***

To detect the faults in a Clos network we should ensure that (a) each input of each switch should get connection with each output of that switch, (b) every link should get complementary test inputs, and (c) each pair of links in the network receive complementary test inputs. Here during testing switching faults and link short circuit faults has been taken seperately.

3.3. Test Vector Design and Algorithms for Switching Faults

In these cases faulty behaviour is that an input of the faulty switch is not getting connection with some output of it. To detect these faults we should ensure that (a) each link is on at least once, (b) each input of each switch should get connection with each output of that switch, (c) every link should get complementary test inputs.

Lemma 1:
Two phases, each having r subphases can ensure (a) and (b).

Proof:
Number the middle stage switches as 0,1,..,(2n-2). Define two phases as:

(i) Only the even numbered middle stage switches will be fully on in first phase. [Note : number of even numbered middle stage switch is equal to n]. Even numbered outputs of input switches and even numbered inputs of output switches will remain on in that phase.

(ii) Odd numbered middle switches and middle switch numbered 0 will be on in second phase.

Two phases are shown in fig. 3 for a 16 × 16 Clos network.

Thus by these two phases each input and output of each switch will be on at least once, i.e., satisfying (a). Now in each phase rotate the connections r times each time shifting by one. These subphases for phase 1 is shown in fig. 4.

Thus in first phase (respective second phase) (i) each input of each even (respective odd) numbered middle switch will get connection with each output of it, (ii) each input of each input switch will get connection with each even (respective odd) numbered output of it, (iii) each output of each output switch will get connection with each even (respective odd) numbered input of it.

Thus it is satisfying (b).

Algorithm A : Test vector generation

1. All links of stage 0 receive test input 0.

Definition 2 : Each subphase of each phase testing consists of setting all the SEs in the mode of that subphase of that phase and performing the following tests.

(i) Feed the test vector generated according to algorithm A and observe the output responses.

(ii)Feed the complementary test vector and observe the output responses.

This definition is satisfying (c).

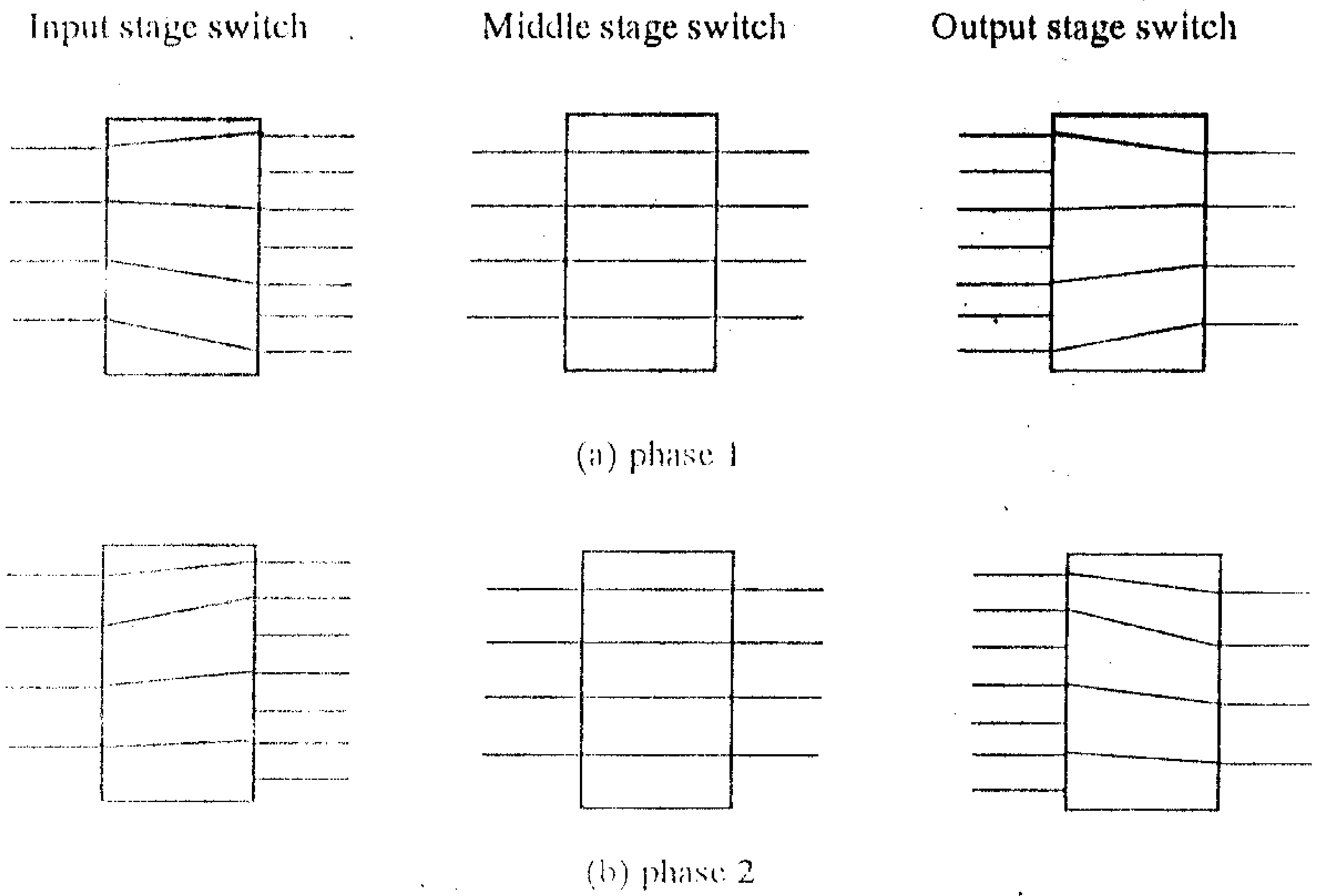
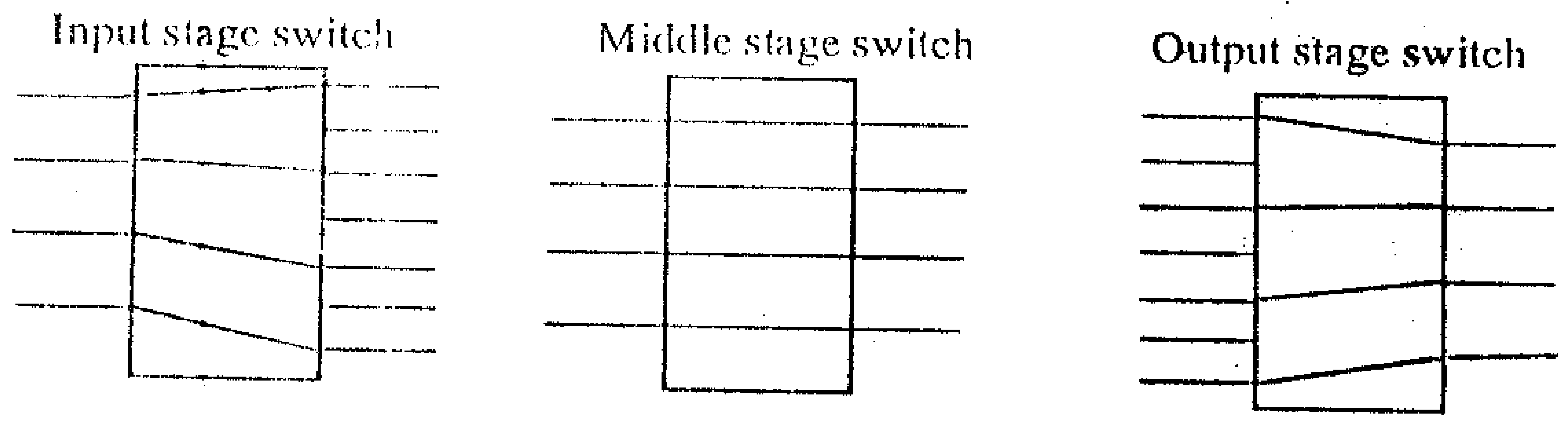
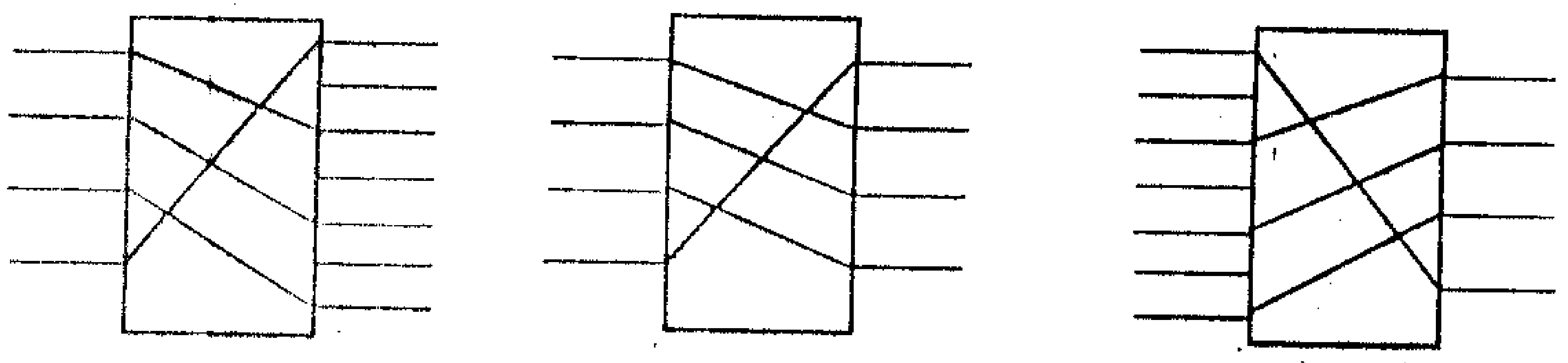


