A Modular Approach to Fault-Tolerant Binary Tree Architectures

by

A. S. Mahmudul Hassan

A thesis submitted to the Faculty of Graduate Studies

and Research in partial fulfillment of the

requirements for the degree of

Master of Engineering

Department of Electrical Engineering

McGill University

Montréal

November 1984

Abstract

The binary tree architecture has been widely used in various VLSI (Very Large Scale Integration) implementations. A new modular, fault-tolerant scheme is proposed for the binary tree architecture. The approach uses modular fault-tolerant building blocks to construct the complete binary tree. Each module consists of a spare node and some redundant links to provide fault-tolerance. The faults are controlled locally within the modules. An optimal $O(n)$ area layout scheme is proposed for an 'n' node modular structure. Partitioning of the complete binary tree among several chips is shown to be very convenient due to the modularity of the structure. The restructuring operation, after each fault-occurrence, is very simple and local to each faulty module. The results of comparison show that the proposed fault-tolerant scheme is more reliable, modular, and easier to implement than the existing fault-tolerant schemes.

Résumé

Les structures en arbre binaire sont utilisées dans plusieurs circuits intégrés à haute performance. Une nouvelle approche est proposée pour permettre à un circuit avec une architecture en arbre binaire de fonctionner même en présence de defectuosités. Cette approche est modulaire et utilise des modules intrinsequement tolerant aux defauts pour construire l'arbre binaire complet. Chaque module comprend un noeud de reserve et plusieurs liens additionels pour permettre un comportement correct même en presence de défauts. Les défauts sont détectés localement dans chaque module. Une topologie régulière pour disposer un abre de n noeuds sur une surface $O(n)$ est suggérée. La structure très regulière de l'approche proposée permet de répartir un arbre complet sur plusieurs circuits intégrés, reliés ensemble. Lorsqu'une erreur est détectée, l'opération de réstructuration qui s'ensuit est très simple et se fait localement dans chaque module. Les resultats obtenus montrent que cette approche est plus fiable et plus facile à réaliser que les approches existantes.

Acknowledgements

The author wishes to express his heartfelt gratitude to Professor V. K Agarwal for his excellent guidance and supervision during the entire preparation of this thesis. Appreciation is also due to Arun K. Somani for his advice and cooperation.

The author is also grateful to the Canadian Commonwealth Scholarship and Fellowship Committee for providing financial assistance.

Table of Contents

List of Illustrations

List of Tables

1. Introduction

The binary tree is a very useful architecture for various VLSI (Very Large Scale Integration) implementations. Networks of hierarchical computing systems and Back-End Storage Networks are efficiently designed using this architecture. Basically, a certain number of nodes (depending upon the depth of the tree) are connected by communication links in a fashion to form the binary tree structure. The contents of a node depends upon the designed network A node can be a specialized computer interconnected with some other computers and exchanging information with the others or it can be a storage site for a data storage network.

One distinct advantage of the tree architecture is the faster speed of information exchange. For an 'n' element binary tree the depth or number of levels in the tree is $O(log_2 n)$. The maximum communication distance between two elements in this tree is $2log_2 n$, which is the diameter of the tree. Thus. the speed at which a binary tree can communicate is $O(log_2 n)$, which is much faster than $O(n)$ communication speed for a linear array

Symmetrical hierarchical binary tree system has widely been used in the design of multiprocessor systems in which each node of the tree represents a processing site or a site from which input or output (I/O) might take place. Some *dictionary machines* and *priority queues* have been designed using this multiprocessing binary tree architecture. Bently and Kung designed a simple but efficient dictionary machine using a binary tree with the data elements residing in the lowest level of the tree. Ottmann et al. [8] describe an improved machine with optimal performance using the X-tree architecture. Independently, Atallah and Kosaraju [1] designed another dictionary machine

with optimal performance using the pure binary tree architecture The pure binary tree architecture has also been used by Somani and Agarwal [2] to design an optimal performance but easy to implement unsorted dictionary machine

As computer systems become more decentralized, there is a greater need to share resources that are either too specialized or too costly to replicate This need has led to the development of specialized *Back-End* processing systems which provide functions that are in a sense optimized and that may be shared by several users Multiprocessing systems like X-Tree and Hypertree are good examples of such systems

Grey et al. [1] propose a Back-End Storage Network (BSN) which uses the binary tree structure. Each node of the tree is treated as a storage site or facility which is interconnected with high-speed communication paths represented by the links of the tree.

These various uses of the binary tree architecture have already proved its worth as a structure However, system reliability is a very important criteria to judge the efficiency of a system It is desired that an efficient system be sufficiently reliable and available as well But, one major weakness of the binary tree architecture is that the operation of different processors and links are very much interdependent Thus the failure of a single processor or a link may invalidate the operation of the whole tree That is why it has been realized that some form of fault-tolerance, a proven design methodology for achieving high reliability should be incorporated into the binary tree architecture to make the designed system reasonably reliable and available

There exist two different fault-tolerance approaches for designing binary tree machines. One approach allows some degradation in performance of the tree system in

the presence of faults. This assumes the system to be repairable However, during the period of outages the rest of the tree is available Some additional links are connected such that the non-faulty nodes remain interconnected in the presence of faults Such schemes are presented in 4 and 9 .

In the second approach the rigid tree structure is maintained even in the presence of failures Redundant nodes and links are provided to replace the faulty nodes and rebuild the complete binary tree Maintaining a rigid structure is a must for multiprocessing networks like dictionary machine For this reason, several papers have considered this approach of designing a fault-tolerant binary tree. Those schemes are reviewed in the next chapter

This thesis proposes a fault-tolerant scheme of designing a binary tree using the rigid tree structure approach The uniqueness of this scheme is the modular structure which is perfectly suited for VLSI design methodology. A fault-tolerant module or building block is designed which can absorb or tolerate a single-fault present in it. Repetition of the module and interconnection give a complete binary tree

The module consists of a number of processors connected by links Since each module can tolerate only a single fault whenever a fault occurs in a module the module must be restructured replacing the faulty part The tree can tolerate multiple failures as long as no module has more than one failure

The modular approach has certain advantages over the existing schemes Due to the regularly structured design an optimal H-tree like layout is obtained for the design An efficient chip partitioning scheme is also the outcome of the modular structure The restructuring scheme is local and reasonably simple The redundancy in links

is much less compared to the existing schemes Reliability of the system,using the new approach, improved significantly compared to the non-redundant system and other existing redundant schemes

The following chapters describe this modular binary tree architecture in details Chapter 2 gives an overview of the existing fault-tolerant schemes We describe the modular approach topologically in chapter 3 Chapter 4 describes the system structurally in terms of layout , restructuring etc In chapter 5, the system is evaluated with the help of probabilistic modeling and compared with RAE (Raghavendra, Avizienis and Ercegovac) fault-tolerant scheme presented in 9

2. Review of Literature

This chapter offers a brief review of the existing fault-tolerant schemes before going into the details of the modular scheme in the following chapters. The first section deals with some fault-tolerant schemes which are related to, but not designed to be, multiple fault-tolerant binary tree architectures. However, the three sections following that describe three different schemes for designing fault-tolerant binary tree architecture.

2.1 Related Schemes

In [5] Hayes has designed a fault-tolerant system such that it requires the minimum number of spare nodes and redundant links for tolerating a single failure in the system. Another study by Kwan and Toida in [6] considers the design of optimal k-fault tolerant system. They have extended the work in [5] and studied procedures for designing optimal fault-tolerant systems for a class of hierarchical tree structures.

Another interesting paper in this regard is [11] on multiprocessing cell architecture suitable for VLSI implementation, based on De Bruijn graphs. Although not directly connected to fault-tolerant binary tree problem, De Bruijn graph can form a fault-tolerant binary tree. In general, the De Bruijn graph has $N = d^k$ nodes with diameter k and degree 2d, and corresponds to the state-graph of a shift register of length k using d-ary digits. A shift register changes a state by adding a digit to one side of the state number and then by deleting one digit from the other side. Thus, De Bruijn graphs are also referred to as shift-and-replace graphs. An 8-node De Bruijn graph with d=2 is shown in Fig. 2.1. This can form a three-level (seven node) binary tree with any node or link going faulty. Thus, De Bruijn graph also gives a one-fault tolerant binary tree.

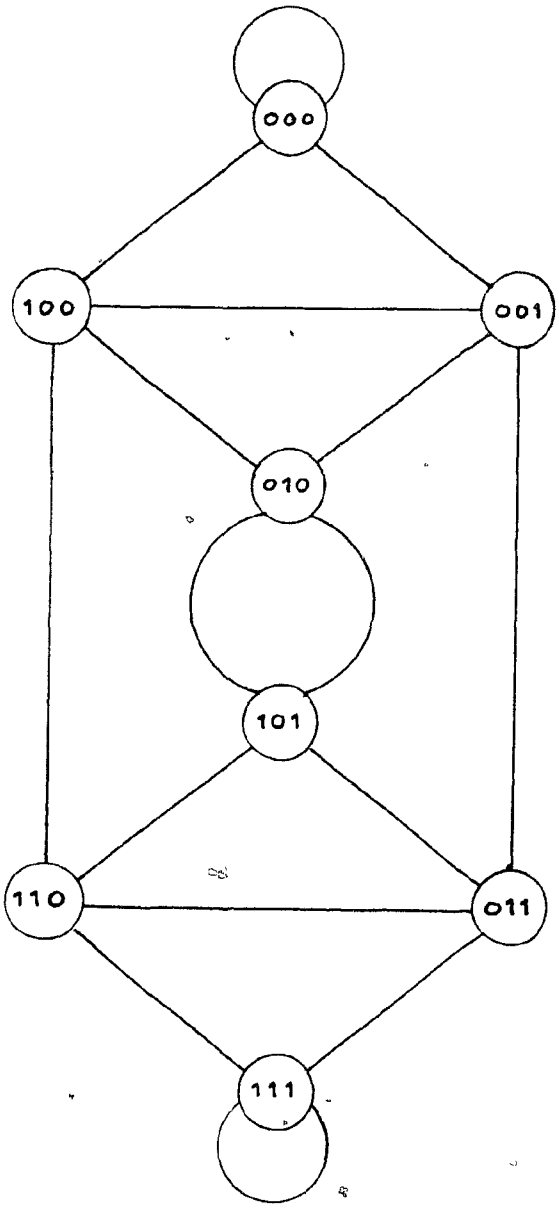

Fig. 2.1 The 8-node De Bruijn graph.

Leiserson, in [7], proposed an $O(nlog^2\ n)$ area layout for a chip which can implement any binary tree of 'n' vertices by simply adding n solder dots. The scheme is shown in Fig. 2.2. Instead of using solder dots, switches can be used at the intersection of the vertical and horizontal wires to form a restructurable fault-tolerant binary tree.

2.2 Grey et al. Scheme

In [4] Grey et al. proposed a fault-tolerant architecture for designing Back-End Storage Networks (BSN) based on hierarchical tree systems. The basic topology is based on a binary tree system in which additional bidirectional *cross* links have been placed to make the resulting topology one-fault tolerant, as shown in Fig. 2.3. In this design, a system will remain operating as long as all the non-faulty nodes are reachable through the links. It is not always necessary to maintain isomorphism. The approach is to provide connectivity in the presence of failed nodes. This is done by providing the cross links. The one-fault tolerance of the resulting topology is guaranteed by observing that every node is reachable in the presence of any single node failure.

The system also tolerates Double and Multiple node failures with some restrictions. It is shown that the fault-tolerant scheme is more reliable than the corresponding non-redundant tree system in all the cases of interest. Also, the architecture is extensible.

The proposed architecture is a performance degradation scheme. The design assumes some system degradation during periods of outages. Such an assumption is valid when we talk about networks like BSN. Because, each node of the tree is considered as a *Storage Site (SS)* which are interconnected with high-speed communication paths represented by the links of the tree. So, as long as the node connectivity is maintained,

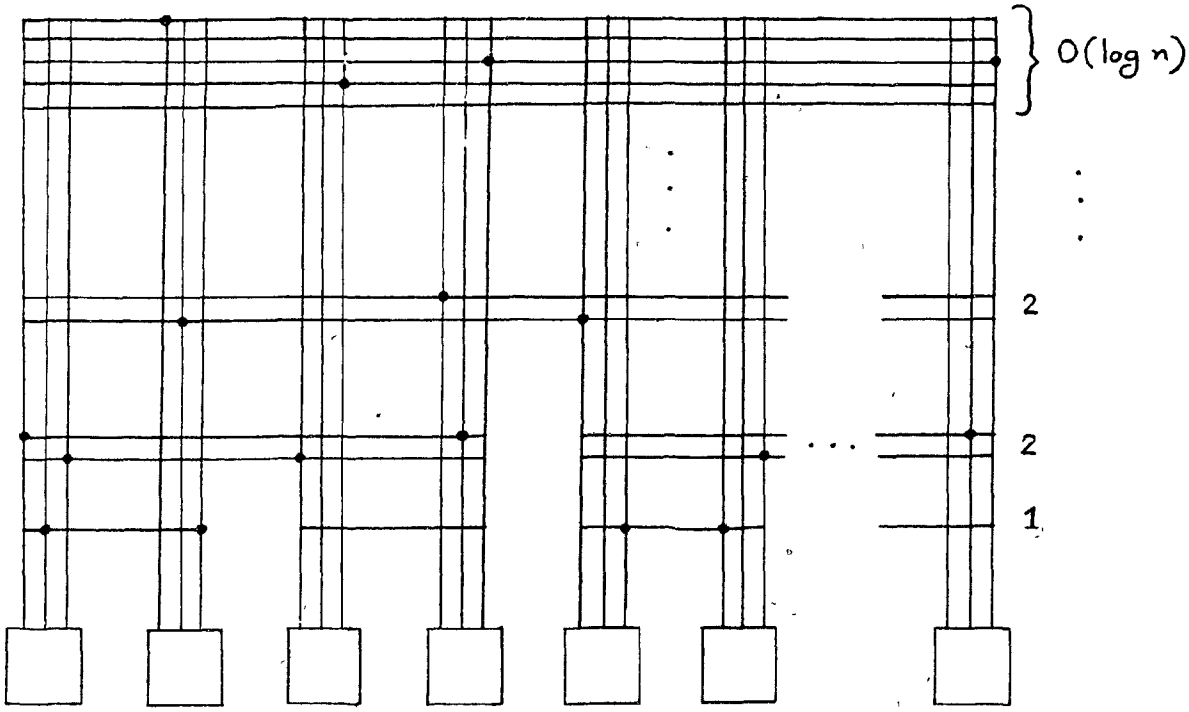

Fig. 2.2 $O(nlog^2n)$ area layout for a fault-tolerant binary tree.