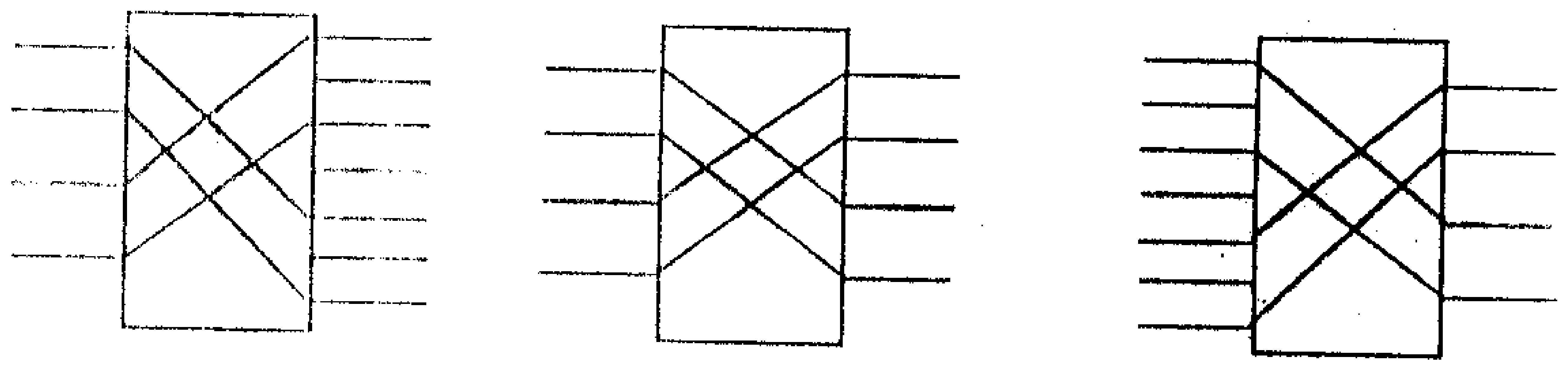
Fig.3
Connection patterns of input output and middle stage switches in a 16 × 16 3-stage Clos network



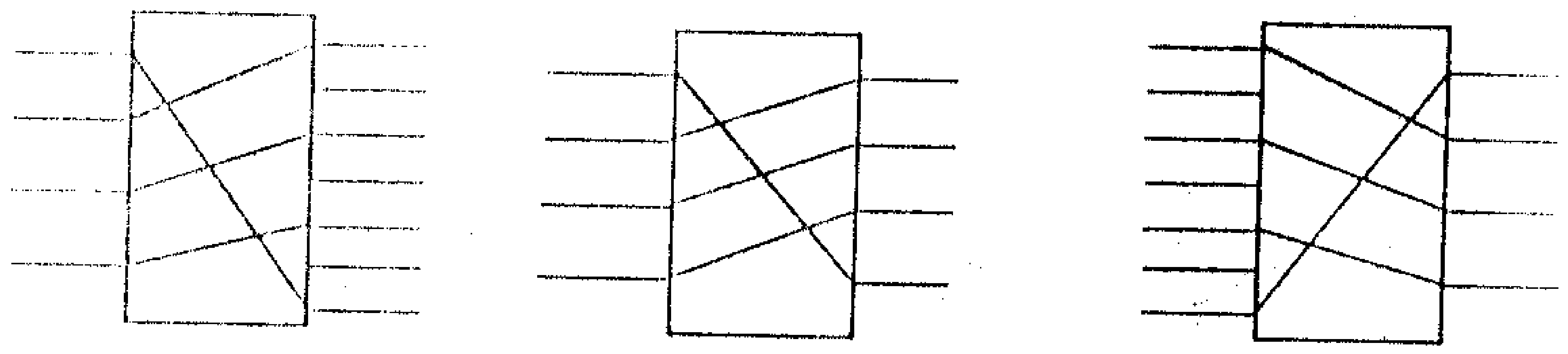
(a) Subphase 1



(b) Subphase 2



(c) Subphase 3



(d) Subphase 4

Fig. 4

Connection patterns of input, output and middle stage switches in 16 × 16 3-stage Clos network

Algorithm B : Testing a Clos network for single switching fault

(1) Generate the test vector by algorithm A.

(2) Using test vector of step 1 perform each subphase of each phase testing (as in definition2).

(3) If there exists a single switching fault some faulty response will be observed. Note those faults and faulty phases.

(4) If more than one faulty response is observed in a subphase of a phase then it is a multiple fault, else it is a single fault.

For diagnosis of single fault we will follow the procedure discussed in 3.4. For diagnosis of mutiple faults we would follow the menthods discussed in 3.9.

Thus total number of test vector required for fault detection = 2 * r * 2 = 4 * r .

3.4. Diagnosis of Single Switching Fault

Definition 3 : Define the output response unary if the observed response is 00 or 11 in the two tests of a subphase. Define it as binary if the observed response is 01 or 10 in the two tests of a subphase. Normal response is 01.
Note that all switching faults will show unary response.

Algorithm C : Detection of faulty switch and diagnosis of fault.

(1)For each control setting where the unary fault was observed (a) set the network with that control setting (b) mark the faulty path.

(2)From the marked paths find the common part where they converse.

(3)If the common part is only an input (respective output) stage switch, then (a) if fault is in every subphase of every phase then it is input stage input stuck-fault (respective output stage output stuck-fault) and (b) else it is control short circuit or control stuck-fault. To distinguish them run the algorithm D.

(4)If the common part is one input stage switch and one middle stage switch, then (a) If faulty response occurs in all subphases of a phase, then fault is any of these three types: (i) Input stage switch output stuck-fault. (ii) Middle stage switch input stuck-fault. (iii) Intrastage link stuck-fault.These three can not be distinguished without extra hardware. (b) Else it is middle stage control short circuit or control stuck-fault. To distinguish them we would run algorithm D.

(5)If common part is one output stage switch and one middle stage switch and faulty response occurs in all subphases of a phase, then fault is any of these three types: (i) Output stage switch input stuck-fault. (ii) Middle stage switch output stuck-fault. (iii) Intrastage link stuck-fault.These three can not be distinguished without extra hardware.