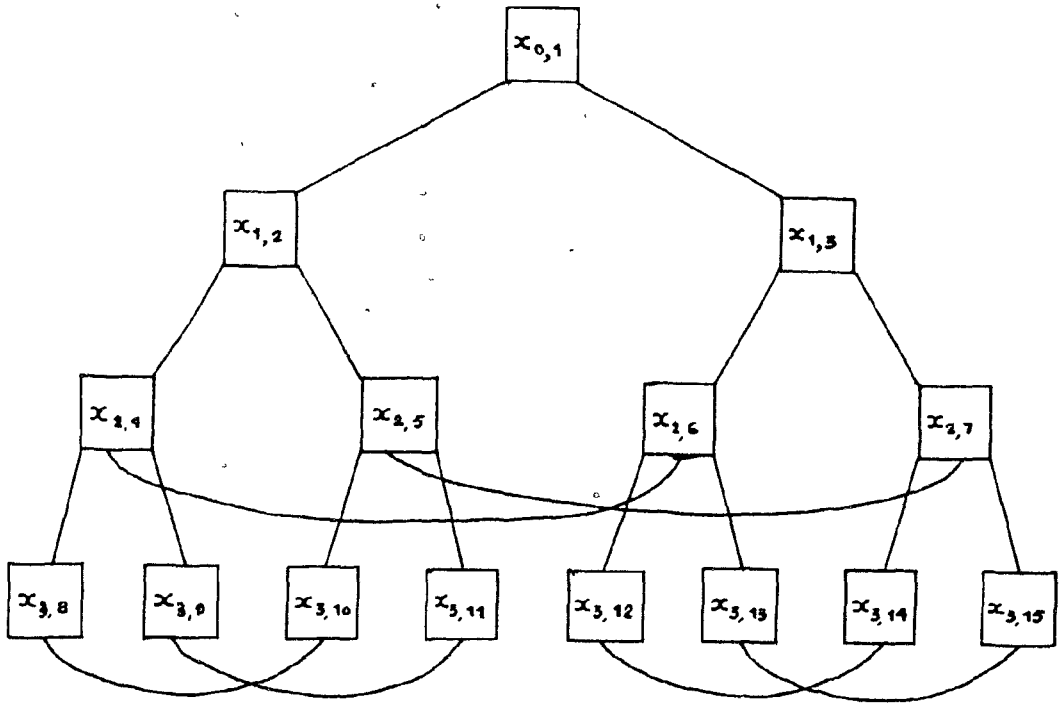

Fig. 2.3 Grey et al. fault-tolerant scheme.

required data can move properly although the rigid tree structure is non-existent

The hierarchical tree system is widely used in the design of multiprocessor systems. In such a system, each node of the tree represents a processing site. So, in these multiprocessing systems, fault-tolerance should be incorporated in such a manner that the rigid tree structure is maintained even in the presence of failures. However, Grey et al. scheme has not been designed to use as a multiprocessing network like dictionary machine, priority queue etc Therefore, its use is limited to the design of storage subsystems for distributed processing systems

Another drawback of the design is the root node In this topology, if the root node fails, the remaining tree is no more accessible through the links So, the root node should be well protected In a BSN the root node might be considered as a global controller for the entire BSN which is heavily fault protected through the application of dynamic redundancy

2.3 Diogenes Approach

[2] and [10] describe the Diogenes approach of designing a fault-tolerant binary tree. The essence of Diogenes layout strategy is to lay the processing elements (PEs) of the desired array out in a line, with some number of *bundles* of wires running above the line. However, for the array to be binary tree, there should be just one bundle Each faulty PE (determined by preliminary testing) would be "told" to connect 0 lines to the bundle (so that it would be just passed over), and each fault free PE would be "told" to take in 1 line, and to let out either 0 or 2 lines, thereby to act, respectively, as a leaf or an internal node of the tree It is the dynamic setting of the control switches that

lends the arrays their configurability, hence their tolerance to faults.

Fig 2.4 shows one cell of the Diogenes layout of a depth-3 complete binary tree. In this simple structure the bundle contains only three wires. In general, it is shown that the number of wires in the bundle never exceeds

$$\lfloor log_2(\text{the number of PEs fabricated}) \rfloor.$$

Thus, for 'n' PEs placed in a (logical) line, the depth of the stack should be $log_2 n$ So, the layout area for the 'n' node binary tree is $O(n log_2 n)$. This is optimal for collinear layouts of complete binary tree.

The Diogenes layout approach is optimal when the nodes of the binary tree are placed along a line But it is possible to do better by spreading the nodes throughout the chip in the H-tree method of layout. In fact, H-tree method is the optimal one for the binary tree layout. It requires $O(n)$ area for 'n' node binary tree. So, Diogenes layout is optimal for a collinear binary tree but not optimal compared to every fault-tolerant binary tree scheme.

Diogenes approach affords dynamic fault-tolerance in the tree structure. Whenever a fault occurs, the whole structure of the tree is changed. Restructuring begins from the right and proceeds towards the left in the array of PEs. Every restructuring gives a new set of fathers and sons in the tree. This can give rise to a problem in machines like *Atallah and Kosaraju Dictionary machine*[1] (or, Ottmann et al. machine '8'). In these machines the data is stored in a sorted order After restructuring, the nodes change their logical positions and the sorted order does not exist any more. Thus, the dictionary machines will not be able to operate with this unsorted data structure.

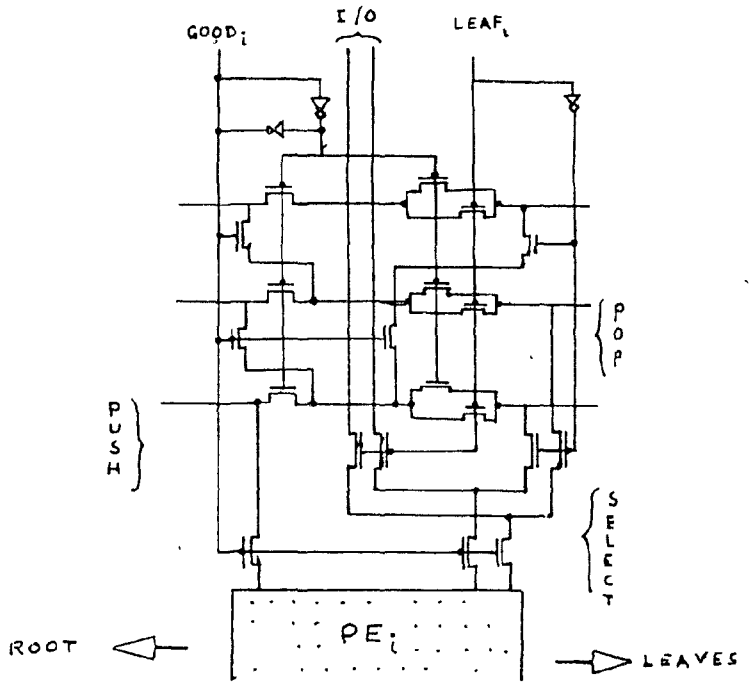

Fig. 2.4 One cell of a Diogenes layout

2.4 RAE Scheme

In 9 Raghavendra, Avizienis and Ercegovac suggested a fault-tolerant tree structure which can tolerate multiple failures with some restrictions. In fact, this scheme (referred to as RAE scheme in the following) is the main inspiration of the proposed design It has been referred to, time and again, throughout the discussion. The modular scheme is compared with RAE scheme in terms of layout, restructuring and system reliability

Raghavendra et al., in 9 , have proposed two different fault-tolerant mechanisms. One with spares and another one with performance degradation. The performance degradation scheme has some similarity with Grey et al. scheme described earlier in this chapter. This performance degradation scheme is shown in Fig 2.5 There is one spare node for the root and extra links from each node for redundancy When a node fails, its neighbor will take over its computation and thus will have to do more work. This type of fault-tolerance scheme is more suitable in systems where the communication is quite robust and the nodes are fairly powerful computer performing independent tasks

The scheme with spares is more interesting which assumes a fixed tree structure even in the presence of failures In this scheme the fault-tolerant tree contains one spare node per level, with redundant links for protection against any failure in that level. The logical structure of the tree is shown in Fig. 2 6 . When there is a failure, reconfiguration is done to maintain the logical structure of the binary tree This scheme tolerates several failures if those are in different levels of the tree

RAE have extended the scheme for higher protection against node failures. The technique is to provide one spare for every 2^i nodes, for some value of 'i'. The extended

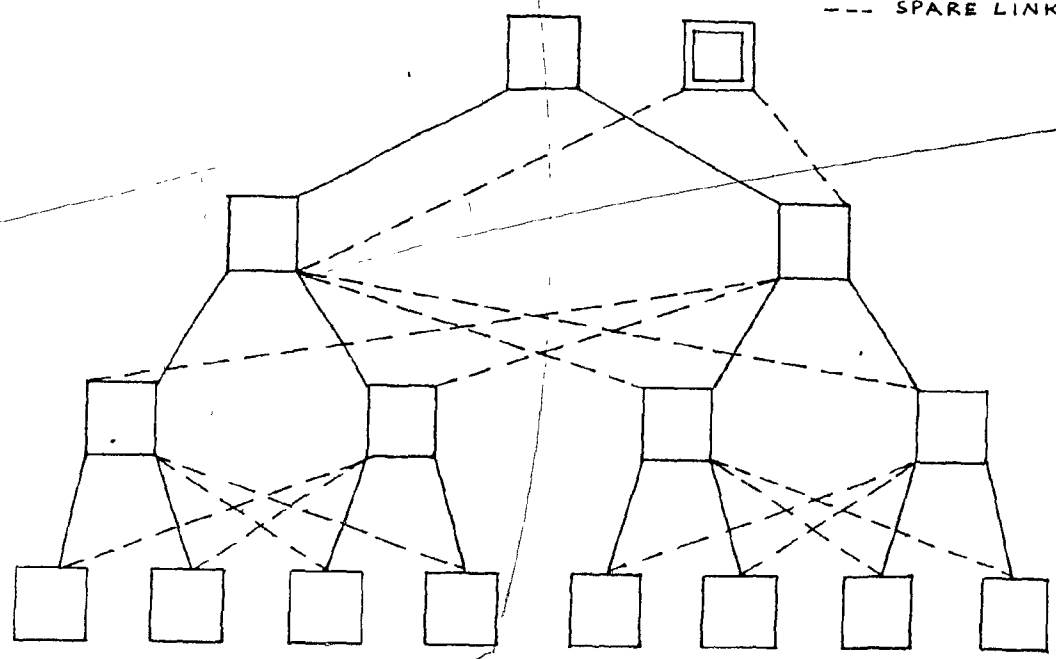

Fig. 2.5 RAE scheme with performance degradation.

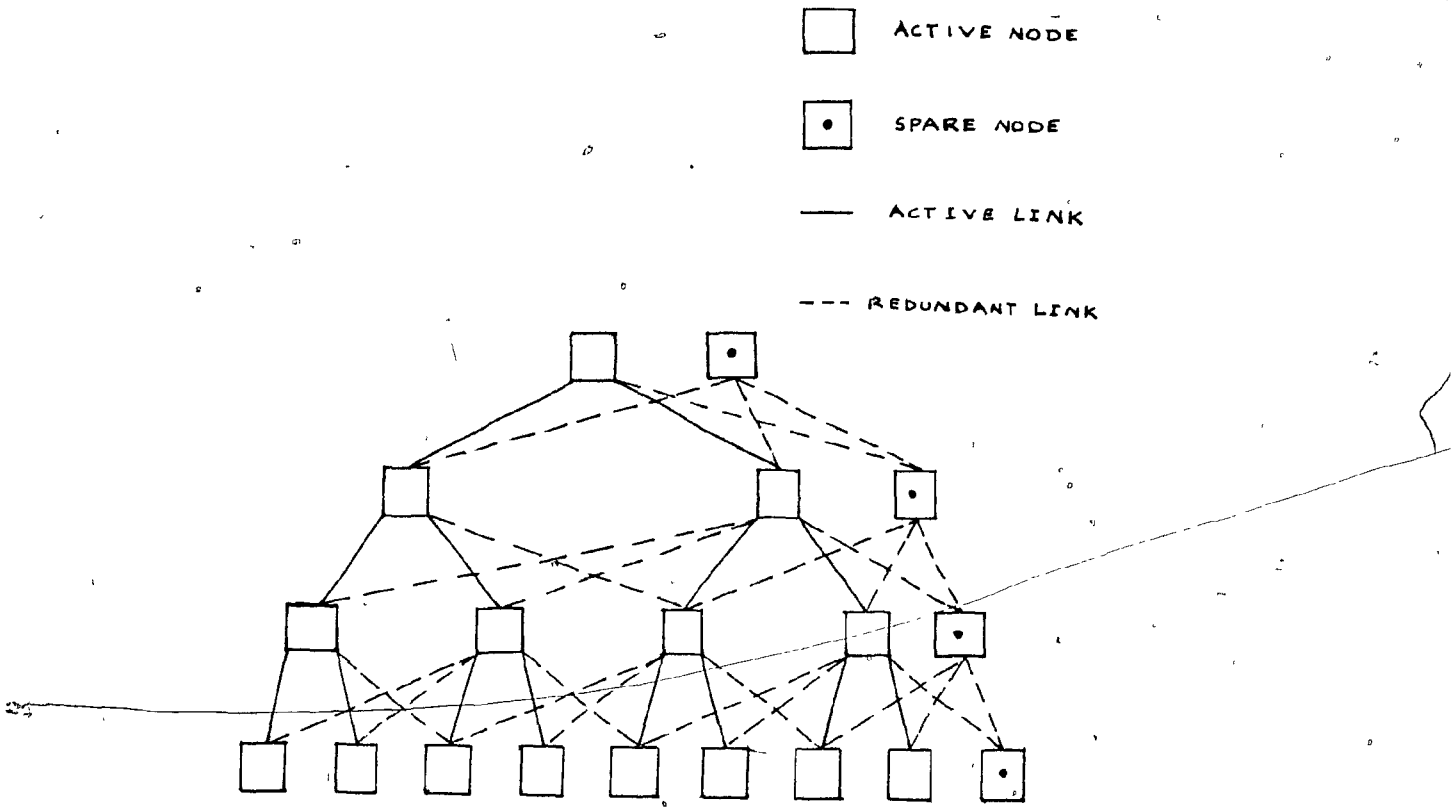

Fig. 2.6 RAE fault-tolerant scheme with spares.

fault-tolerant scheme with most spares will be that where there is a spare for every pair of nodes, which is shown Fig 2 7

RAE scheme uses a large number of redundant links for fault-tolerance A number of these links cross each other This clearly complicates the layout scheme for such a structure The layout would not be area efficient in terms of the number of nodes

The fact that a lot of links (at most eight) are connected to each node increases the node complexity However, Raghavendra et al have suggested the use of decoupling networks between the levels of the binary tree to lower down the node complexity But, decoupling networks, again, will increase the layout and routing complexity

In fact, the addition of a large number of redundant links makes the RAE structure less attractive from the VLSI point of view The scheme is not area-efficient due to the overlapping of the links. Also the tree cannot be conveniently partitioned among several chips using this scheme.

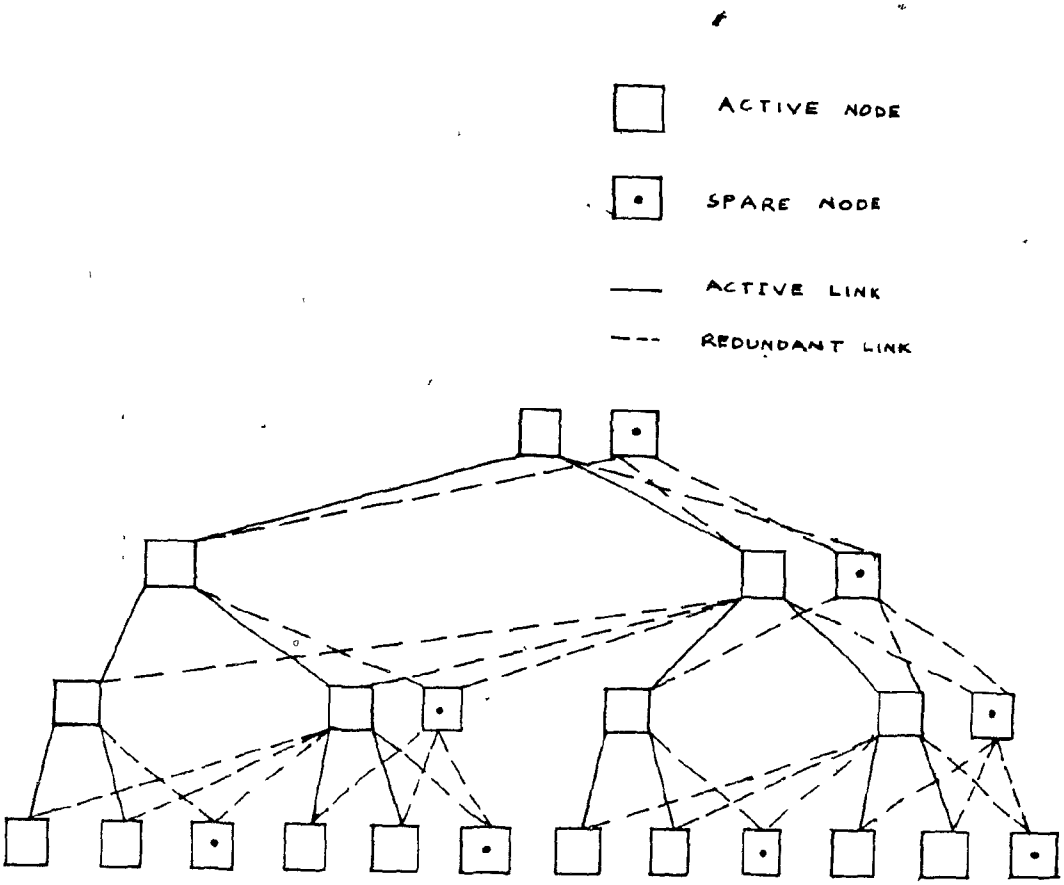

Fig. 2.7 RAE extended fault-tolerant scheme with spares.

3. The Module Topology

The proposed fault-tolerant binary tree architecture is based on a modular scheme. The topology of this modular scheme is described in this chapter. Sec. 3.1 describes the topology of the basic modular block. Module interconnection, which is necessary to form a complete binary tree, is discussed in Sec. 3.2. Sec. 3.3 describes some properties of the modular scheme and finally Sec. 3.4 gives some extensions of the scheme to form higher level modules.

3.1 The Basic Module

In general, a complete binary tree can be formed with some nodes connected by links. Nodes are the building blocks or units to form the binary tree. The basic idea of the proposed topology is to replace a group of nodes by a modular block. We will see later how these blocks are interconnected by links to form the complete binary tree.

The modular block is a fault-tolerant building block for the binary tree. It consists of some nodes connected by links to operate as a subtree of the complete binary tree. Also, some redundancy is incorporated in the module to make it fault-tolerant.

Fig. 3.1 shows the *basic modular block*. From now on, the term *module* will be used to mean the *basic modular block*, unless otherwise stated. The module consists of four nodes (processor) connected by links. The nodes are interconnected in such a fashion that at any time any one of these four nodes can go faulty and the remaining three nodes will form the active sub-tree. For example, when N2 (or N1) goes faulty, N3 and N4 (or N2) are the sons of N1. Similarly, for N3 to be faulty, N2 and N4 connect to N1. Finally, for faulty N1, N2 and N4 become the sons of N3. All these changes are

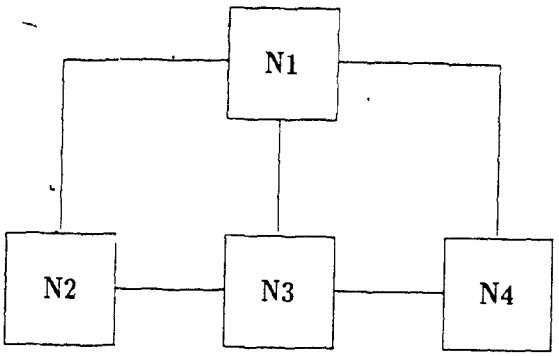

Fig. 3.1 The fault-tolerant basic module.

implemented by a local restructuring scheme described in Chapter 4

The module is a one-fault tolerant (or, one redundant) block It can tolerate at most one fault at any time Whether the module is faulty or not, three out of four nodes form the active sub-tree. Thus, the module makes a three-node two-level sub-tree for the complete binary tree

3.2 Module Interconnection

The module is a two-level three-node subtree In order to form a complete binary tree , it is necessary to connect the modules together in a certain manner Some additional links are required to connect the modules. The module can be designed in such a way that the interconnection links are also included within the module Fig. 3.2 shows a module with interconnection links included in it. Links L2 through L6 are the normal links of the module Seven extra links, L1 and L7 through L12, are added to this module for the interconnection purpose A module has one node in the first level and two nodes in the next level. The two nodes in the second level should have four sons in the next level So, four modules are connected to these two nodes in the second level of the first module The first level node of each of these four modules form the third level of the binary tree. So, these four modules form the third and fourth level of the tree Then, again, sixteen modules are connected to these four modules to form the fifth and sixth levels of the tree. This interconnection process continues until the desired depth of the tree is achieved. The interconnection scheme is shown in Fig 3.3

Notice that, though each higher level module is connected to four modules in the next lower level, six interconnection links are used instead of four This is because, the

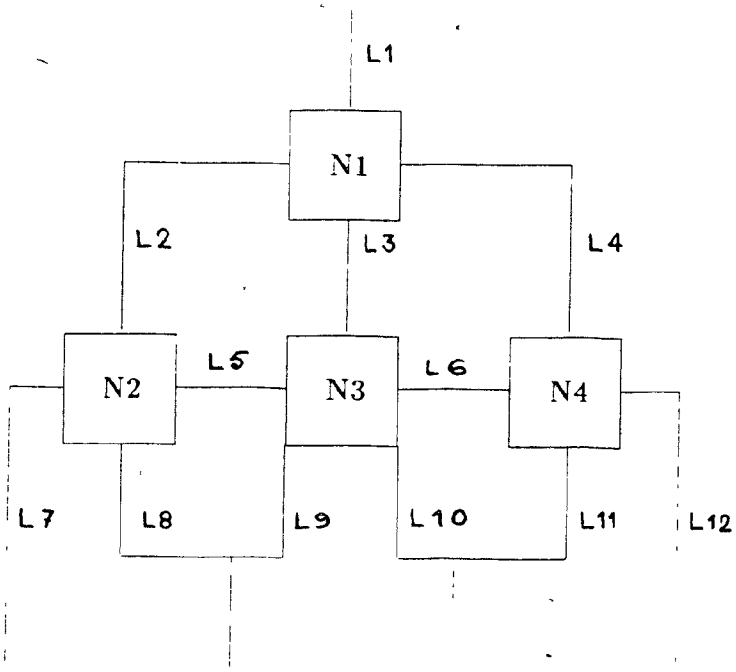

Fig. 3.2 The module with interconnection links.

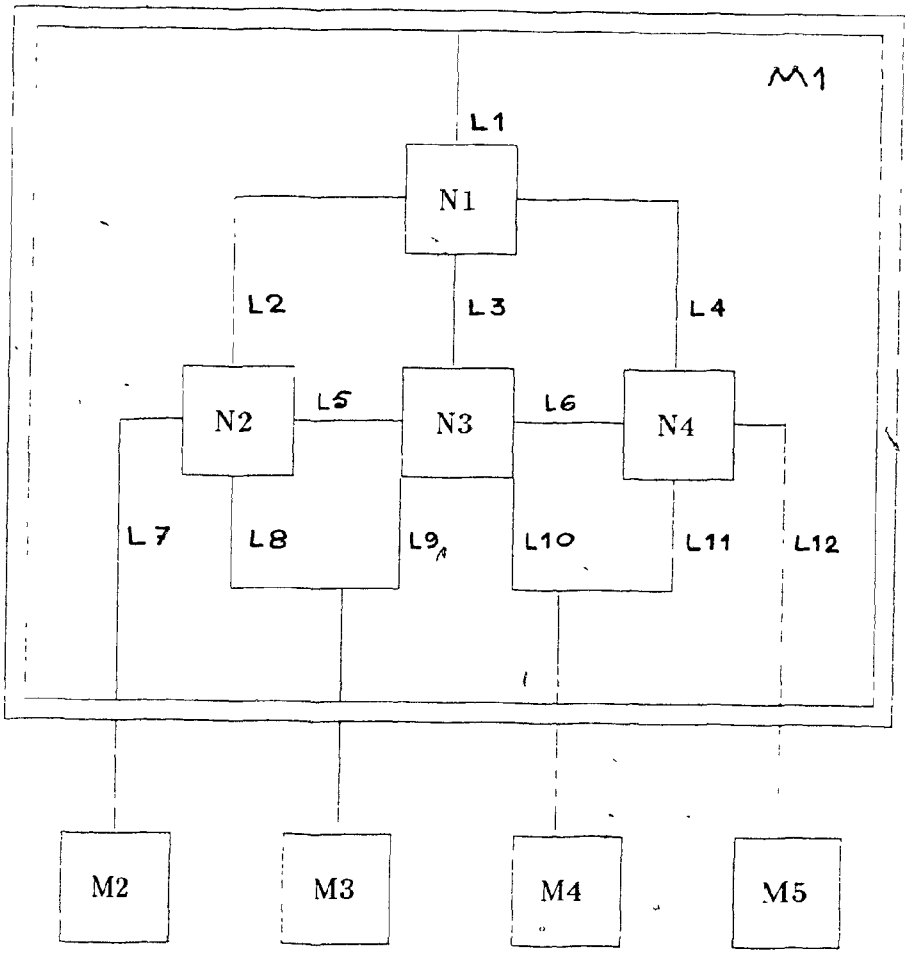

Fig. 3.3 Five interconnected modules.

interconnection between the two nodes of the connected modules is not fixed. Rather it is a dynamic interconnection which can eliminate the faulty node and connect two active nodes of the two interconnected modules. As for example, in Fig. 3.3. L8 and L9 connect M3N1 (node 1 of module 3) to M1N2 and M1N3 respectively This is a flexible connection. When M1N3 is faulty, L8 actively connects M3N1 and M1N2, and L9 is inactive. However, if M1N2 is faulty, the case is reversed. Similar is the situation where L10 and L11 connect M4N1 to M1N3 and M1N4 respectively. Thus, two extra interconnection links provide a *dynamic interconnection scheme* suitable for the fault-tolerant structure

3.3 Properties of the Modular Scheme

A module consists of four nodes connected by five links. Adding seven links for interconnection (one for the root and six for the sons). makes it a four node twelve link building block. We now investigate several interesting properties of this topology.

Property 1: *The number of links in a modular tree is given by*

$$32.5\Sigma k$$

$$= 32.5 * \sum_{i=0}^{m} 2^{4i} \qquad (3.1)$$

where 4m is the number of levels in the tree for any integer value of 'm'.

(Eqn. 3.1 is derived in Chapter 4).

If the reliability of the links are treated differently from the reliability of the nodes, that is, link and node failures are independent, then the number of links in the topology

influences the system reliability significantly A topology with a large number of links is likely to be less reliable than one with a smaller number of links, all else being equal.

It will be seen later in the next chapter that the link ratio is significantly improved for the modular scheme compared to RAE scheme.

Property 2: *The maximum degree of any node in a modular tree is five.*

From the point of view of node complexity, in terms of information transfer and message routing, it is important to keep the number of connections per node as small as possible. Whereas RAE scheme has the maximum degree of nine, this topoloy succeeds in this respect by guaranteeing that no node requires more than five connections to communication channels.

Property 3: *The modular binary tree is equivalent to a k-ary tree with k=4.*

Each module is connected to four modules in the next level to increase the tree depth Thus, treating the module as a black box, the binary tree becomes an equivalent 4-ary tree consisting of $\frac{1}{3}(4^{i+1} - 1)$ modules for any 'i' level 4-ary tree, as shown in Fig. 3.4 This is important because during the layout and other structural issues we consider the binary tree as an equivalent k-ary tree. This helps to obtain a symmetric structure convenient to repeat in VLSI

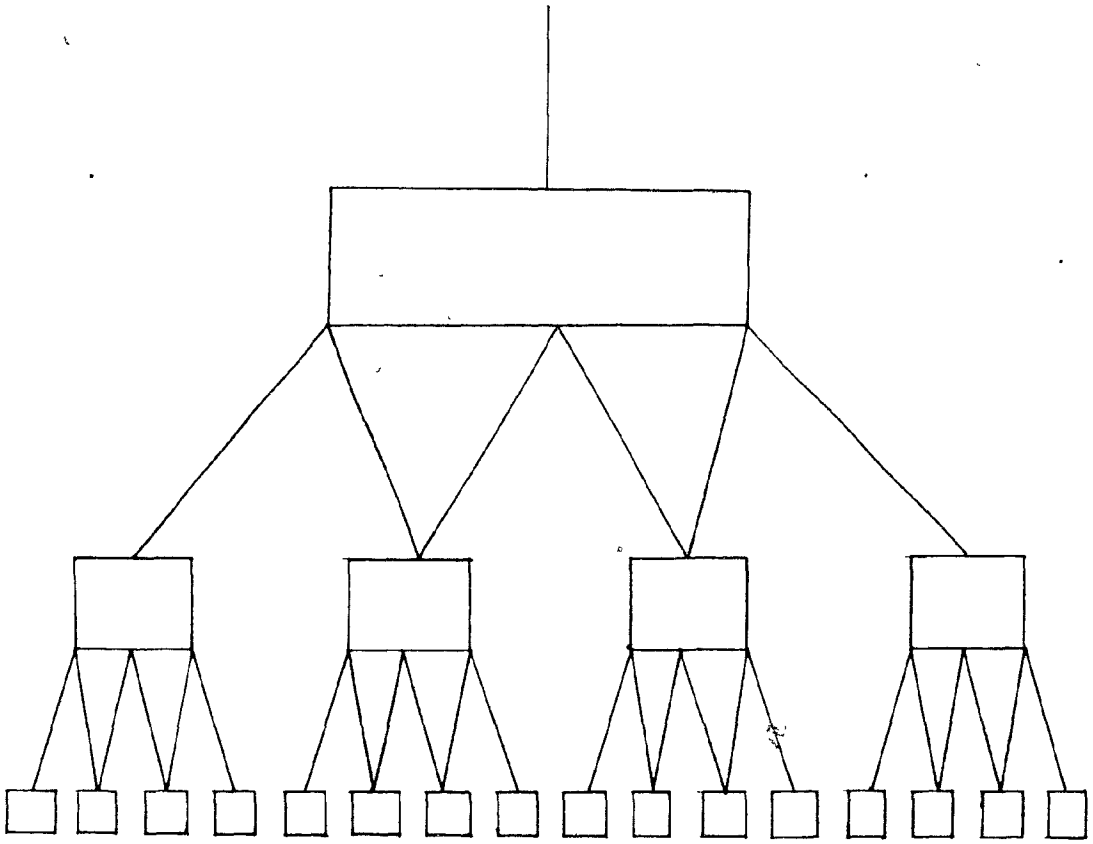

Fig. 3.4 Equivalent 4-ary tree using the modules

Property 4: *The modular binary tree can tolerate faulty nodes, at most, equal to one-third of the non-faulty nodes in the system.*

If there are n modules in a tree system and there is a single fault in every module, this 4n node binary tree can operate with n faults in it. If the fault is greater than n, there must be at least one module with more than one fault But a module cannot tolerate more than one fault at any time. So, it implies that a 3n node binary tree can at most tolerate n faults in it. It is also obvious from this fact that the modular binary tree has one-third redundancy in the system

3.4 Extensibility of the Scheme

The proposed modular scheme has the capability of extension. What it means is that, a building block larger in size can be constructed out of these smaller blocks to form the modular binary tree.

The first level extension scheme gives a three-level module by connecting two basic modules However, this connection for extension is something different from the module interconnection discussed earlier. An extended three-level module is shown in Fig. 3 5 . LX is the connecting link between the two basic modules for this extension scheme. Seven of the eight nodes in this module form a three level subtree for the complete binary tree. A study of Fig 3.5 makes it clear that any one of the eight nodes in the three-level module can go faulty and it can be restructured as a seven node three-level subtree Either N1 or $N1'$ acts as the module father.