(6)If input, output and middle stage switches are same and fault is in only one subphase of a phase, then it is interstage link stuck-fault. To find the faulty switch (a) Fix the input and output stage switch control settings to the same values with the subphase where faulty response was found and the middle stage switch control settings completely different with that subphase. (b) Generate the test vector by algorithm A. (c) Using the test vector of step (b) and connection pattern of step (a) perform test according to definition 2. (d) If there exists no faulty response then fault is at middle stage switch, else if faulty response is at same output then it was at output stage switch, else it is at input stage switch.

(7)In all other cases it is multiple fault.

Algorithm D : Distinguishing between control stuck-fault and control short circuit fault.

(1)Bitwise AND the control values of the marked input (respective output) of the faulty input or middle (respective output) stage switch.
(2)If any of the bits results to 1 then it is the control stuck-at-0 fault at that control line.
(3)Else {
    Bitwise OR the control values of the marked input (respective output) of the input or

```
middle (respective output) stage.
(a) If any of the bits results to 0, then it is the control stuck-at-1 fault at that control line.
(b) Else {
        Let p be the number of control lines. Number them by 0 through p-1 from
        lsb to msb. Generate ($^pC_2$) different combinations. For each combination
        XOR the corresponding control values for all the unmarked outputs (respec
        tive inputs) of the faulty input or middle (respective output) stage switch. if
        for a combination result is 0 for all unmarked outputs, then it is control short
        circuit fault between those two lines.
    }
}
```

3.5. Link Short Circuit Faults

Upto section 3.4 we have seen how single switching faults in a 3-stage Clos network can be diagnosed. There is another type of fault namely, short circuit fault among the links of the network. The problem is ''To study and develop diagnostic procedures for link short circuit fault in a Clos network''.

A little insight into the problem will reveal that this general problem has the following parameters:

(1) the number of links that belongs to a short circuit group.

(2) the number of stages of the Clos network.

(3) the number of disjoint groups of shorted links or in other words, whether there is a single link short circuit fault or multiple link short circuit faults.

(4) restrictions on the location of the fault i.e. whether or not the shorted links belong to the same stage.

Here we have restricted the the problem to single fault with the number of shorted links limited to two. The location of the fault is unrestricted, i.e., the fault may occur in any one of the switches in any one of the switches.

Here the upper bound on the number of tests has been obtained and the corresponding test vectors are also developed. The fault diagnosis of link short circuit fault in a group of two links has been completed. The results of this study summarised as follows:

(1) link short circuit fault needs at most $(2\lceil \log (2n-1)\rceil - 1)\lceil \log N\rceil + 1$ test vectors for visibility of fault.

(2) a 3-stage Clos network with a 2 link short circuit fault needs at most $2\lceil \log N\rceil + 1$ extra test vectors for diagnosis.

3.6. Upper Bound on the Number of Tests and Test Vector Generation:

The logical value of a shorted link is zero. The paths on which the shorted links lie will give a 0 output response at the corresponding outputs when the test vector provides complementary test inputs to these paths. The short circuit becomes transparent otherwise, .e.g., fig. 5.

Thus the link short circuit fault behaviour can be characterised as

(a) fault is visible when the short circuited links lie on different paths which receive complementary inputs.

(b) fault is transparent when either the shorted links lie on different paths which receive same test inputs or they lie on same path.

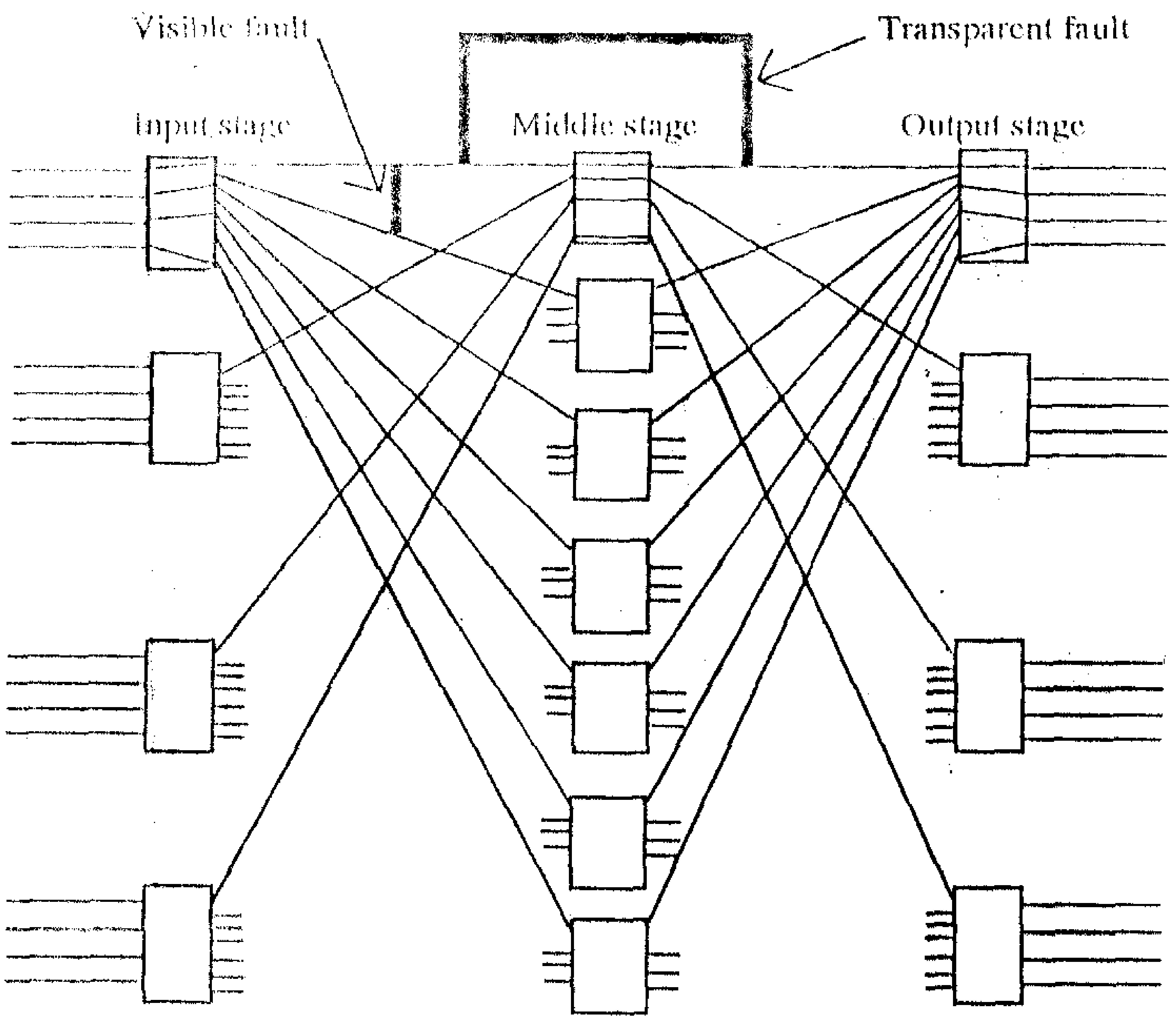


Fig. 5
Visible and Transparent faults in 16 × 16 3-stage Clos network

3.6.1. Upper Bound on the Number of Tests:

Since we assume that the shorted response is zero, whenever a link short circuit fault is visible with a given test vector, the faulty response will be observed only at some of the paths involved in the short circuit. To get hold of the other faulty paths we use the test strategy of testing the network in the same control setting with the complementary test vector. This is similar to the pair of tests discussed in definition 2.
Henceforth, we will always bear in mind that for a visible link short circuit fault exactly two outputs will be observed with complementary test vectors. Therefore to test a link short circuit fault we need to ensure that all pair of links in the network receive complementary test inputs.

Lemma 2 : For an N × N Clos network, at most $\lceil \log_2 N \rceil$ test vectors are needed to ensure that all pairs of links which are on in a particular control setting belonging to the same stage receive complementary test inputs.