The interconnection of three-level modules, to increase the tree depth, is the similar as in case of the basic modules. However, each three-level module is connected to eight

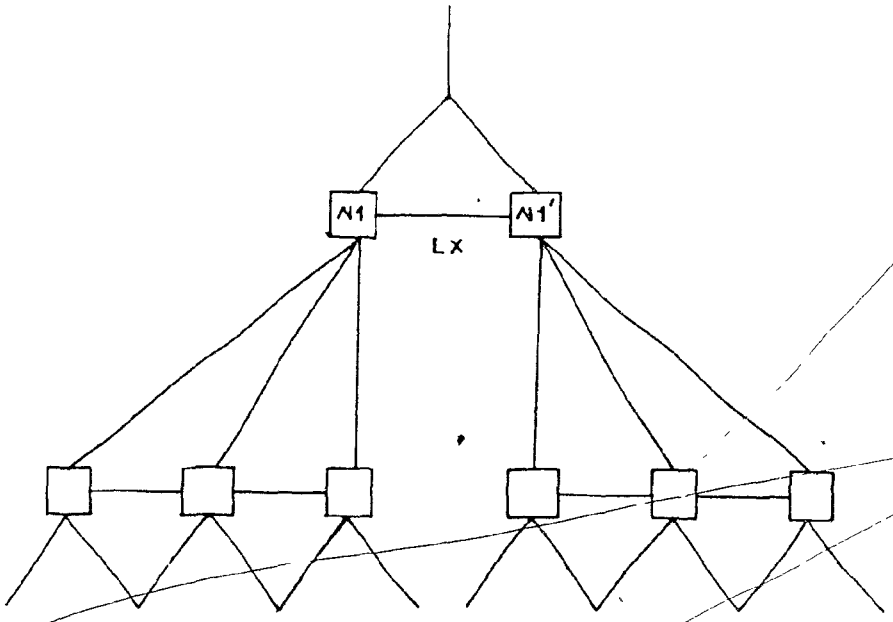

Fig. 3.5 Fault-tolerant three-level module.

three-level modules in the next level Therefore, it becomes an equivalent 8-ary tree.

The three-level module has one in seven redundant structure. A three-level module binary tree system can tolerate one fault in every seven nodes of the tree Thus, the three-level module tree becomes less fault-tolerant than a basic module tree Also, the system reliability, being a function of the number of spares, goes down for the three-level module tree system

The scheme can be extended further Four modules can be connected together to form a four-level module with sixteen nodes as shown in Fig 3 6 This scheme can tolerate one fault in every fifteen nodes The maximum degree of connectivity is seven

It is possible to extend the scheme further. But for the extended modules, the number of extra links and degree of connectivity are gradually increased while system reliability falls down.

Interconnection of different type of modules is also possible For example, a basic module can be connected to four three-level modules (or, one three-level module to eight basic modules) to form a five level binary tree However the system reliability is greater in the latter structure

Thus, different type of modules are available with different amount of redundancy and fault-tolerant capability The choice of a particular module depends upon the nature and demand of the system to be designed

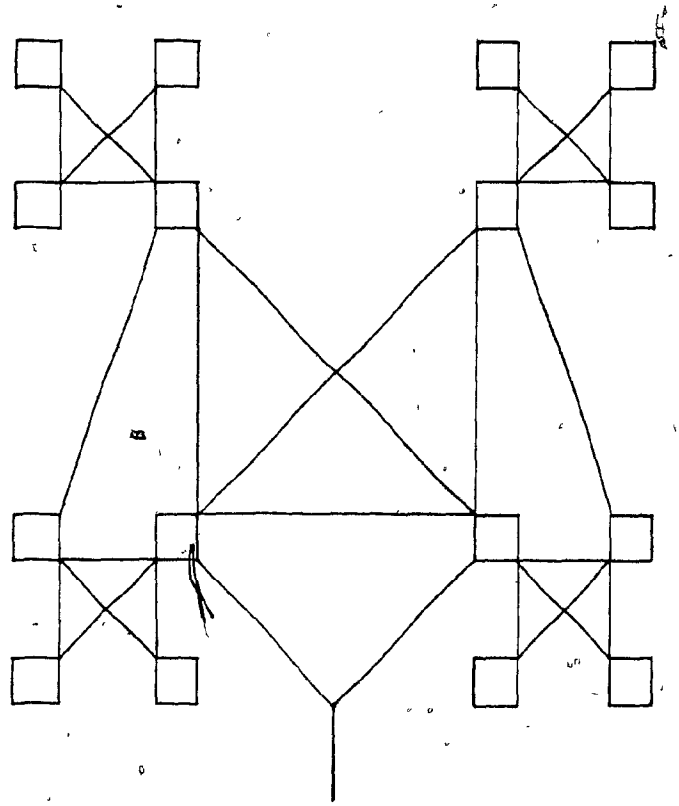

Fig. 3.6 Fault-tolerant four-level module.

4. Structure of the Module

In chapter 3 we dealt with the topology of the modular structure. In this chapter we discuss the layout and related topics. Section 4.1 describes the layout scheme of the modular tree and the complexity of the scheme. In section 4.2 a packaging scheme is shown for a multichip binary tree. Section 4.3 describes the restructuring methods necessary for the fault-tolerant binary tree to maintain the fixed tree structure after every fault occurrence.

4.1 Layout

In this section, we look into the lay out issue of the modular fault-tolerant binary tree in details. The standard methods of laying out a complete binary tree are discussed in Section 4.1.1 . The ways and means for the complexity analysis of the circuit comes in Section 4.1.2 . Then a layout scheme is described for the modular structure in Section 4.1.3 . This structure is compared with the proposed structure for RAE scheme at the end in Section 4.1.4 .

4.1.1 Layout of a binary tree

In [7] Leiserson considered the problem of laying out a complete binary tree of $n = 2^k - 1$ vertices, where k is the number of levels of the tree. Fig 4.1 shows an obvious solution that requires $O(nlogn)$ area - $O(n)$ across the bottom times $O(logn)$ height. As the tree is ascended from the leaves to the root, the number of wires is halved from one level to the next, but the length of the wires doubles This means that the wire area devoted to each level of the tree is the same.

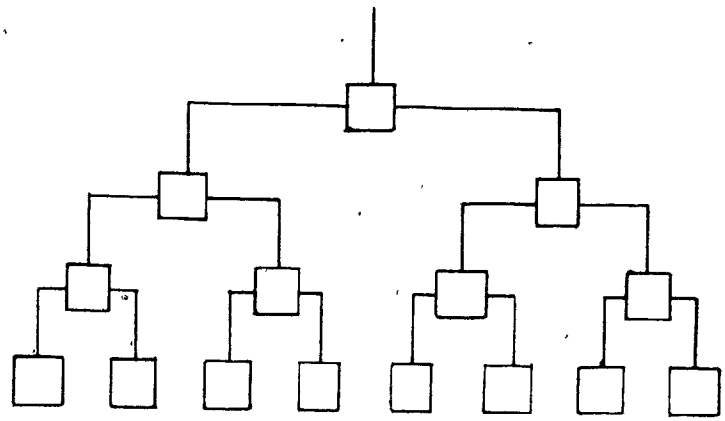

Fig. 4.1 O(nlogn) area layout of a complete binary tree.

Structure of the Module

There is a more efficient solution to this embedding problem The so-called *H-tree* layout shown in Fig. 4.2 requires only $O(n)$ area in spite of the fact that relatively long wires are used towards the root of the tree In this layout the number of wires is halved from level to level as we ascend to the root, but the length of the wires double every two levels. Whereas the standard $O(nlogn)$ layout uses just one dimension for routing most of the wires, the H-tree makes better use of both spatial dimensions.

Also in [6] Ullman shows how to layout a complete binary tree in $O(n)$ area by placing leaves throughout the chip However, if the leaves of such a tree are constrained to be on the border, then $\Omega(nlogn)$ area is needed for it It is also shown that the H-tree, both theoretically and in practice, is essentially the most area-efficient way to layout a complete binary tree To within a constant factor, the H-tree is as good as possible, since surely $\Omega(n)$ area is needed to layout any graph with n nodes

The length of wires in any layout is of concern because it limits the speed at which the chip can run reliably In [6] it is shown that $\Omega(\sqrt{n}\ logn)$ is a lower bound on wire length for complete binary trees and this bound is achievable Although the H-tree layout cannot achieve this bound, still it is close to the bound The H-tree has individual wires half as long as a side of the layout In terms of the number of nodes of the tree, n, the longest wire has length $\Omega(\sqrt{n})$

4.1.2 Lower bounds on Area and Time

In the VLSI domain, the complexity of a circuit is analyzed in terms of lower bounds on the area and time or some combination of these

One of the simplest forms of lower bound arguments concerns the memory require-

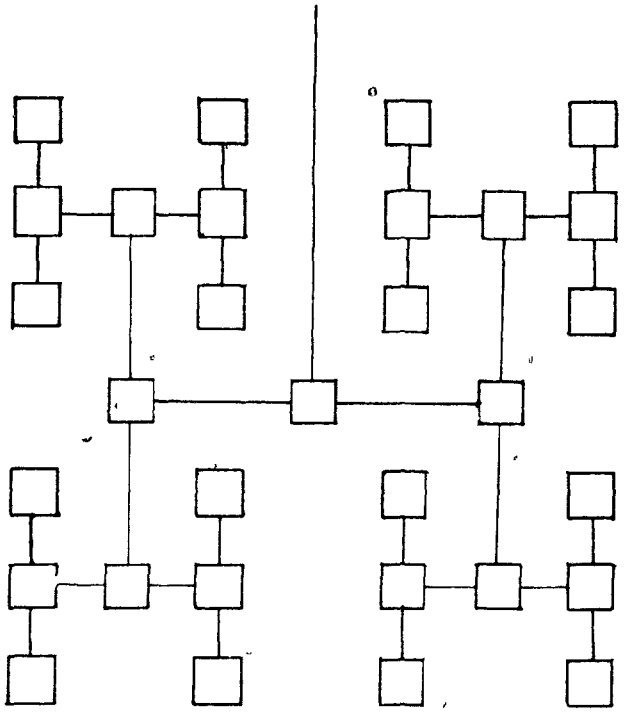

Fig. 4.2 H-tree layout of a complete binary tree

ment of a circuit and its relation to the area of that circuit If a circuit has area A, it does not have more than A circuit elements, and therefore cannot remember more than A bits from one time unit to the next. Thus, the lower bound on A, the area of a circuit, can be used to analyze its complexity.

According to [6], in many cases, area based bounds are weak, in the sense that there do not appear to be circuits as good as the bound imply might be possible Many strong lower bounds, that do match the best circuits we can construct, are lower bounds on the product AT^2, where A is the area of the circuit and T is the time used by the circuit. So, we will analyze the complexity of the modular scheme in terms of A and AT^2 bounds and compare the results with those of RAE scheme in the following two sections.

4.1.3 Layout : Modular Scheme

It has been seen at the beginning of this section that the H-tree method of layout is the optimal and the most area-efficient way to layout a complete binary tree The modular fault-tolerant binary tree, structurally, is different from a non-redundant binary tree (although it is equivalent from the functional point of view) In spite of that we have a layout scheme for the modular structure which is *H-tree like* The most interesting feature is that it has an area-bound of $O(n)$, which is optimal for this n node graph.

In the conventional H-tree layout, the nodes of the complete binary tree are placed on a grid in a recursive pattern like the letter "H" However, in the H-tree like method, a module corresponds to a single node in the H-tree layout. Here, five modules are connected together to form the H-structure. Four modules are placed at the four corners

These are connected to a fifth module at the center to complete the H-tree pattern. Such an H-tree is shown in Fig. 4.3 Four levels of the binary tree have formed this one-level H-tree. The one-level H-tree can be repeated in a recursive pattern to get deeper trees Thus, placing four one-level H-trees at the four corners and connecting those with a module at the center give the two-level H-tree or six-level binary tree (shown in Fig. 4.4) In general, any 'i' level H-tree is equivalent to (2i+2) level binary tree.

Therefore,

$l = 2i+2$

or, $h=2i+1$ for $h \geq 3$

where, l=levels of the binary tree (only for even 'l')

and h=height of the binary tree.

Now, for any 'l' levels, total number of nodes in the fault-tolerant binary tree is

$(2^l - 1) + (2^l - 1)/3$

$= \frac{4}{3}(2^l - 1)$

$= \frac{4}{3}(2^{2i-2} - 1)$.

By placing the nodes and links of the modular binary tree in grid, we find that,

For i=1 (or, l=4), area $A=8 * 7$

For i=2 (or, l=6), $A=(8 * 2 - 1) * (7 * 2 - 3)$

For i=3 (or, l=8), $A=(8 * 4 - 1 * 3) * (7 * 4 - 3 * 3)$

and so on.

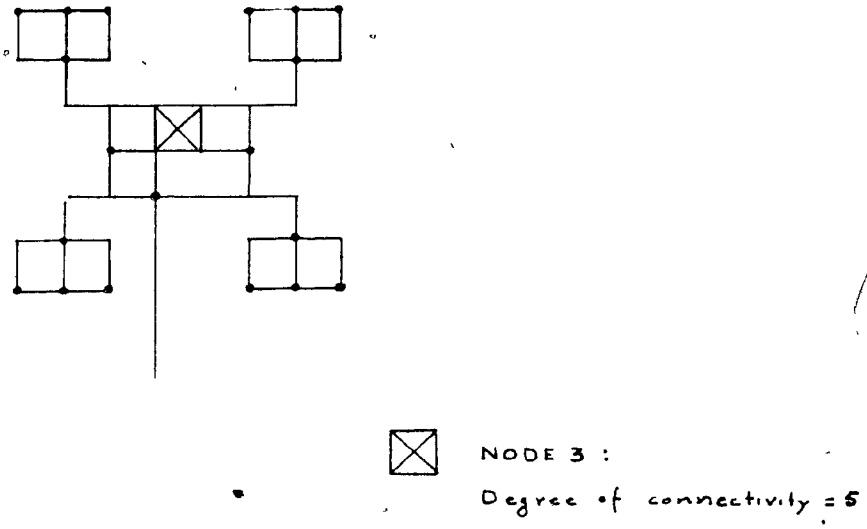

Fig. 4.3 H-tree like layout of a modular binary tree (H=1).

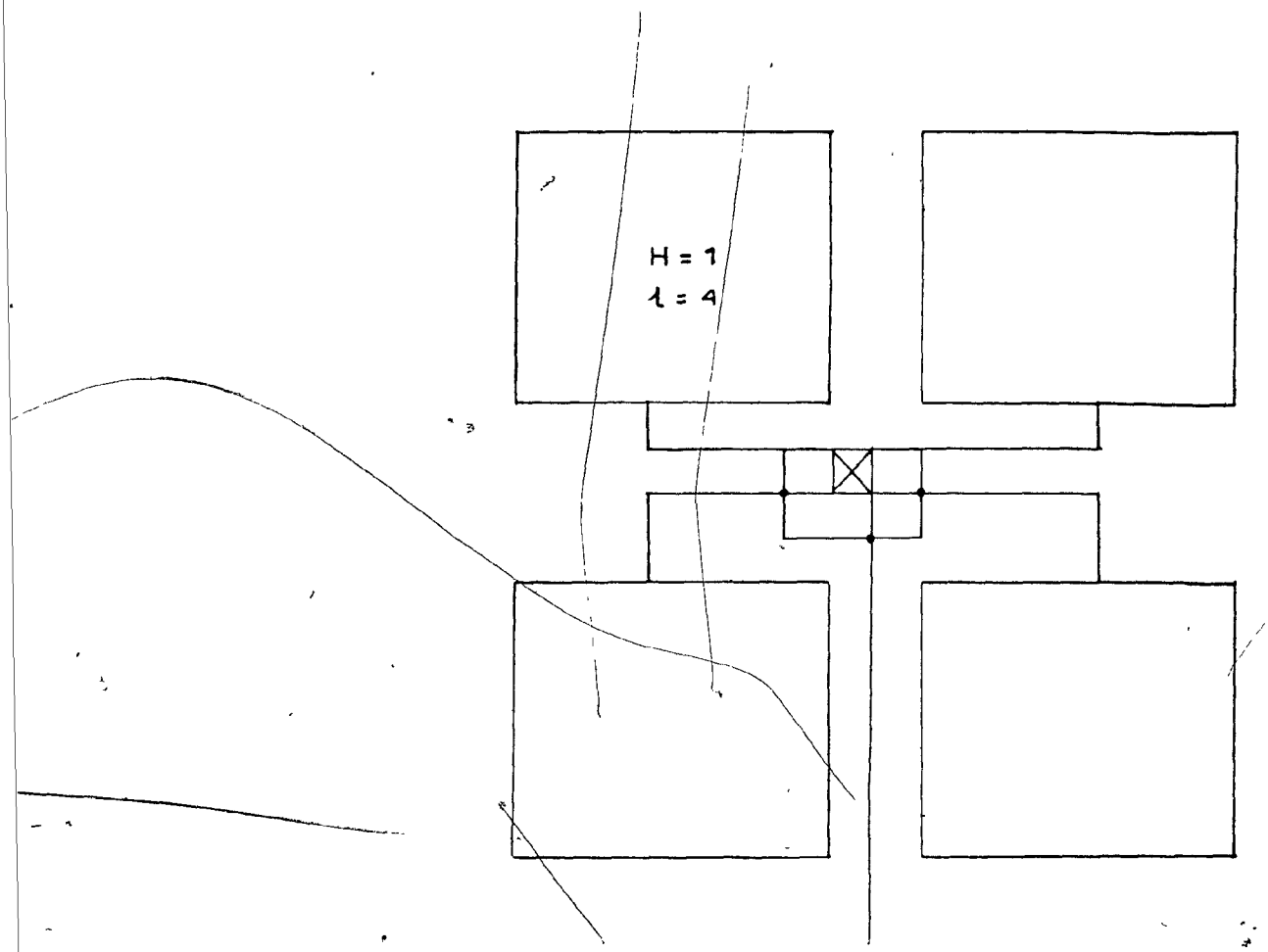

Fig. 4.4 H-tree like layout of a modular binary tree (H=2).