Proof : Let f(p) be the number of test vector needed for the stated purpose for p input lines. our goal is to find f(N). Now any test vector applied to an N × N Clos network will have k 0's and (N-k) 1's, $0 \leq k \leq N$. With such a test vector all such pair of links of the input side one of which receives a 0 test input and the other 1, get tested for link short circuit fault. Further tests are required to test link short circuit fault among the group of links that received a 0 (size k) and the group that received a 1 (size N-k) independently. Hence we can write the following recurrence relation

$$f(N) \leq 1 + \min(\max(f(k), f(N-k))).$$

Clearly f(k) is a nondeceasing function the optimum value of k is $\lceil N/2 \rceil$, so that

$$f(N) \leq 1 + f(\lceil N/2 \rceil).$$

The solution of the equation is $f(N) \leq \lceil \log_2 N \rceil$.
Thus it can be concluded that all the pairs of input lines of a Clos network can be provided complementary test inputs using at most $\lceil \log_2 N \rceil$ test vectors.
Now, in case of Clos network all links will not be on at a time. But we have to test link short circuit fault for each pair of links.So we have to use some strategy such that each pair of links should remain on at a time at least once within the test procedure.

Lemma 3 : For an N × N $\overset{\text{3-stage}}{\wedge}$ Clos network $(2 \lceil \log_2 (2n-1) \rceil - 1)$ tests are sufficient to ensure that
(a) each pair of links should remain on at a time at least once and
(b) every fault should be visible.

Proof : Instead of giving a direct proof we are describing one strategy which will give the proof. All input and output switches will remain on during one test if all inputs are on.So, if we can show that $2 \lceil \log_2 (2n-1) \rceil - 1$ tests are sufficient to ensure that each pair of output of an input switch should remain on at a time at least once, it will be enough for (a). Take (2n-1) output of input switch (respective input of output switch) as 2n-1 bits numbered from left to right and the bit value will be 0 or 1 if corresponding link is on or off respectively.
(1) First fix the 1st bit value to 0 and generate 2 types of bit pattern given.below.

Bit Position		0	1	2	3	....	2n - 2
Bit	Type I	0	1	0	1	....	0
Value	Type II	0	0	1	0	....	1

Thus first output (respective input) line will remain on with every other output (respective input) lines at least once. (proof is obvious)

(2) Next fix first bit value to 1 and 2nd & 3rd bit value to 0 and generate two types of bit pattern given below.

Bit Position		0	1	2	3	....	2n - 2
Bit	Type I	1	0	0	1	....	0
Value	Type II	1	0	0	0	....	1

Thus after step 2 the state is like that first three
output (respective input) lines will remain on with every other output (respective input) lines at least

once.
At ith step we fix first ($2^{(i-1)}$ - 1) bits to 1 and next $2^{(i-1)}$ bits to 0 and for remaining 2n - 2^i bits two types of bit pattern as before.($i \leq (\lceil \log_2 (2n - 1) \rceil - 1$)
Note that here in every case number of bits getting value 0 is equal to $2^{(i-1)} + (2n - 2^i) / 2$, i.e., n.
This way after $\lceil \log (2n - 1) \rceil - 1$ steps number of lines which will remain on with other output line at least once is equal to

$$2^{(\lceil \log (2n-1)\rceil - 1) - 1} - 1 + 2^{(\lceil \log (2n-1)\rceil - 1) - 1}$$
$$= 2^{(\lceil \log (2n-1)\rceil - 1)} - 1$$
$$= n-1 \text{ ,if } 2n-1 = 2^k-1 \text{ for some integer } k$$
$$> n-1 \text{ otherwise.}$$

Then generate a connection pattern such that n inputs (respestive outputs) are connected to last n outputs (respective inputs). Then we have reached to a point that all outputs of an input switch (respective all inputs of an output switch) are pairwise on at least once.
Number of tests = $2 (\lceil \log_2 (2n - 1) \rceil - 1) + 1 = 2 \lceil \log_2 (2n - 1) \rceil - 1$.
Thus it is satisfying (a).
These tests are numbered like $s_0, s_1, .., s_{(2 \lceil \log (2n-1) \rceil - 1)}$.
For all tests other than s_1, s_2 the control settings are in straight mode and for test s_1 and s_2 the control settings are shifted by plus one for input and middle stage switches and minus one for output stage switches from straight mode.
Thus in s_1, s_2 all links which were in same path in other phases will be in different paths. Therefore every fault will be visible at least once within $2 \lceil \log_2 (2n - 2) \rceil - 1$ tests.

3.6.2. Test Vector Design

The following algorithm generates the test vectors for detecting single short circuit fault.

Algorithm E : Test vector to test an N × N 3-stage Clos network, where $p = \lceil \log_2 N \rceil$

```
(1) For i:= p downto 1 do
    (a) classify the input lines into odd and even lines based on the i_leftmost bits of
        their addresses.
    (b) Even input lines receive a test input 1, odd lines receive a test input 0.
end for
```

Fig. 6 shows the test vectors generated for a 16 × 16 3-stage Clos network.

3.7. Detection of Short Circuit Faults in a 3-stage Clos Network

Following algorithm describes the procedure for single link short circuit fault detection.

Algorithm F :

```
(1) For each phase $s_0, s_1, .., s_k$ for each test vector test the network until faulty response is
    observed where $k = ( 2 \lceil \log_2 ( 2n - 1 ) \rceil - 1)$.
(2) During step 1 if faulty response is found, test the network with the complementary test
    vector of that vector and in that phase in which phase and by which vector it was found.
```

The test procedure dicussed above will cause exactly two faults. In worst case testing upto last phase and last test vector is required and then one test by complementary test vector. Thus total number