Theorem : The H-tree like layout for a modular binary tree of n nodes has area $O(n)$.

Proof: An easy induction on i shows that the H-tree of order i has area,

$$A = [2^{i+2} + (2^{i-1} - 1)] * [(2^3 - 1)2^{i-1} + 3(2^{i-1} - 1)]$$
$$= [2^{i-1}(8 + 1) - 1] * [2^{i-1}(7 + 3) - 3]$$
$$= [\tfrac{9}{4}(2^{i+1}) - 1] * [\tfrac{10}{4}(2^{i+1}) - 3]$$
$$= O(\sqrt{2^{2i+2}}) * O(\sqrt{2^{2i+2}})$$
$$= O(\sqrt{n}) * O(\sqrt{n})$$
$$= O(n).$$

So, the H-tree like layout is optimal for the modular fault-tolerant binary tree.

In case of the conventional H-tree, the individual wires are half as long as a side of the layout. The H-tree like layout given for the modular binary tree also has the longest wire half as long as a side of the layout Thus, considering the propagation time to be proportional to the length of the wire,

$$T = O(\sqrt{n}).$$

So, the bound on information flow is found to be

$$AT^2 = O(n^2).$$

4.1.4 Layout : For RAE Scheme

Raghavendra et al have not given any layout for their scheme. So, for the sake of comparison, we tried to layout their *one spare per processor* scheme in a grid While laying out, we tried to maintain a modular structure as far possible so that an orderly extension is possible. Decoupling network blocks are omitted for the sake of simplicity

Structure of the Module

The scheme consists of three types of blocks. Six nodes in the lowest level together with three nodes in the next level of the tree form a building block (Block A in Fig 4.5) Two such blocks are then connected to another block with three nodes in the next level (Block B in Fig 4.6) Finally, this big block is connected to a third type of block (Block C in Fig 4.6) consisting of the two nodes in the highest level of the tree This completes a four level binary tree. To construct a five level binary tree, four blocks of A are connected to two blocks of B These are, in turn, connected to another block of B which is finally connected to Block C In general, to increase the tree depth Block A is repeated in its own level side by side. Block B is also repeated in its own level However, an extra level has to be created each time with Block B between the previous last level of Block B and Block C. One level of Block C always lie at the end

As the number of levels is increased, the grid grows faster in length than in width. A rectangular grid is obtained with a lot of wastage in space A large number of processors (nine) are directly connected by means of redundant links This makes the layout area inefficient

For any 'i' levels of the tree, total number of nodes is

$$(2^i - 1) + (2^i - 1)\ 2$$

$$= \frac{3}{2}(2^i - 1)$$

By placing the nodes and links of the tree in grid we find that,

For 3 levels of the tree, area $A = 11 * (5 - 1) = 11 * 6$

For 4 levels of the tree, $A = (11 * 2 - 4) * (5 - 2 - 3 - 1) = 26 * 11$

For 5 levels of the tree, $A = (11 * 2) * 2 - 4 * 3 * ((5 - 2 - 3) + 2) - 3 + 1 = 56 * 16$

For 6 levels of the tree, $A = ((11 * 2) * 2) * 2 - (4 * 3) * 2 - 4 * 15 - 2 - 3 - 1 = 116 * 21$

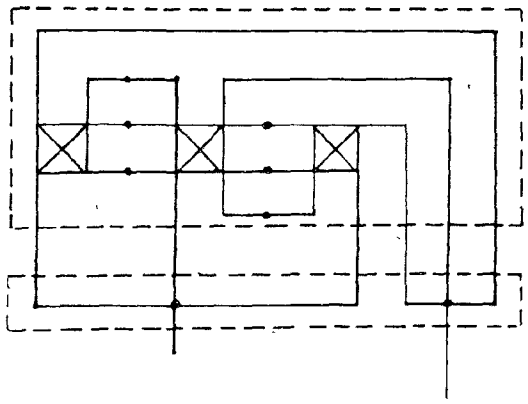

Fig. 4.5 A layout of RAE extended scheme with spares (l=3)

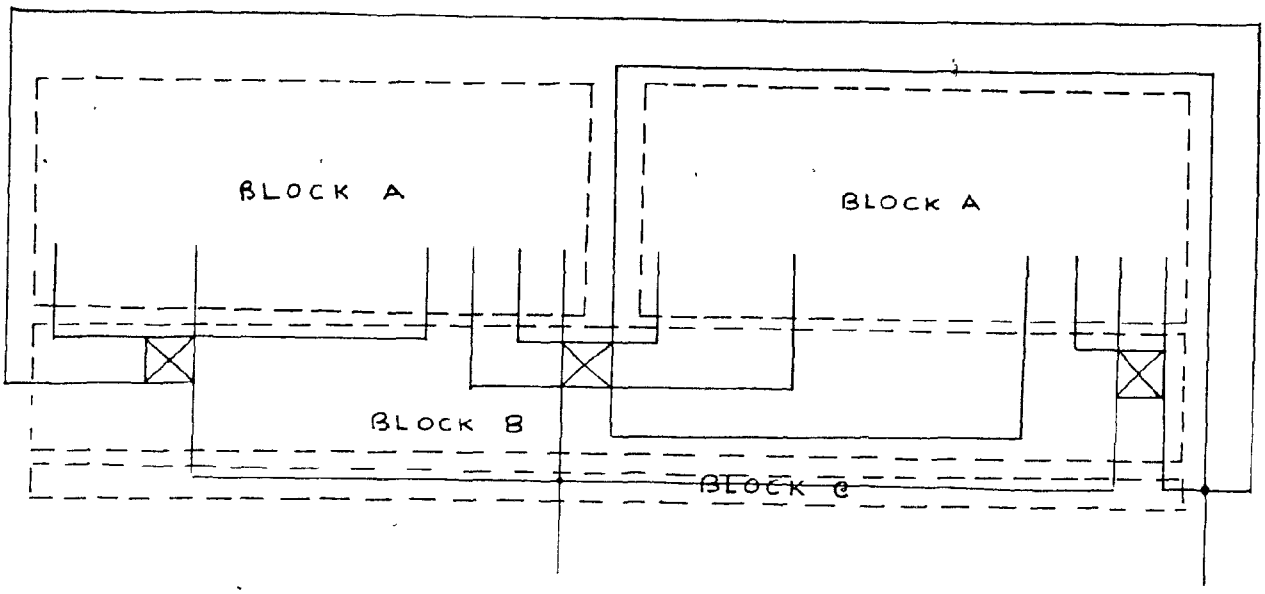

Fig. 4.6 A layout of RAE extended scheme with spares (l=4)

Therefore, for any 'i' levels,

$$A = [11 * 2^{i-3} + 4(2^{i-3} - 1)] * [5(i-2) + 1] \text{ for } i \geq 3$$

$$= (\tfrac{15}{8} * 2^i - 4) * (5i - 9)$$

$$= O(2^i) * O(i)$$

$$= O(nlogn).$$

The longest wire in this layout traverses the whole length and width of the grid. Thus, the longest wire has the length of $O(n + logn) \approx O(n)$.

Therefore,

$$T = O(n).$$

$$\text{and } AT^2 = O(n^3 logn).$$

Table 4.1 shows the ratio of the layout area required by the binary tree using modular scheme and RAE scheme for different tree depths.

4.2 Link Ratio

Both the modular scheme and RAE scheme have redundant links in the tree to provide fault-tolerance. The number of links in the system has an influence on the layout scheme of the system. In fact, we have seen in the previous section that the large number of links in RAE scheme have made the layout complicated. Here, we compare the number of links used in these two schemes.

We begin with any four-level section of a binary tree system as shown in Fig. 4.7. Let there be k non-redundant processors in the first level of this section connected by k non-redundant links to the next higher level, where $k = 2^{4i}$ for any integer value of i. Beginning with this assumption we get the data in Table 4.2.

No. of Levels in Binary tree	*Layout Area for RAE Scheme*	*Layout Area for Modular Scheme*	*Ratio of Areas*
4	308	64	4.8125
6	2,604	361	7 2133
8	15,748	1,681	9 3680
10	83,804	7,225	11.599

Table 4.1 Comparison of the layout area requirement for a complete binary tree using two different schemes

Scheme used	Level Number	Number of Active Links	Number of Redundant Links
Modular Scheme	1	k	k 2
	2	2k	3k
	3	4k	2k
	4	8k	12k
One spare per pair (RAE) Scheme	1	k	2k
	2	2k	4k
	3	4k	8k
	4	8k	16k

Table 4.2 Number of links in a 4-level section of a fault-tolerant binary tree.

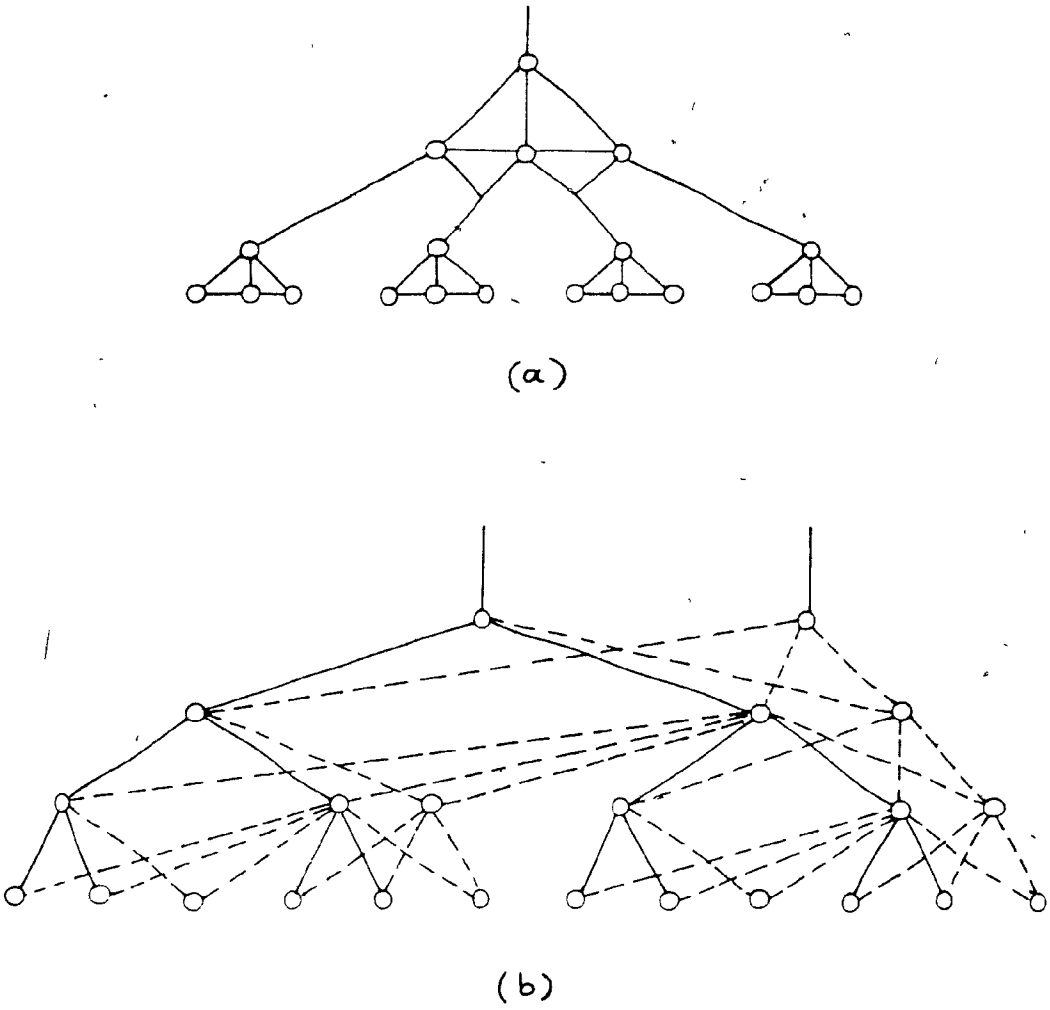

Fig. 4.7 The four-level section of a binary tree using,

(a) the modular scheme,

(b) the one spare per pair (RAE) scheme.

Structure of the Module

From the table, for a 4-level section,

Total links (Modular scheme) = 32.5k.

Redundant links (Modular scheme) = 17.5k

Total links (RAE scheme) = 45k.

Redundant links (RAE scheme) = 30k.

Now, the total number links in the tree can be obtained by summing up for different values of k

Therefore, the ratio of total links is

$$\frac{45\Sigma k}{32.5\Sigma k} = 1.384$$

and the ratio of redundant links is

$$\frac{30\Sigma k}{17.5\Sigma k} = 1.71428$$

Note that the link ratio remains the same for any number of levels in the tree

Thus, RAE scheme requires 1.71 times as many redundant links as required by the modular scheme.

4.3 The MultiChip Binary Tree

Although integrated circuit technology is advancing at a breathtaking pace, one sector of that technology is crawling in comparison The number of external connections from an integrated circuit is severely limited. Whereas some enthusiastic technologists project an eye-opening 10^8 components per chip, two hundred pins per chip seems a

large number to most. A chip that requires many more is unlikely to be realizable for quite some time

According to Leiserson [7], a complete binary tree is an attractive structure from this point of view if the tree fits entirely on one chip and the root is the only off-chip connection. Several researchers have proposed, however, that much larger tree systems be built. When any system is larger than a single chip, it becomes necessary to partition it among separate chips that can be assembled at the printed circuit level.

As it is claimed earlier, modularity is the most attractive feature of this design. The module can be repeated as many times as required to get the desired tree depth. However, for greater tree depth, the complete binary tree may not fit on a single chip. In that case, partitioning is necessary. Here, again, modularity is a great advantage. We can house one module in a single chip with five off-chip connections (four for the two sons and one for the module father). Thus, arbitrarily large complete binary trees can be built out of a single chip. At the printed circuit level, the structure is a complete k-ary tree with k+1 off-chip data paths. k is four in this particular case. This is shown in Fig. 4.8.

However, with the increasing density of components in the integrated circuit, it is possible to accommodate a large number of modules in a single chip. A chip with five modules (20 processors) is shown in Fig. 4.9. It has seventeen off-chip data paths (which is well within the available pin limit). This chip can also be used in the same manner to form a large binary tree.