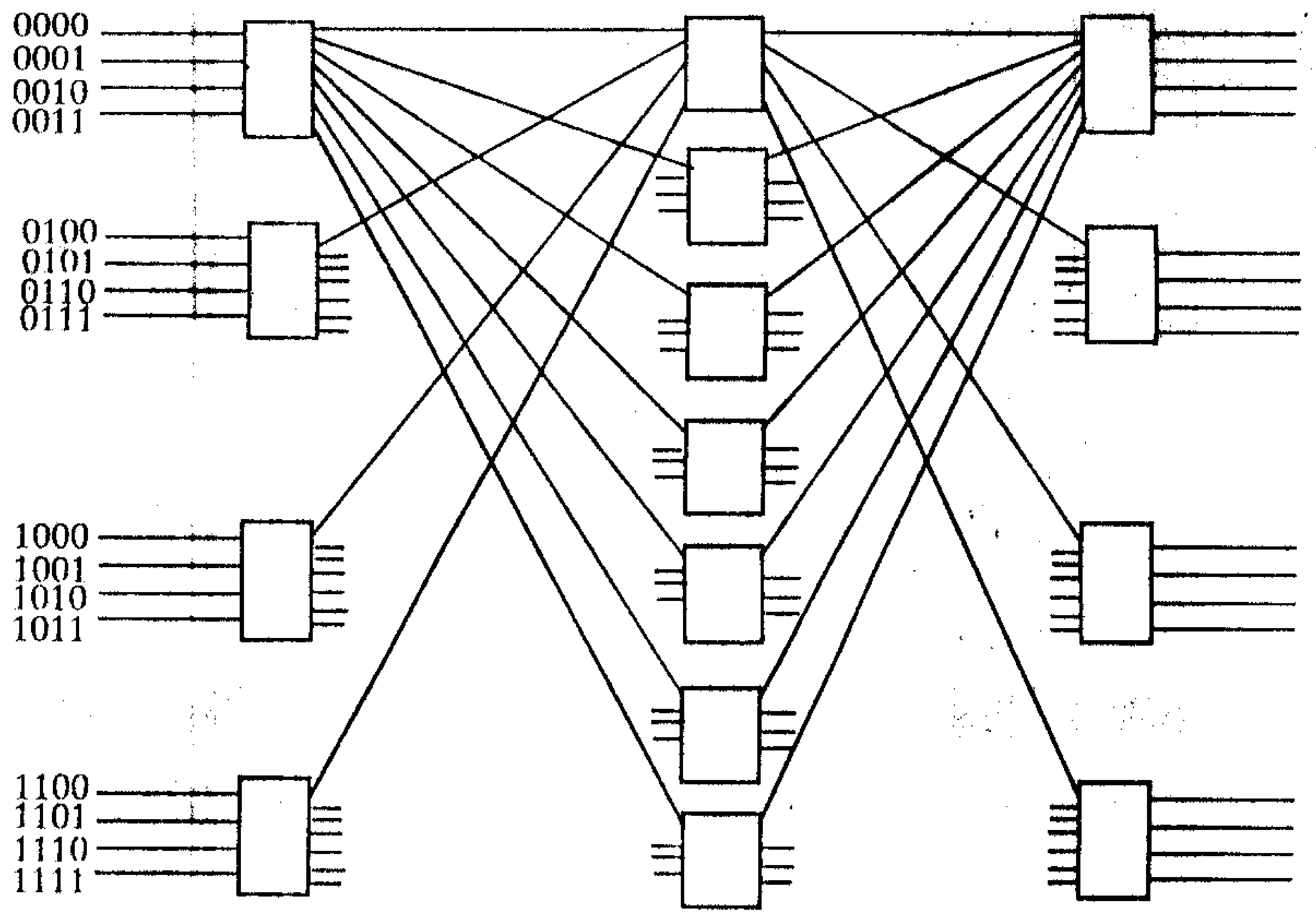


Fig. 6
Test vectors generated by algorithm E in a 16 × 16 3-stage Clos network

of test vector ($2 \lceil \log_2 (2n - 1) \rceil - 1) \lceil \log_2 N \rceil + 1$ is required.

3.8. Diagnosis of Link Short Circuit Faults

Algorithm G :

(1) Set the network control settings same as the phase in which faulty response was observed.
(2) Trace the faulty paths and mark them.
(3) Set the network shifing by plus one for input or middle stage switches and by minus one for output switches with same links on as step 1.
(4) Test the network by each test vector generated by algorithm E. If faulty response is found for some vector, test the network by its complementary test vector and go to step 5.
(5) If no faulty respanse found by step 4,go to step 3
else trace the faulty paths and mark them.
(6) Find the intersecting links.

Lemma 4 : For ~~a~~ an N×N 3-stage Clos network at most $2 \lceil \log_2 N \rceil + 1$ extra test vectors are required for diagnosis of fault.

Proof : By three completely different control settings with same links off as in straight mode the fault will be visible and the faulty paths will be intersecting. Among them one type of control setting has been used during fault detection.
In worst case testing after shifting by one may not give any faulty output, thus total $2 \lceil \log_2 N \rceil + 1$ test vectors are required.

3.9. Multiple Switching Fault Diagnosis

If there is a multiple fault in the network then during running of the algorithm C faulty paths may

converse to (a) different switch, or (b) different link, or (c) different input of same switch.
In each of the above three cases treat them as different faults and for each fault seperate the faulty paths and faulty outputs of the faulty switch. Then run step 3 to 6 of algorithm C for each fault. Thus fault diagnosis can be done.
[*Note :* If faulty paths converse to same input of same switch then they may not be seperated and diagnosed. But if the multiple fault is control stuck or control short circuit fault then they can be seperated and diagnosed by algorithm D only.]

3.10. Single multiple link short circuit fault diagnosis.

Follow the algorithm G only modifying step 6 as follows:
If faulty paths are not intersecting go to step 3, else find the faulty links.

CHAPTER 4

CONCLUSION

Fault diagnosis procedures for two-link short circuit fault and single switching faults have been developed. The upper bound for number of test vectors needed for detecting and diagnosis of them has been found. But diagnosis of multiple faults has not been done totally. Further work can be done on multiple faults.

After diagnosis of fault if it is seen that (1) fault is in input and output switch then they have to be replaced (2) if fault is at middle stage network can be used as rearrangeable network. According to the Slepian Duguid [B62] theorem a Clos Network N(n,r,m) can be used as rearrangeable network if $m \geq n$. If fault is at the middle stage switch, what will be the algorithm to use the network as rearrangeable network is not known yet. Further work can be done on it.

CHAPTER 5

BIBLIOGRAPHY

[C52] : Charles Clos, "A study of Non-blocking Networks,"*Bell System Technical Journal,* vol. 32, pp 406 - 424, March 1953 .

[B62] : V. E. Benes, "On Rearrangeable Three-stage connecting networks," *Bell System Technical Journal,* vol. XLI, no 5, pp 1481 - 1492, Sep. 1962 .

CHAPTER 6

APPENDIX

6.1. Simmulation Program for Diagnosis of Single switching fault

```
#include <stdio.h
#include <math.h>
void in_to_out (void);
void gen_alg_c(void);

struct inp {
       int inps[4];
       int cons[4];
       int outs[7] ;
         } inpt[4];
struct mid {
       int inps[4] ;
       int cons[4];
       int outs[4] ;
         } midl[7];

struct oup {
       int inps[7];
       int cons[4];
       int outs[4];
         } oupt[4];

int fault[20][6];

int kk0,k2,k3,kk3,kk5,kk6,kk7,kk8,kk9,fault1,fault2,fault3;

main ()
{
int j,i,k,m;
kk5 = -1;
k3=0;

.........................................................
.                        Algorithm B                    .
.........................................................

for (i=0;i<4;i++)
  for (j=1;j<4;j++) {
        oupt[i].cons[j] = inpt[i].cons[j] = 2*j-1;
        midl[2*j-1].cons[i] = i;
        midl[2*j].cons[i] = -1;
        }
for (i=0;i<4;i++) {
        oupt[i].cons[0] = inpt[i].cons[0] = 0;
        midl[0].cons[i] = i;
        }
kk3=0;
gen_alg_c();
for (i=0;i<3;i++) {
        for (k=0;k<4;k++)
          for (j=0;j<4;j++) {
              if (inpt[k].cons[j]==0) {
                                     inpt[k].cons[j] =1;
```

```
                                        oupt[k].cons[j] =1;
                                        }
                else {
                        inpt[k].cons[j] =(inpt[k].cons[j] +2 )%7;
                        oupt[k].cons[j] =(oupt[k].cons[j]+2)%7;
                     }
                if (k==0) midl[k].cons[j] =(midl[k].cons[j]+1)%4;
                else midl[2*k-1].cons[j]
                        =(midl[2*k-1].cons[j]+1)%4;
                        }
                kk3=kk3+1;
                gen_alg_c();
                }
for (i=0;i<4;i++)
  for (j=0;j<4;j++) {
        oupt[i].cons[j] = inpt[i].cons[j] = 2*j;
        midl[2*j].cons[i] = i;
        if (j!=0) midl[2*j - 1].cons[i] = -1;
        }
kk3=0;
gen_alg_c();
for (i=0;i<3;i++) {
        for (k=0;k<4;k++)
          for (j=0;j<4;j++) {
                inpt[k].cons[j] =(inpt[k].cons[j]+2)%8;
                midl[2*k].cons[j] =(midl[2*k].cons[j]+1)%4;
                oupt[k].cons[j] =(oupt[k].cons[j]+2)%8;
                        }
                kk3=kk3+1;
                gen_alg_c();
                }

***********************************************************
*                   Algorithm C and D                     *
***********************************************************

        if (k3!=1) printf("It is a multiple fault\n");
        else  {

        kk6=kk7=kk8=0;
                for (j=0; j<=kk5;j++) if (kk6==0) {
                        if (fault[0][1]!=fault[j][1]) kk6 =1;
                                }
                for (j=0; j<=kk5;j++) if (kk8==0) {
                        if (fault[0][3]!=fault[j][3]) kk8 =1;
                                }
                if ((kk6 == 0) && (kk8 ==0))  fault2 = -1;
                else if (kk6==0) fault2=0;
                else if (kk8==0) fault2=2;
                else  fault2=1;
        if ((fault2==0) || (fault2==2)) {
                        for (j=1; j<=kk5;j++) if (kk7==0) {
                        if (fault[0][2*fault2]!=fault[j][2*fault2]) kk7 =1;}
                if (kk7==0) {
                   if (kk5==7) printf("It is  input stuck fault at stage no\
%d switch no %d and io_no %d\n",fault2,fault[0][fault2+1],fault[0][2*fault2])
                   else if (kk5==2) {
                                fault1=fault[0][2];
                                for (i=1;i<=kk5;i++) fault1=fault1&fault[i][2
                if (fault1 == 1)
        printf("It is control stuck at 0 fault at stage \
        %d switch no %d and i/o_no %d and at control bit\
        no 0\n",fault2,fault[0][fault2+1],fault[0][2*fault2]);
                else if (fault1 == 2)
        printf("It is control stuck at 0 fault at stage \
        %d switch no %d and i/o_no %d and at control bit\
        no 1\n",fault2,fault[0][fault2+1],fault[0][2*fault2]);
                else if (fault1 == 4)
        printf("It is control stuck at 0 fault at stage \
        %d switch no %d and i/o_no %d and at control bit\
        no 2\n",fault2,fault[0][fault2+1],fault[0][2*fault2]);
                else printf("It is a multiple fault\n");
        }
   else if (kk5 ==4) {
                fault1=fault[0][2];
                for (i=1;i<=kk5;i++) fault1=fault1|fault[i][2]
                if (fault1 == 6)
        printf("It is control stuck at 1 fault at stage \
        %d switch no %d and i/o_no %d and at control bit\
        no 0\n",fault2,fault[0][fault2+1],fault[0][2*fault2]);
                else if (fault1 == 5)
        printf("It is control stuck at 0 fault at stage \
        %d switch no %d and i/o_no %d and at control bit\
        no 1\n",fault2,fault[0][fault2+1],fault[0][2*fault2]);
                else if (fault1 == 3)
        printf("It is control stuck at 0 fault at stage \
        %d switch no %d and i/o_no %d and at control bit\
        no 2\n",fault2,fault[0][fault2+1],fault[0][2*fault2]);
                else printf("It is a multiple fault\n");
        }
   else if (kk5 ==3) {
                fault1=fault[0][2] ^ fault[1][2];
                if (fault1 == 6)
        printf("It is control short circuit fault between \
        control bits 1 & 2 at stage %d switch no %d and i/o_nc
        %d\n",fault2,fault[0][fault2+1],fault[0][2*fault2]);
                else if (fault1 == 4)
        printf("It is control short circuit fault between \
        control bits 0 & 1 at stage %d switch no %d and i/o_nc
        %d\n",fault2,fault[0][fault2+1],fault[0][2*fault2]);
                else if (fault1 == 2)
        printf("It is control short circuit fault between \
        control bits 0 & 2 at stage %d switch no %d and i/o_nc
        %d\n",fault2,fault[0][fault2+1],fault[0][2*fault2]);
                else printf("It is a multiple fault\n");
        }
        else printf("It is a multiple fault\n");
}
else  {
        kk6 = 0;
        for (j=0; j<=kk5;j++) if (kk6==0) {
                if (fault[0][2]!=fault[j][2]) kk6 =1;
                }
        if (kk6==0) {
                if (kk5==7) {
if (fault2==0)  printf("It is stage no 1 input stuck or\
        input stage output stuck fault at stage no 1 switch\
        no %d and input no %d and at input stage switch no %d
and output no %d \n",fault[0][2],fault[0][1],fault[0][1],fault[0][2]);
   else  printf("It is stage no 1 output stuck or\
                stage no 2 input stuck fault at stage no 1 switch\
                no %d and output no %d and at stage no 2 switch no %d\
and input no %d \n",fault[0][2],fault[0][3],fault[0][3],fault[0][2]);
                                }
                else if ((fault2==0) && ((kk5 ==1) || (kk5==3))) {
                                fault1 = fault[0][3];
                                for (i=0;i<=kk5;i++)
                                 fault1=fault1 & fault[i][3];
                                fault2 = fault[0][3];
                                for (i=0;i<=kk5;i++)
                                 fault2=fault2 | fault[i][3];
                                if (fault1!=0) {
                        if (fault1 == 1)
                printf("It is control stuck at 0 fault at stage no 1\
                switch no %d and input no %d and at control bit\
                no 0\n",fault[0][2],fault[0][1]);
                        else if (fault1 == 2)
                printf("It is control stuck at 0 fault at stage no 1\
                switch no %d and input no %d and at control bit\
                no 1\n",fault[0][2],fault[0][1]);
                                        }