An even higher density is possible with the structure as a complete k-ary tree with k+1 off-chip data paths where $k = 2^i$ for $i = 1, 2, 3, \ldots$

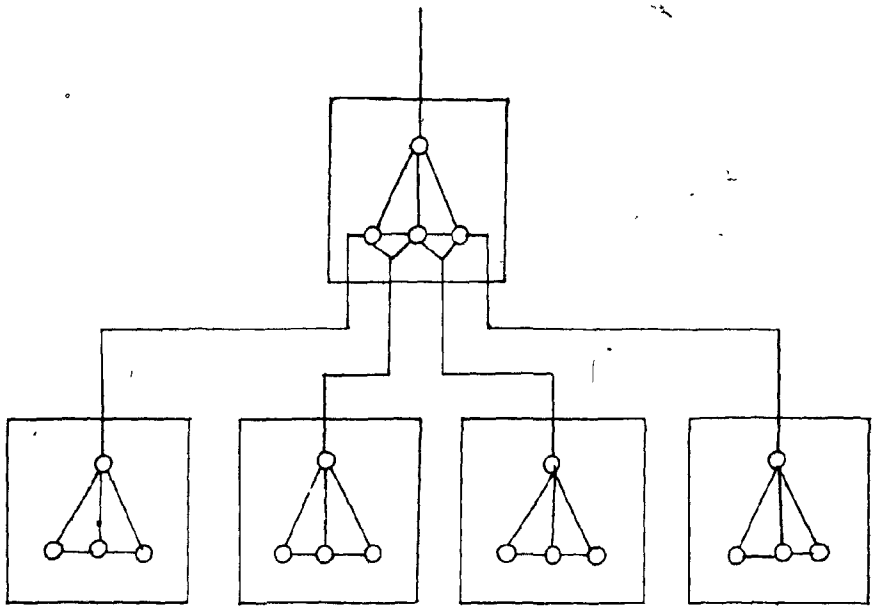

Fig. 4.8 Multichip packaging of the modular binary tree.

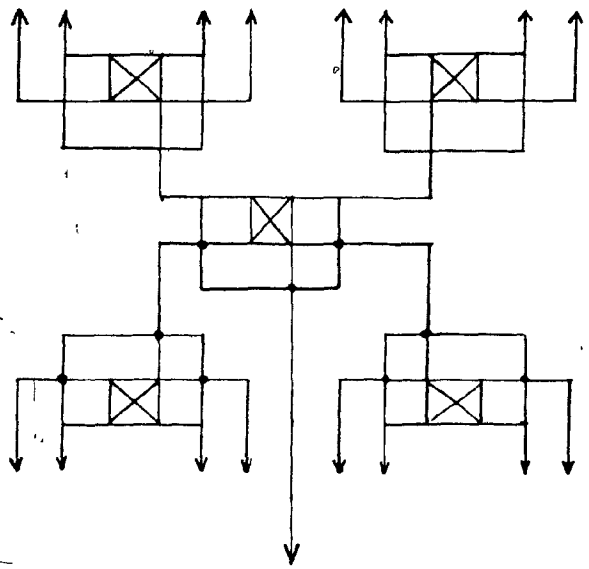

Fig. 4.9 A denser chip with five modules.

4.4 Restructuring

As was described earlier, the module consists of four nodes connected by twelve links. Being a fault-tolerant module, any one of these four nodes can go faulty and the remaining nodes act as a two-level three-node sub-tree. Whenever a node fails in the module, this faulty node should go out of the active binary tree and a redundant node should replace this faulty node before the tree begins to function again. This internal change in the module is necessary for maintaining the fixed binary tree structure. Thus, *restructuring*, as we can call it, is a very important feature for the fault-tolerant binary tree.

Restructuring is an operation which is not required for the non-fault tolerant architecture. It is clear that all the normal operations should be halted during the restructuring period. In other words, restructuring is an unwanted interruption in the normal operation mode. So, it is desired that the restructuring operation of the fault-tolerant architecture should be fast and simple compared to the normal operation. Overhead involved with restructuring should also be small.

The fault-tolerant binary tree has two modes. The first one is the operation mode which is common for all types of trees (redundant or non-redundant). Secondly the restructure mode which is specific for the fault-tolerant structure. But, prior to restructuring, we have to think of another mode, namely, the *test mode*. That is the fault diagnosis issue of the structure. For the detection and location of faults, there should be some testing scheme. At present, it is assumed that some standard testing scheme like periodic testing takes care of the fault diagnosis part. But it is really an open problem. One can think of an on-line real-time testing or a BIT (Built in Testing) scheme

particularly well-suited for this modular structure

As soon as the testing is done, restructuring operation should begin So far, the nodes in the module are considered as black boxes But, for throwing out a faulty node and bringing a new node in we have to look into some details of these black boxes

A node consists of some processing elements forming the processor and the data elements In the fault-tolerant module whenever a node is a part of the active tree its processor is connected by links to the rest of the tree But for a faulty node the links are not supposed to connect the processor Rather, the link by-passes the faulty processor and connects another link Referring to Fig 4.10 , if N1 is not faulty L3 and L1 connect to N1 But when N1 is faulty L3 by-passes the processor in N1 and connects L1 Similarly when N2 is faulty L7 by-passes N2 and connects L5 Thus there are basically two types of connection

(1) A link connecting a processor

(2) A link connecting another link by-passing a processor.

4.4.1 Restructuring : RAE Scheme

RAE have described their redundant binary tree system of processors as a special purpose processor attached to a host computer The host can perform such tasks as compiling and initializing the tree system When a failure is detected by any node it signals the host which can perform reconfiguration and recovery of the executing program They also use decoupling networks controlled by a separate host The hardware reconfiguration is performed by setting switches in the decoupling network When a processor fails, all the links of the processor to the right of it are adjusted to right neighbors So,

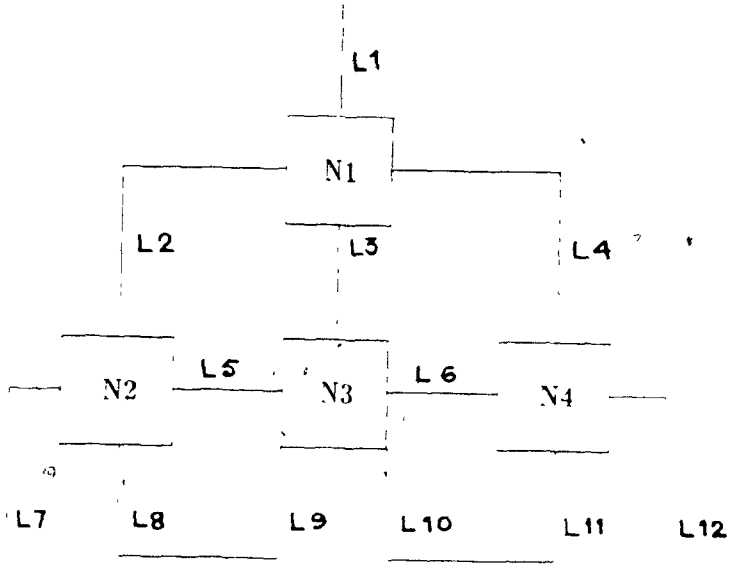

Fig. 4.10 The module with interconnection links

at level 'i', a readjustment of 2^i processors may be necessary in the worst case. For large value of 'i', this can be a huge readjustment for a single failure. Also, the sorted machines like Atallah and Kosaraju [1] dictionary machine cannot operate with this readjusted structure. The sorted data structure will be lost due to readjustment So, the data have to be shifted accordingly to get back the sorted order before the machine can operate again This, also, can be a huge shifting which may require additional logic.

4.4.2 Restructuring : Modular Scheme

In contrast to RAE restructuring scheme, we propose a simple switching scheme which is local to the module. The use of a separate host in RAE scheme means that either the restructuring information should flow through all the levels of the tree (which is time consuming) or the host should be directly connected to each node Direct access to each node complicates the layout Also, the use of decoupling networks does not help much. This network would, again, require external control The insertion of a block of decoupling network between each level of the tree would make the layout even more complicated

Keeping these drawbacks in mind, a restructuring scheme is developed which does not require any external control or decoupling network as a separate unit The basic idea is to connect the processor if it is faulty and to by-pass it for faulty condition This is done by using several switches connected to each node. The controls are generated for these switches based on the test results of the module

From the restructuring point of view, N1, N2 and N4 of the module are identical (Fig 4.10) This is because, each of these processors is included in the tree if it is good

and is by-passed if faulty In fact, these three nodes can be termed as *self-controlled* nodes Looking at its own status (faulty or good) the processor can cut itself out of the tree or can connect itself as an active node None of these three nodes has to depend upon the status of any other node even in its own module It is assumed that the test result is stored in a one-bit fault-tolerant register for each of these nodes The status-bit becomes the control bit for the switches surrounding the processor

However, N3 of a module is different from the other three nodes In some sense it supplements the other nodes of the module For example when N1 is faulty, N3 replaces N1 and becomes the father of N2 and N4 But when N2 is faulty, N3 takes the place of N2 as the left son of N1 Similar is the case for the failure of N4 The role of N3 is rather flexible in the module This is why N3 is not a self-controlled node in the module It has to look at the status of the remaining three nodes and restructure it as a father, left son, right son or a spare depending upon the environment Interestingly, N3 does not have to take into account its own status This is due to the basic assumption that only one node can go faulty in a module at one time So, when the status of N1, N2 and N4 are available the status of N3 is implied The status bits of N1, N2 and N4 are sufficient to generate controls for the switches surrounding N3 Although the controls for N3 are not as simple as in case of N1, N2 or N4 it is bound within the module

Fig. 4.11 shows the location of the switches controlling different processors in the module The location of the switches and their positions (close or open) are shown in Fig 4.12a through 4.12d for the faulty condition of different processors in the module Table 4.3 gives the switch positions Each of N1, N2 and N4 has five switches (S1 through S5) For a faulty node, S1 is closed and S2 through S5 are open. This puts the

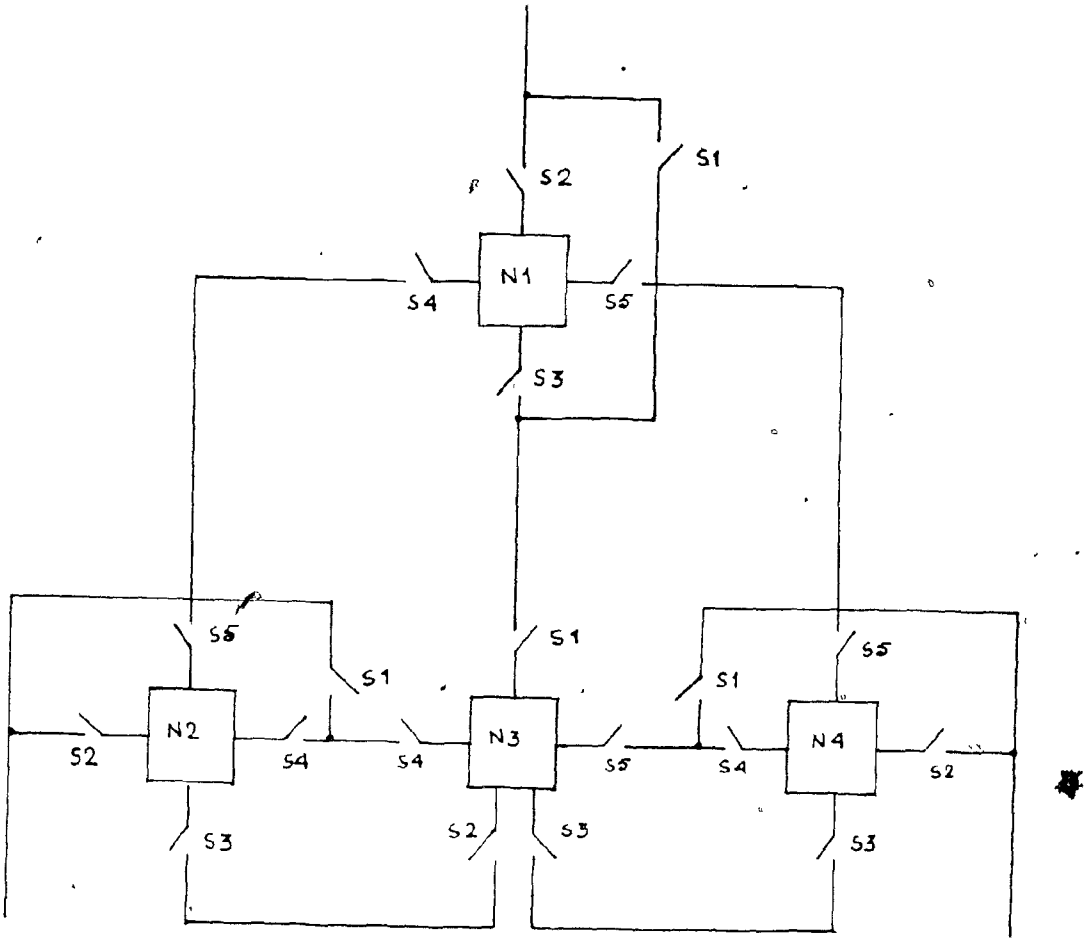

Fig. 4.11 Switches for restructuring a faulty module.

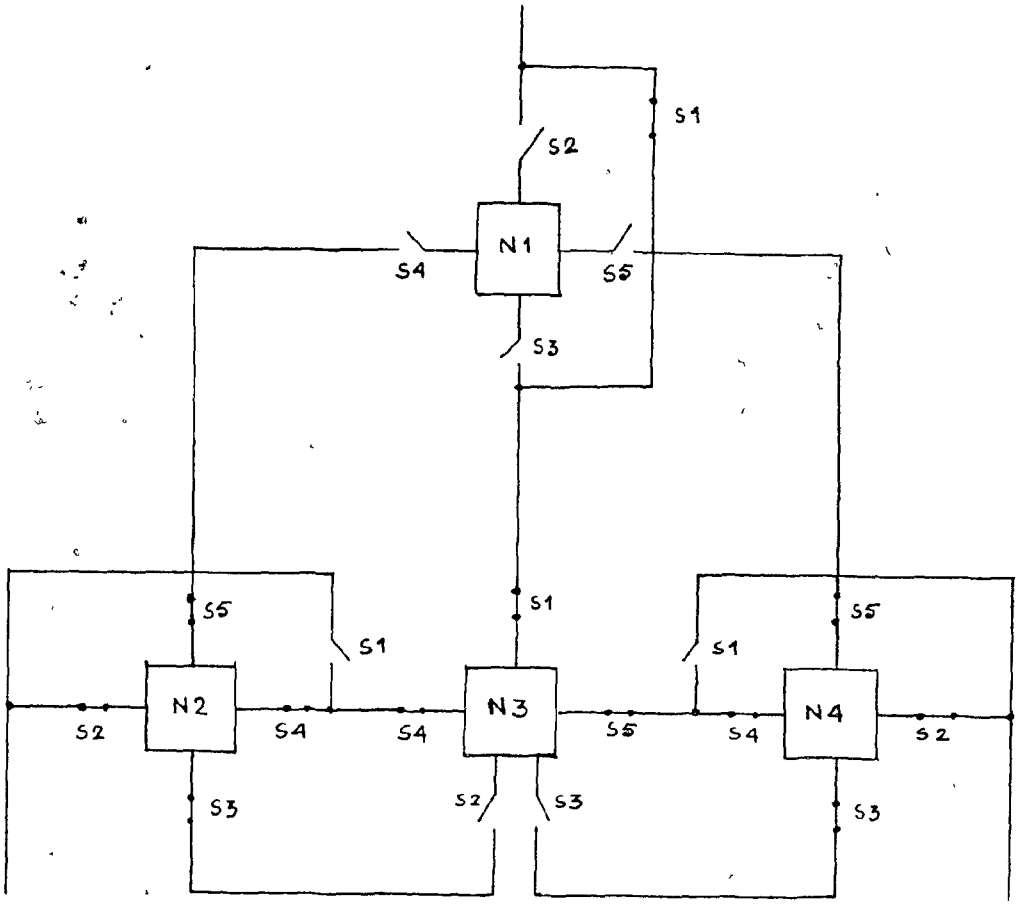

Fig. 4.12a Switch positions for faulty N1.