                          else  if (fault2!=3) {
                        if (fault2 == 2)
                printf("It is control stuck at 1 fault at stage no 1\
                switch no %d and input no %d and at control bit\
                no 0\n",fault[0][2],fault[0][1]);
                        else if (fault1 == 1)
                printf("It is control stuck at 1 fault at stage no 1\
                switch no %d and input no %d and at control bit\
                no 1\n",fault[0][2],fault[0][1]);
                                        }
                        else printf("It is control short circuit at\
stage no 1 switch no %d input no %d\n",fault[0][2],fault[0][1]);
                                }
                        }
                else  printf("It is stage no 1 output stuck or\
                stage no 2 input stuck fault at stage no 1 switch\
                no %d and output no %d and at stage no 2 switch no %d
and input no %d \n",fault[0][2],fault[0][3],fault[0][3],fault[0][2]);
        }
        }
else if (fault2== -1) {
                kk9=0;
                if (kk5==1) {
                        if ((fault[0][0]!=fault[1][0]) ||
        (fault[0][4]!=fault[1][4]) || (fault[0][2]!=fault[1][2])) kk9=
                }
                if (kk9==0) {
                        kk8=fault[0][2]/2;kk8=kk8*2;
                        if (kk8==fault[0][2]) {
                                kk3=(fault[0][3] - fault[0][1] +4)%4;
                                for (i=0;i<4;i++) for (j=0;j<4;j++) {
                        oupt[i].cons[j]=inpt[i].cons[j]=(2*(j+kk3))%8;
                                mid1[2*j].cons[i]=(i+kk3+1)%4;
                                }
                                fault1=kk5;
                                gen_alg_c();
```

```
                                        if (fault1==kk5)
                         printf("It is inter link stuck fault at stage no 1\
switch no %d input no %d output no %d\n",fault[0][2],fault[0][1],fault[0][3]);
                                        else if (fault[0][3]==fault[kk5][3])
                         printf("It is inter link stuck fault at stage no 2\
switch no %d input no %d output no %d\n",fault[0][3],fault[0][2],fault[0][4]);
                                        else if (fault[0][1]==fault[kk5][1])
                         printf("It is inter link stuck fault at stage no 0\
switch no %d input no %d output no %d\n",fault[0][1],fault[0][0],fault[0][2]);
                                else printf("It is a multiple fault\n");
                                }
                                else  {
                                        kk3=(fault[0][3] - fault[0][1])%4;
                                for (i=0;i<4;i++)  for (j=1;j<4;j++) {
                                oupt[i].cons[j] = inpt[i].cons[j] = 2*j-1;
                                midl[2*j-1].cons[i] = (i+1)%4;
                                        }
                                for (i=0;i<4;i++) {
                                oupt[i].cons[0] = inpt[i].cons[0] = 0;
                                        midl[0].cons[i] =(i+1)%4;
                                                }
                                for (i=0;i<kk3;i++) {
                                for (k=0;k<4;k++) for (j=0;j<4;j++) {
                                if (inpt[k].cons[j]==0) {
                                        inpt[k].cons[j] =1;
                                        oupt[k].cons[j] =1;
                                        }
                                else {
                                  inpt[k].cons[j] =(inpt[k].cons[j] +2 )%7;
                                  oupt[k].cons[j] =(oupt[k].cons[j]+2)%7;
                                }
                         if (k==0) midl[k].cons[j] =(midl[k].cons[j]+1)%4;
                         else midl[2*k-1].cons[j] =(midl[2*k-1].cons[j]+1)%4;
                                }
                                }
                                        fault1=kk5;
                                        gen_alg_c();
                                        if (fault1==kk5)
                          printf("It is inter link stuck fault at stage no 1\
switch no %d input no %d output no %d\n",fault[0][2],fault[0][1],fault[0][3]);
                                        else if (fault[0][3]==fault[kk5][3])
                          printf("It is inter link stuck fault at stage no 2\
switch no %d input no %d output no %d\n",fault[0][3],fault[0][2],fault[0][4]);
                                        else if (fault[0][1]==fault[kk5][1])
                          printf("It is inter link stuck fault at stage no 0\
switch no %d input no %d output no %d\n",fault[0][1],fault[0][0],fault[0][2]);
                                else printf("It is a multiple fault\n");
                                }
                         }
                else printf("It is a multiple fault\n");
                }
        else if (k3==0) printf("It is not a faulty network\n");
                else printf("It is a multiple fault\n");
        }
}
.............................................................................
```

```
void in_to_out ()
{
int ii,jj,kk,kk1,kh;
.............................................................................
                  Implementation of Fault
.............................................................................
kh=inpt[0].cons[1];
inpt[0].cons[1]= (inpt[0].cons[1] | 02) ;
/*inpt[1].inps[1]=0;
kh=midl[0].cons[1];
midl[0].cons[1]= midl[0].cons[1] | 01;*/
.............................................................................
for (ii=0;ii<4;ii++)
  for (jj=0;jj<4;jj++) {
                   kk=inpt[ii].cons[jj];
                   if (kk!=7) {
                   midl[kk].inps[ii]= inpt[ii].outs[kk] = inpt[ii].inps[jj];
                   kk1=midl[kk].cons[ii];
                   if (kk1!= -1) midl[kk].outs[kk1] = midl[kk].inps[ii];
                   }
                   }
for (ii=0;ii<4;ii++)
  for (jj=0;jj<4;jj++) {
                   if (oupt[ii].cons[jj] != 7)
                   oupt[ii].outs[jj] = midl[oupt[ii].cons[jj]].outs[ii];
                   }
inpt[0].cons[1] = kh;
/*oupt[1].outs[1]=0;*/
/*midl[0].cons[1] = kh;*/
}


.............................................................
.              Algorithm A and Checking of Outputs          .
.............................................................

void gen_alg_c()
{
int iii,jjj;
k2 = 0;
for (iii=0;iii<4;iii++)
        for (jjj=0;jjj<4;jjj++) inpt[iii].inps[jjj]=0;
  in_to_out();
for (iii=0;iii<4;iii++)
        for (jjj=0;jjj<4;jjj++) if (oupt[iii].outs[jjj]!=0)   {
                            kk5 = kk5 + 1;
                            k2=k2+1;
                            fault[kk5][3] = iii;
                            fault[kk5][2] = oupt[iii].cons[jjj];
                            fault[kk5][1] = (iii - kk3 +4)%4;
                            fault[kk5][0] = fault[kk5][4] =jjj;
                            fault[kk5][5] = 1;
                   }
for (iii=0;iii<4;iii++)
        for (jjj=0;jjj<4;jjj++) inpt[iii].inps[jjj]=1;
  in_to_out();
for (iii=0;iii<4;iii++)
        for (jjj=0;jjj<4;jjj++) if (oupt[iii].outs[jjj]!=1)   {
                            kk5 = kk5 + 1;
                            k2=k2+1;
```

```
                        fault[kk5][3] = iii;
                        fault[kk5][2] = oupt[iii].cons[jjj];
                        fault[kk5][1] = (iii - kk3 + 4)%4;
                        fault[kk5][0] = fault[kk5][4] =jjj;
                        fault[kk5][5] = 0;

if (k2==2) k3=2;
else if ((k2==1) && (k3!=2)) k3 = 1;
```

Result:
It is control stuck at 0 fault at stage 0 switch no 0 and i/o_no 1 and at
control bit no 1

6.2. Simmulation Program for Diagnosis of Single Two-link short circuit fault

```
#include <stdio.h>
#include <math.h>
void in_to_out (void);
void gen_alg_a(void);
int check_fault( int faulty);

struct inp {
     short inps[4];
     short cons[4];
     short outs[7] ;
       } inpt[4];
struct mid {
     short inps[4] ;
     short cons[4];
     short outs[4] ;
       } midl[7];

struct oup {
     short inps[7];
     short cons[4];
     short outs[4];
       } oupt[4];

fault[5][5];

int ph,kk3,kk5,fault1,fault2,fault3,fault4,fault5,fault6;

main ()
{
short j,i,k,m,ko,ki;
kk5 = -1;
.........................................................
.                          Algorithm E                  .
.........................................................
for (i=0;i<4;i++)
  for (j=0;j<4;j++) {
        oupt[i].cons[j] = inpt[i].cons[j] = 2*j;
        midl[2*j].cons[i] = i;
        if (j!=0) midl[2*j -1].cons[i] = -1;
        }
ph = 0;
```

```
gen_alg_a();
if (kk5 == -1) {
for (i=0;i<4;i++)
  for (j=1;j<4;j++) {
        oupt[i].cons[(j+1)%4] = inpt[i].cons[j-1] = 2*j-1;
        midl[2*j-1].cons[i] = (i+1)%4;
        midl[2*j].cons[i] = -1;
        }
for (i=0;i<4;i++) {
        oupt[i].cons[1] = inpt[i].cons[3] = 0;
        midl[0].cons[i] = (i+1)%4;
        }
ph = 1;
gen_alg_a();
if (kk5 == -1) {
for (i=0;i<4;i++)
  for (j=1;j<4;j++) {
        oupt[i].cons[(j+1)%4] = inpt[i].cons[j-1] = 2*j;
        midl[2*j].cons[i] = (i+1)%4;
        midl[2*j-1].cons[i] = -1;
        }
for (i=0;i<4;i++) {
        oupt[i].cons[1] = inpt[i].cons[3] = 1;
        midl[1].cons[i] = (i+1)%4;
        midl[0].cons[i] = -1;
        }
ph = 1;
gen_alg_a();
if (kk5 == -1) {
for (i=0;i<4;i++) {
        oupt[i].cons[3] = inpt[i].cons[3] = 5;
        midl[5].cons[i] = i;
        midl[0].cons[i] = -1;
        midl[4].cons[i] = -1;
        midl[6].cons[i] = -1;
        }
for (i=0;i<4;i++)
 for (j=0;j<3;j++) {
        oupt[i].cons[j] = inpt[i].cons[j] = j+1;
        midl[j+1].cons[i] = i;
        }
ph = 0;
gen_alg_a();
if (kk5 == -1) {
for (i=0;i<3;i++)
        for (j=0;j<4;j++) midl[i].cons[j] = -1;

for (i=0;i<4;i++)
  for (j=0;j<4;j++) {
        oupt[i].cons[j] = inpt[i].cons[j] = j+3;
        midl[j+3].cons[i] = i;
        }
ph = 0;
gen_alg_a();
}
}
}
}
if (kk5== -1) printf("There is no fault\n");
```

```
     if (fault[1][fault1 -1]==fault[2][fault1 -1]) fault2 = fault1 -1;



lse {

*********************************************
*                        Algorithm G        *
*********************************************

for (k=0;k<2;k++) if (kk5 <3) {
       ph = ph + 1;
       ki = inpt[0].cons[1];
       ko = oupt[0].cons[3];
              for (i=0;i<4;i++)
              for (j=1;j<4;j++) {
                     inpt[i].cons[j] = inpt[i].cons[(j+1)%4];
                     oupt[i].cons[(4-j)] = oupt[i].cons[(3-j)];
                     if (midl[2*j].cons[i] != -1)
                     midl[2*j].cons[i] =(midl[2*j].cons[i] +1)%4;
                     if (midl[2*j-1].cons[i] != -1)
                     midl[2*j-1].cons[i] = (midl[2*j-1].cons[i] +1)%4;
                     }
              for (i=0;i<4;i++) {
                     inpt[i].cons[0] = ki;
                     oupt[i].cons[0] = ko;
                     if (midl[0].cons[i] != -1)
                     midl[0].cons[i] = (midl[0].cons[i] +1)%4;
                     }
       gen_alg_a();
       }
 fault1 = -1;
 for (i=1; i<4; i++) if (fault[0][i] == fault[2][i]) fault1 = i;
 if (fault1!= -1) {
       if (fault[0][fault1 -1]==fault[2][fault1 -1]) fault2=fault1 -1;
       else if (fault[0][fault1+1]==fault[2][fault1+1]) fault2=fault1+1;
       else fault1 = -1;
       }
 fault3 = -1;
 for (i=1; i<4; i++) if (fault[0][i] == fault[3][i]) fault3 = i;
 if (fault3!= -1) {
       if (fault[0][fault3 -1]==fault[3][fault3 -1]) fault4 = fault3 -1;
       else if (fault[0][fault3+1]==fault[3][fault3+1]) fault4=fault3+1;
       else fault3 = -1;
       }
 if ((kk5>3) || ((fault1== -1) && (fault3 == -1)))
              printf("It is a multiple fault\n");
 else if (fault1 == -1) {
         for (i=1; i<4; i++)
                if (fault[1][i] == fault[2][i]) fault1 = i;
         if (fault1!= -1) {
         if (fault[1][fault1 -1]==fault[2][fault1 -1]) fault2 = fault1 -1;
         else if (fault[1][fault1+1]==fault[2][fault1+1]) fault2=fault1+1;
         else fault1 = -1;
         }
         if (fault1== -1) printf("It is a multiple fault\n");
         else {
              if ((fault2>fault1) || (fault2!=2))
                     fault1=fault[1][fault1]*4 + fault[1][fault2];
              else fault1 = fault[1][fault1]*7 + fault[1][fault2];
              if ((fault4>fault3) || (fault4!=2))
                     fault3=fault[0][fault3]*4 + fault[0][fault4];
              else fault3 = fault[0][fault3]*7 + fault[0][fault4];
```

```
                printf("Fault is stage no %2d link no %2d and stage no %2d \
                link no %2d\n",fault2,fault1,fault4,fault3);
                }
            }
else if (fault3== -1)    {
            for (i=1; i<4; i++)
                    if (fault[1][i] == fault[3][i]) fault3 = i;
            if (fault1!= -1) {
            if (fault[1][fault3 -1]==fault[3][fault3 -1]) fault4 = fault3 -1;
            else if (fault[1][fault3+1]==fault[3][fault3+1]) fault4=fault3+1;
            else fault3 = -1;
            }
            if (fault3== -1) printf("It is a multiple fault\n");
            else {
                if ((fault2>fault1) || (fault2!=2))
                        fault1=fault[0][fault1]*4 + fault[0][fault2];
                else fault1 = fault[0][fault1]*7 + fault[0][fault2];
                if ((fault4>fault3) || (fault4!=2))
                        fault3=fault[1][fault3]*4 + fault[1][fault4];
                else fault3 = fault[1][fault3]*7 + fault[1][fault4];
                printf("Fault is stage no %2d link no %2d and stage no %2d \
                link no %2d\n",fault2,fault1,fault4,fault3);
                }
            }
else  {
            fault5 = -1;
            for (i=1; i<4; i++) if (fault[1][i] == fault[2][i]) fault5 = i;
            if (fault5!= -1) {
            if (fault[1][fault5 -1]==fault[2][fault5 -1]) fault6=fault5 -1;
            else if (fault[1][fault5+1]==fault[2][fault5+1]) fault6=fault5+1;
            else fault5 = -1;
            }
            if (fault5!= -1) {
                if ((fault6>fault5) || (fault6!=2))
                        fault5=fault[1][fault5]*4 + fault[1][fault6];
                else fault5 = fault[1][fault5]*7 + fault[1][fault6];
                if ((fault4>fault3) || (fault4!=2))
                        fault3=fault[0][fault3]*4 + fault[0][fault4];
                else fault3 = fault[0][fault3]*7 + fault[0][fault4];
                printf("Fault is stage no %2d link no %2d and stage no %2d \
                link no %2d\n",fault6,fault5,fault4,fault3);
                }

            else    {
              for (i=1; i<4; i++) if (fault[1][i] == fault[3][i]) fault5 = i;
              if (fault5!= -1) {
              if (fault[1][fault5 -1]==fault[2][fault5 -1]) fault6=fault5 -1;
              else if (fault[1][fault5+1]==fault[2][fault5+1]) fault6=fault5+1;
              else fault5 = -1;
              }
              if (fault5== -1) printf("It is a multiple fault\n");
              else {
                if ((fault6>fault5) || (fault6!=2))
                        fault5=fault[1][fault5]*4 + fault[1][fault6];
                else fault5 = fault[1][fault5]*7 + fault[1][fault6];
                if ((fault2>fault1) || (fault2!=2))
                        fault1=fault[0][fault1]*4 + fault[0][fault2];
                else fault1 = fault[0][fault1]*7 + fault[0][fault2];
                printf("Fault is stage no %2d link no %2d and stage no %2d
                    link no %2d\n",fault6,fault5,fault2,fault1);
                    }
              }
   }
}
}


void in_to_out ()
{
int ii,jj,kk,kk1,kh;
for (ii=0;ii<4;ii++)
  for (jj=0;jj<4;jj++) {
                    kk=inpt[ii].cons[jj];
                    if (kk!=7) {
                    midl[kk].inps[ii]= inpt[ii].outs[kk] = inpt[ii].inps[jj];
                    kk1=midl[kk].cons[ii];
                    if (kk1!= -1) midl[kk].outs[kk1] = midl[kk].inps[ii];
                    }
                    }
for (ii=0;ii<4;ii++)
  for (jj=0;jj<4;jj++) {
                    if (oupt[ii].cons[jj] != 7)
                    oupt[ii].outs[jj] = midl[oupt[ii].cons[jj]].outs[ii];
                    }
.......................................................................
                    Implementation of Fault
.......................................................................
if (oupt[1].outs[0] != oupt[2].outs[2])
        oupt[1].outs[0] = oupt[2].outs[2]=0;
.......................................................................
}


.......................................................................
.                            Algorithm E                              .
.......................................................................

void gen_alg_a()
{
short i,j,k,p,n,m,faulty;
n = 4;
faulty = 0;
for (i=n;i>0;i--){
  if (faulty==0) {
  for (k=0;k<4;k++)
    for (j=0;j<4;j++) {
        m = (4*k + j)/pow(2,(4-i));
        p = m/2;
        p = p*2;
        if (m == p)  inpt[k].inps[j] = 0;
        else inpt[k].inps[j] = 1;
        }
  in_to_out();
  faulty = check_fault(faulty);
  if (faulty==1) {
                    for (k=0;k<4;k++)
                    for (j=0;j<4;j++) {
                    if (inpt[k].inps[j] == 0) inpt[k].inps[j] =1;
```

```
                  else inpt[k].inps[j] = 0;
                  }
                  in_to_out();
                  faulty = check_fault(faulty);
   }
}
}
}
***************************************************************************

int check_fault( int faulty)
{
 int j,k;
  for (k=0;k<4;k++)
    for (j=0;j<4;j++) {
        if (oupt[(k+ph)%4].outs[(j+2*ph)%4]!=inpt[k].inps[j]) {
                  faulty = faulty+1;
                          kk5 = kk5 + 1;
                          fault[kk5][3] = (k+ph)%4;
                          fault[kk5][2] = oupt[(k+ph)%4].cons[(j+2*ph)%4];
                          fault[kk5][1] = k;
                          fault[kk5][0] = j;
                          fault[kk5][4] = (j+2*ph)%4;
                  }
            }
return faulty;
}
```

Result:

```
Fault is stage no  4 link no  4 and stage no  4 link no 10
```