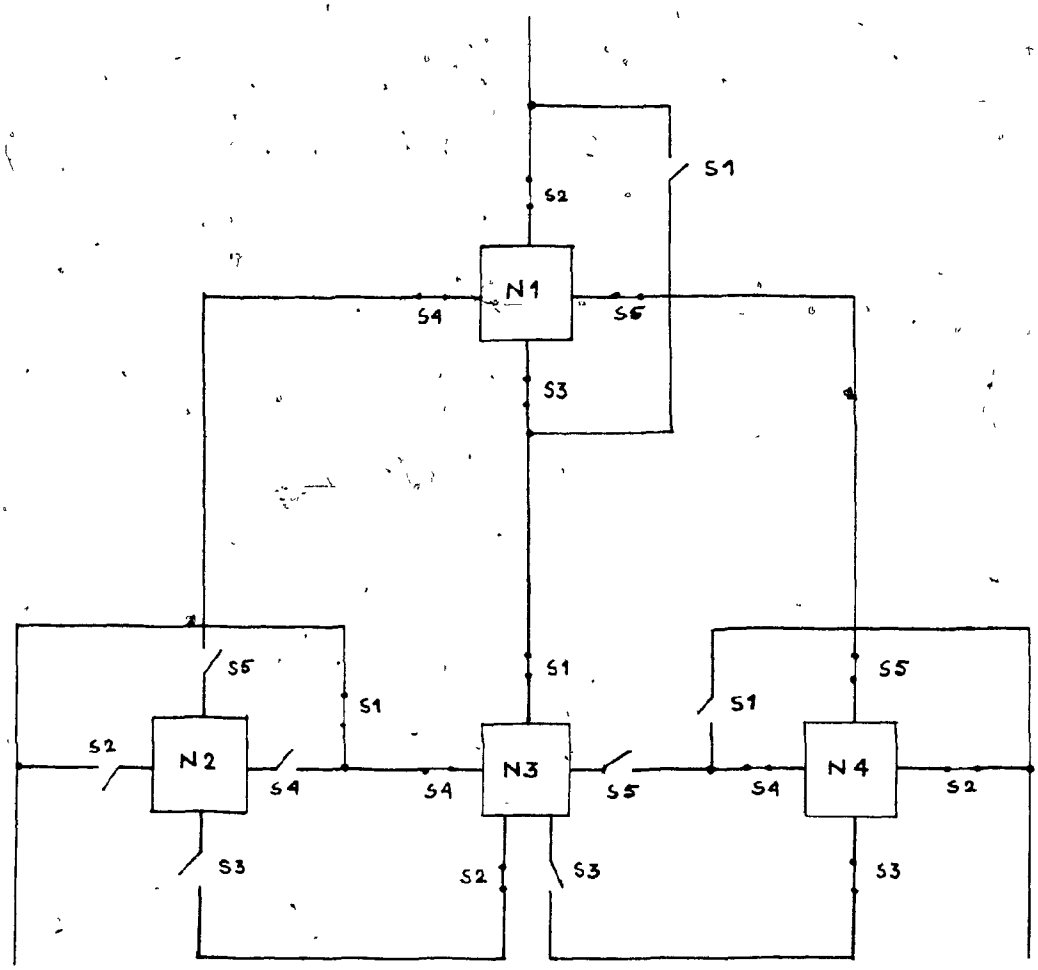

Fig. 4.12b Switch positions for faulty N2.

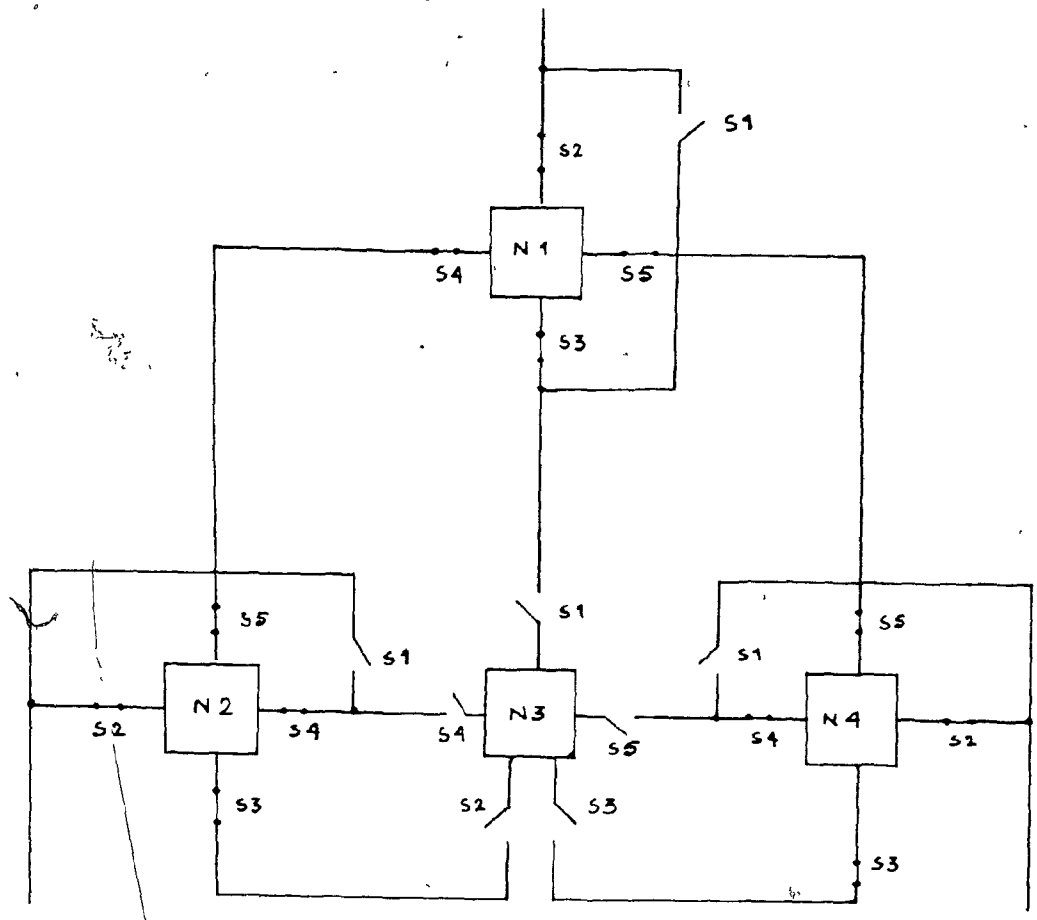

Fig. 4.12c Switch positions for faulty N3.

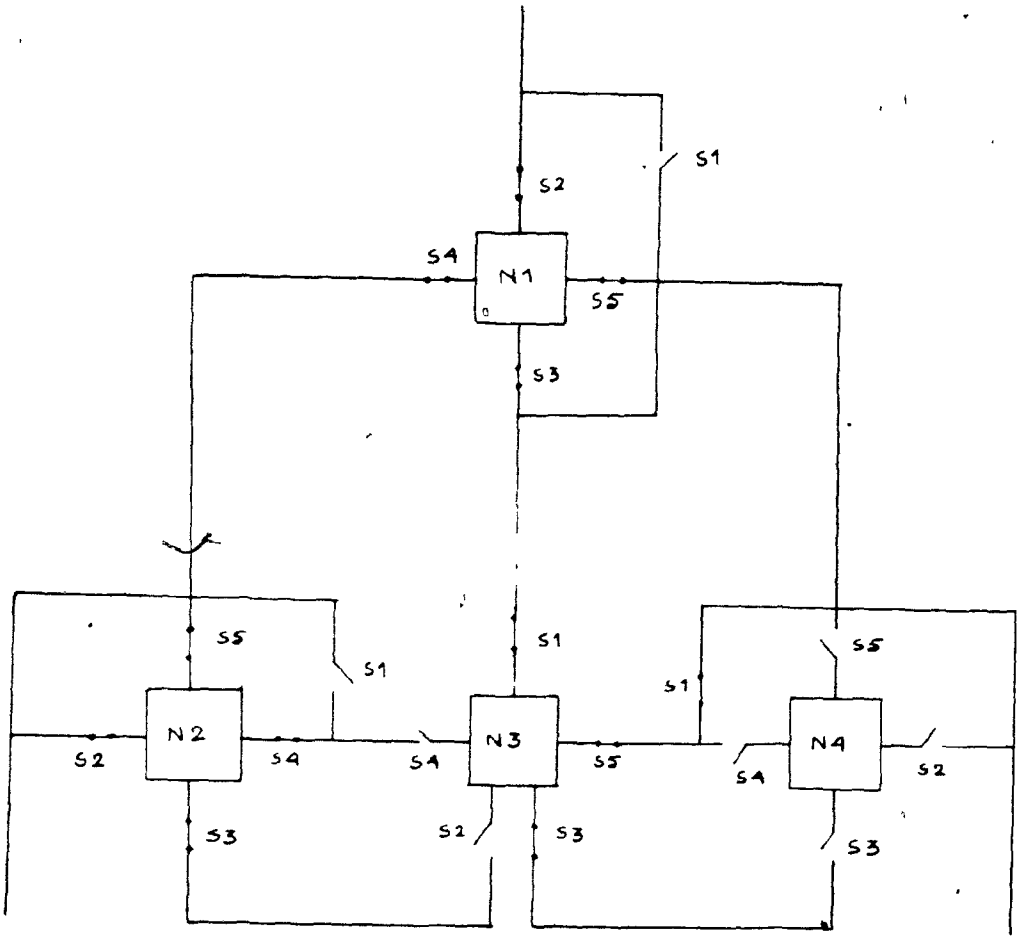

Fig. 4.12d Switch positions for faulty N4.

Module Status	*N1 Switches*	*N2 Switches*	*N3 Switches*	*N4 Switches*
No-Fault	$\overline{S1}$ S2 S3 S4 S5	$\overline{S1}$ S2 S3 S4 S5	$\overline{S}1\overline{S}2\overline{S}3\overline{S}4\overline{S}5$	$\overline{S}1$ S2 S3 S4 S5
N1 Faulty	S1 $\overline{S}2\overline{S}3\overline{S}4\overline{S}5$	$\overline{S}1$ S2 S3 S4 S5	S1 $\overline{S}2\overline{S}3$ S4 S5	$\overline{S}1$ S2 S3 S4 S5
N2 Faulty	$\overline{S}1$ S2 S3 S4 S5	S1 $\overline{S}2\overline{S}3\overline{S}4\overline{S}5$	S1 S2 $\overline{S}3$ S4 $\overline{S}5$	$\overline{S}1$ S2 S3 S4 S5
N3 Faulty	$\overline{S}1$ S2 S3 S4 S5	$\overline{S}1$ S2 S3 S4 S5	$\overline{S}1\overline{S}2\overline{S}3\overline{S}4\overline{S}5$	$\overline{S}1$ S2 S3 S4 S5
N4 Faulty	$\overline{S}1$ S2 S3 S4 S5	$\overline{S}1$ S2 S3 S4 S5	S1 $\overline{S}2$ S3 $\overline{S}4$ S5	S1 $\overline{S}2\overline{S}3\overline{S}4\overline{S}5$

Table 4.3 State of the control switches for different status of the module

processor out of the tree and the node is by-passed For a good node the controls are reversed, i.e., S1 is open and S2 through S5 are closed. So if the status is *Good* for a non-faulty node and $\overline{Good}$ for a faulty node, then the control for S1 is $\overline{Good}$ and that for S2 through S5 is *Good*

Node N3 also has five switches S1 through S5 The status of N1 N2 and N4 generate control for these switches The truth table with the status inputs and control outputs are shown in Table 4.4 A, B, C are the status of N1, N2 and N4 respectively We assumed 0 and 1 for good and faulty processor respectively Z1 through Z5 are the controls for S1 through S5 of N3

It has been seen that, status of N1 N2 and N4 are sufficient to generate controls for N3. Now, the question is can we generate these controls for N3 using the status of N2, N3 and N4 only (without using the the status of N1) The answer is NO for this particular switching scheme In this case N3 does not have the information about N1 So N3 has to assume N1 either good or faulty But in both the cases the assumptions can be wrong

A switching scheme is described which has very small additional hardware (a few logic gates and switches for each module) The scheme operates with local controls without any external supervision It is clear that the extra circuitry becomes the part of the node and does not have any effect on the layout complexity or degree of connectivity of the structure

The status bit register and the switching circuitry are hard-core part of the node It is assumed that the probability of these parts of the node going faulty is very small compared to that of the processor going faulty This is a reasonable assumption in the

A	B	C	Z1	Z2	Z3	Z4	Z5
0	0	0	0	0	0	0	0
1	0	0	1	0	0	1	1
0	1	0	1	1	0	1	0
0	0	1	1	0	1	0	1

From the table, after simplification.

$\overline{Z1} = \overline{A} \cdot \overline{B} \cdot \overline{C}$

$Z2 = \overline{A} \cdot B \cdot \overline{C}$

$Z3 = \overline{A} \cdot \overline{B} \cdot C$

$Z4 = \overline{A} \cdot B$

$Z5 = \overline{B}(A \cdot \overline{C} + \overline{A} \cdot C)$.

Table 4.4 Controls for the switches connecting Node 3.

fault-tolerant design approach.

An important issue is the revival of the data from a faulty node. This is also an open problem However, the local restructuring scheme should be convenient for data shifting This involves the shifting of data only within the module Whereas in the global restructuring scheme the data should be moving until an empty node is available For example, in RAE scheme the data should be moving 2^i places, in the worst case, for any i-th level

5. Evaluation of the System

So far the proposed system is described from the topological and structural point of view. In this chapter we evaluate the system and compare its performance with that of a non-redundant binary tree and of RAE scheme. Sec. 5.1 describes the reliability and some other related criteria of the modular tree system based on probabilistic models. Then the modular scheme is compared with RAE scheme in Sec. 5.2.

5.1 Reliability Analysis

One main objective of designing a fault-tolerant system is to improve the reliability of the system. In this section we will examine how far this goal has been achieved in case of the fault-tolerant modular binary tree architecture. The system reliability of the modular tree is estimated and compared with that of a non-redundant tree to see the improvement. Several other reliability related criteria are discussed later in this section. The results are compared with RAE scheme in the next section.

Probabilistic models for hardware evaluation are discussed extensively in [3]. We have used those models and definitions throughout the chapter for reliability analysis.

Probabilistic modeling based on relative component failure and repair rates is the most often used evaluation criteria for system reliability. Based on this model, the reliability function $R(t)$ of a system is mathematically defined as the probability that the system will perform satisfactorily from time zero to time t, given that operation commences successfully at time zero. It is monotonically decreasing function whose initial value is one. We have obtained an expression for the reliability function of a redundant system and used it to derive some other reliability measures.

Evaluation of the System

It is assumed throughout the rest of the section that the failure rate of links is negligible compared with nodes, thus only node failures are considered. Admittedly, this is an optimistic assumption, but it is supported by the fact that some of the link failures can be treated as failure of nodes those are connected by these links. Relatively simple reliability analysis of the topology is possible by this assumption.

5.1.1 System Reliability

The reliability function for a single component is defined in [13] as:

$$R(t) = \Pr\{\text{0 failures in time } (0,t)\}$$

$$= e^{-m(t)}$$

where $m(t) = \int_0^t z(x)dx$.

and $z(x)$ is the time-dependent failure rate called hazard function.

For a constant failure rate λ,

$$R(t) = e^{-\lambda t}$$

For a non-redundant system, every component must function properly for the system to work. If component failures are statistically independent, then the system reliability, in this case, is the product of the component reliabilities and is thus also exponential. Therefore, for n components,

$$R_{sys}(t) = \prod_{i=1}^{n} R_i(t)$$

$$= \prod_{i=1}^{n} e^{-\lambda_i t}$$

$$= e^{-(\sum_{i=1}^{n} \lambda_i)t}$$

Evaluation of the System

In a non-redundant binary tree with 'l' levels, there are $2^l - 1$ nodes. The reliability of each node is the same and equal to λ. So, the system reliability of such a tree is

$$R_{nr}(t) = \prod_{i=1}^{2^l - 1} R_i(t)$$

$$= R^{2^l - 1} \tag{5.1}$$

by letting $R = R_i(t)$ for all i.

The modular binary tree structure with spares can be considered as a fault-tolerant system consisting of a series of homogeneous subsystems S_p , p=1,2, ... M. A subsystem contains some active nodes and a spare node. For the modular system, each module with three nodes and a spare is a subsystem. We now analyze this fault-tolerant system to derive an expression for the reliability of the system in terms of the node reliabilities.

In the redundant tree structure, each subsystem can tolerate a single failure. In general, if the subsystem contains 'x' nodes and a spare, there cannot be more than one node failure in the subsystem. So, the reliability of the subsystem is

$$R_{sub} = R^{x+1} + (x+1)R^x(1 - R) \tag{5.2}$$

All the subsystems must be working for the system to be operating. Thus the system reliability is

$$R_{sys} = \prod_{p=1}^{M} R^{x+1} + (x + 1)R^x(1 - R) \tag{5.3}$$

Coverage is a concept most often used in reliability modeling of redundant systems. In its quantitative sense, coverage is the probability that a particular class of fault is

successfully detected before a complete system corruption occurs. So, taking coverage factor into consideration, the system reliability is given by:

$$R_{sys} = \prod_{p=1}^{M} [R^{x+1} + (xc + 1)R^x(1 - R)]. \tag{5.4}$$

For the modular binary tree, x, the number of active nodes is three. So eqn. 5.4 becomes

$$R_{sys} = R^4 + (3c + 1)R^3(1 - R)^M \tag{5.5}$$

However, eqn. 5.4 can be used to compute the system reliability of any other extended modular scheme and also for other redundant tree systems

The units are normalized such that the dimension of product of failure rate and time, i.e λt, is unity. Typically, failure rates are per million hours and thus time is in millions of hours

As an example, we consider a 4-level system for reliability computation. The system reliability of the 4-level non-redundant binary tree for a typical value of node failure is calculated using eqn. 5.1 and shown in Table 5.1. Whereas Table 5.2 gives the system reliability for the 4-level modular redundant binary tree with the same value of node failure rate and different values of coverage. It is assumed that $\lambda = \mu = 0.1$, where μ is the constant repair rate. The effect of coverage on system reliability is shown as a plot in Fig. 5.1. The significant improvement on system reliability is obvious from this plotting.

Evaluation of the system

t	R_{sys}
0.00	1.0000
0.05	0.9275
0.10	0.8600
0.20	0.7397
0.30	0.6371
0.40	0.5480
0.50	0.4721
0.60	0 4061
0.70	0.3499
0.80	0.3011
0.90	0.2591
1 00	0.2229

Table 5.1 Reliability of a four level non-redundant system with $\lambda = \mu = 0.1$

t	R_{sys} for c=1	R_{sys} for c=0 99	R_{sys} for c=0 98	R_{sys} for c=0 95	R_{sys} for c=0 90
0.00	1 0000	1 0000	1.0000	1 0000	1.0000
0 05	0 9990	0 9985	0.9975	0 9955	0.9915
0.10	0 9970	0 9955	0.9940	0.9895	0 9826
0 20	0 9885	0 9855	0 9826	0 9749	0 9611
0.30	0 9747	0 9708	0.9620	0 9553	0.9357
0 40	0 9567	0 9519	0.9466	0 9319	0 9067
0 50	0 9347	0 9286	0 9229	0 9053	0 8765
0 60	0 9094	0 9053	0.8956	0.8761	0.8437
0 70	0 8806	0 8734	0 8662	0 8442	0.8090
0 80	0 8499	0 8420	0.8342	0 8107	0 7733
0 90	0 8170	0 8090	0.8006	0 7762	0 7366
1 00	0 7832	0 7746	0.7656	0 7405	0.6998

Table 5.2 Reliability of a 4-level modular system with $\lambda = \mu = 0\ 1$

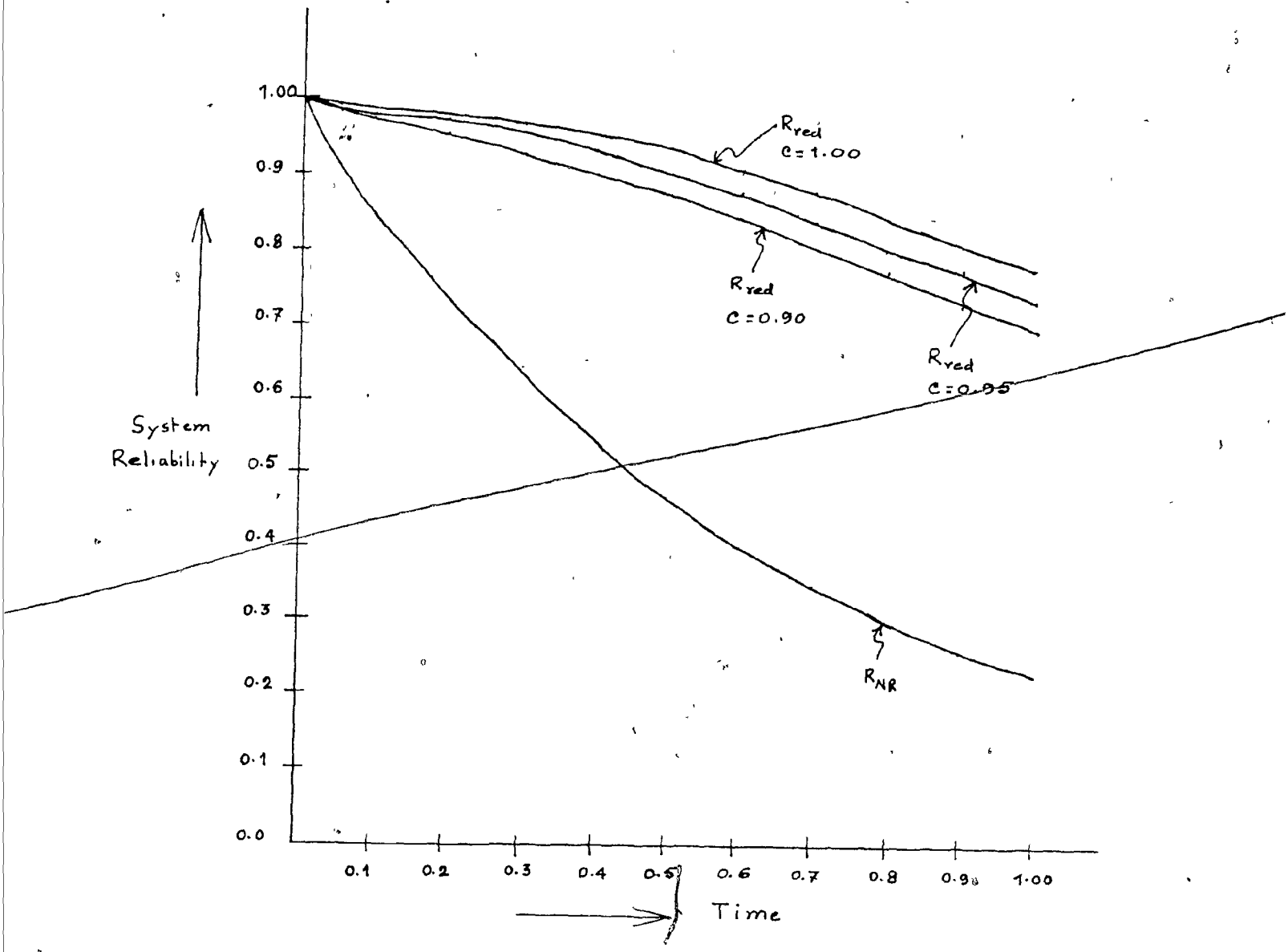

Fig. 5.1 System reliability of a four-level tree with different coverages.

5.1.2 Reliability Improvement Factor

A convenient way to compare the reliability of the redundant tree system with that of non-redundant tree system is by evaluating the reliability improvement factor, which is defined in [9] as

$$RIF = \frac{1 - R_{nr}}{1 - R_{red}} \tag{5.6}$$

where R_{nr} = reliability of the non-redundant system

and R_{red} = reliability of the redundant system

The reliability improvement factor achieved by the redundant system with coverage factor of 1.00 (perfect coverage), 0.99 and 0.98 are shown in Table 5.3.

5.1.3 Mission Time

Another criteria to study a redundant binary tree is the mission time. The mission time function MT(r) gives the time at which system reliability falls below the level r. The relationship between R(t) and MT(r) is given by

$$R[MT(r)] = r$$

$$MT[R(t)] = t$$

For a constant failure rate λ,

$$r = e^{-\lambda t}$$

So, the component mission time function is

Coverage	t	R_{sys}	RIF
1	0.05	0 9990	72 5
	0 10	0.9970	46 66
	0 20	0 9885	22 63
	0 30	0 9747	14 34
	0.40	0 9567	10 43
	0 50	0 9347	8 08
0.99	0 05	0 9985	48 33
	0 10	0 9955	31 11
	0 20	0 9855	17 95
	0 30	0.9708	12 42
	0 40	0 9519	9 39
	0 50	0 9286	7 13
0.98	0.05	0.9975	29 00
	0.10	0.9940	23 33
	0.20	0 9826	14 96
	0.30	0 9620	9 55
	0.40	0.9466	8 46
	0.50	0.9229	6.84

Table 5.3 RIF of a 4-level modular system with $\lambda = \mu = 0.1$.

$$MT(r) = \frac{ln(r)}{\prod_{i=1}^{n} \lambda_i} \tag{5.7}$$

However, for a redundant tree system we have

$$R_{sys} = \prod_{i}^{M} R^{i-1} + (i-1)R^{i}(1-R)$$

and

$$R(t) = e^{-\lambda t}$$

For a given value of R the time t which is the mission time MT r for the redundant tree system can be calculated combining these two eqns. The details of the procedure is given in Appendix A

The improvement in mission time for a redundant tree system is obtained by evaluating mission time improvement factor (MTIF)

MTIF is defined as

$$MTIF = \frac{MT(r)_{red}}{MT(r)_{nr}}$$

where $MT(r)_{red}$ and $MT(r)_{nr}$ are the mission time of the redundant and non-redundant tree system respectively

The mission time and mission time improvement factors for a four-level modular binary tree with coverage factor of 1.00 and 0.98 are shown in Table 5.4

5.1.4 Mean Time To Failure

The MTTF of a system is the expected time of the first failure in a population of identical systems given successful start up at time zero. It assumes a perfect system at

R(t)	Coverage=1 00		Coverage=0 98	
	MT	MTIF	MT	MTIF
0 80	0 7759	5 2153	0 7445	5 0067
0 90	0 3098	7 2571	0 4793	6 8276
0 95	0 3449	10 0790	0 3152	9 2163
0 96	0 3053	11 2036	0 2759	10 1433
0 98	0 2106	15 6214	0 1821	13 5895
0 99	0 1465	21 6695	0 1191	17 7761

Table 5.4 Mission Time and Mission Time Improvement Factor of a 4-level modular system with $\lambda = \mu = 0.1$

time zero For the reliability functions used earlier, the MTTF is defined as

$$MTTF = \int_0^\infty R(t)dt \tag{5.8}$$

For a non-redundant tree system with n nodes each with individual constant failure rate, λ_i

$$MTTF = \int_0^\infty R(t)dt$$

$$= \int_0^\infty e^{-(\prod_{i=1}^{n} \lambda_i)t} dt$$

Hence,

$$MTTF = \frac{1}{\prod_{i=1}^{n} \lambda_i}$$

Eqn. 5.8 is equally applicable for the redundant binary tree system. The details of the procedure is given in Appendix B

The mean time to first failure of the four-level modular redundant binary tree is 2.03 time units, compared to 0.6667, which is the corresponding figure for four-level non-redundant binary tree

5.2 Comparative Estimation of System Reliability

In the preceding sections, we have evaluated the modular binary tree quantitatively based on probabilistic models As expected, the redundant tree system turns out to be much more reliable than its non-redundant counterpart Improved system reliability and MTTF together with high values of RIF and MTIF stand in favor of the redundant

tree system The modular tree system is found to be better than the non-redundant tree system in every term of comparison

In chapter 2 we have described the RAE fault-tolerant binary tree In [9] Raghavendra et al have done similar type of reliability estimation for their tree architecture as we have done in this chapter In this section we will compare the modular binary tree system with RAE scheme

RAE have designed a fault-tolerant tree system which uses one spare node for each level of the binary tree This is the *one spare per level* scheme They have extended this scheme using one spare for every 2^i processors of the tree for some value of i The extended scheme with maximum number of spares is the one with one spare for every pair of active nodes This is the *one spare per pair scheme*

The reliability figures of a four-level binary tree system designed by using 'one spare per level' scheme modular scheme and 'one spare per pair' scheme are presented in Table 5 5 It appears from the tabulated results that the modular tree is more reliable than 'one spare per level' tree and 'one spare per pair' tree is more reliable than the modular tree How do we explain this fact ? Eqn 5 3 shows that the reliability model is based on the number of spares used in the system Although system reliability is not a linear function of the spares in the system it increases with the increased number of spares in the system Since, 'one spare per pair' tree uses more spares than the modular tree, the reliability figures are better in the former scheme For the same reason, the reliability figures are better for the modular scheme than that for 'one spare per level' scheme. Thus, these reliability figures fail to make a valid and justified comparison of these schemes Instead of comparing the effectiveness of the two schemes, the results only show that the system reliability increases with the increase in number of spares used in the system

t	R_{modular}	$R_{\text{one spare per level}}$	$R_{\text{one spare per pair}}$
0.00	1.0000	1.0000	1.0000
0.05	0.9990	0.9988	0.9995
0.10	0.9970	0.9952	0.9978
0.20	0.9885	0.9812	0.9915
0.30	0.9747	0.9613	0.9813
0.40	0.9567	0.9033	0.9675

Table 5.5 Reliability of a 4-level system using different schemes with $c=1$ and $\lambda = \mu = 0.1$

Evaluation of the system

Instead we plotted the *system reliability curve* with the *number of spares* vs *reliability* to make a better comparison of the schemes. The idea is to consider any n level binary tree and construct the tree using the two schemes. In RAE scheme the tree can be constructed out of different modules like one spare per level, one spare per four, one spare per pair etc. In the modular scheme basic module three-level module four-level module and even higher level module or a mixture thereof can be used to construct the n level binary tree. The use of different type of modules for the same system leads to the use of different number of spares and correspondingly different system reliability for the same n level binary tree. These different number of spares are plotted against the corresponding system reliability to get the *reliability curve*. The nature and relative position of the two reliability curves for the two schemes compare these schemes in the true sense of term.

Three such set of curves are plotted in Fig. 5.2 through Fig. 5.4 for comparing the two schemes. The relevant data are given in Table 5.6. The curves in Fig. 5.4 correspond to a four-level binary tree with perfect coverage and time equal to 0.5 million hours. RAE tree has been constructed out of 'one spare per pair', 'one spare per four' and 'one spare per eight' modules. Whereas the modular tree have used the basic module, three-level module and four-level module. The results for six level and eight level trees, are plotted in Fig. 5.3 and Fig. 5.1 respectively. The fact that the modular reliability curve lies above the RAE reliability curve in all the three cases shows that the modular scheme is more reliable than RAE scheme while the same number of spares are used for the two cases. Clearly this is an even comparison which shows the advantage of the modular scheme.

Scheme used	Module used	No. of Spares	R_{sys}
Modular Scheme	2-M	5	0.9349
	3-M	2	0.8499
	4-M	1	0.8179
RAE Scheme	1-2	8	0.9504
	1-2-4	5	0.9280
	1-2-4-8	4	0.9033

Table 5.6a Data for plotting the reliability curves of a 4-level system with c=1 and t=0.5

Scheme used	Module used	No. of Spares	R_{sys}
Modular Scheme	2-M	21	0.8814
	3-M	9	0.7137
	(2-M)-(3-M)	5	0.5720
RAE Scheme	1-2	32	0.8684
	1-2-4	17	0.8020
	1-2-4-8	10	0.7043
	1-2-4-8-16	7	0.5906

Table 5.6b Data for plotting the reliability curves of a 6-level system with c=1 and t=0.4

Scheme used	Module used	No. of spares	R_{sys}
Modular Scheme	2-M	85	0.6510
	(2-M)+(3-M)	37	0.4513
	4-M	17	0.2436
RAE Scheme	1+2	128	0.7207
	1+2+4	65	0.5921
	1+2+4+8	34	0.4179
	1+2+4+16	19	0.2390

Table 5.6c Data for plotting the reliability curves of an 8-level system with c=1 and t=0.3

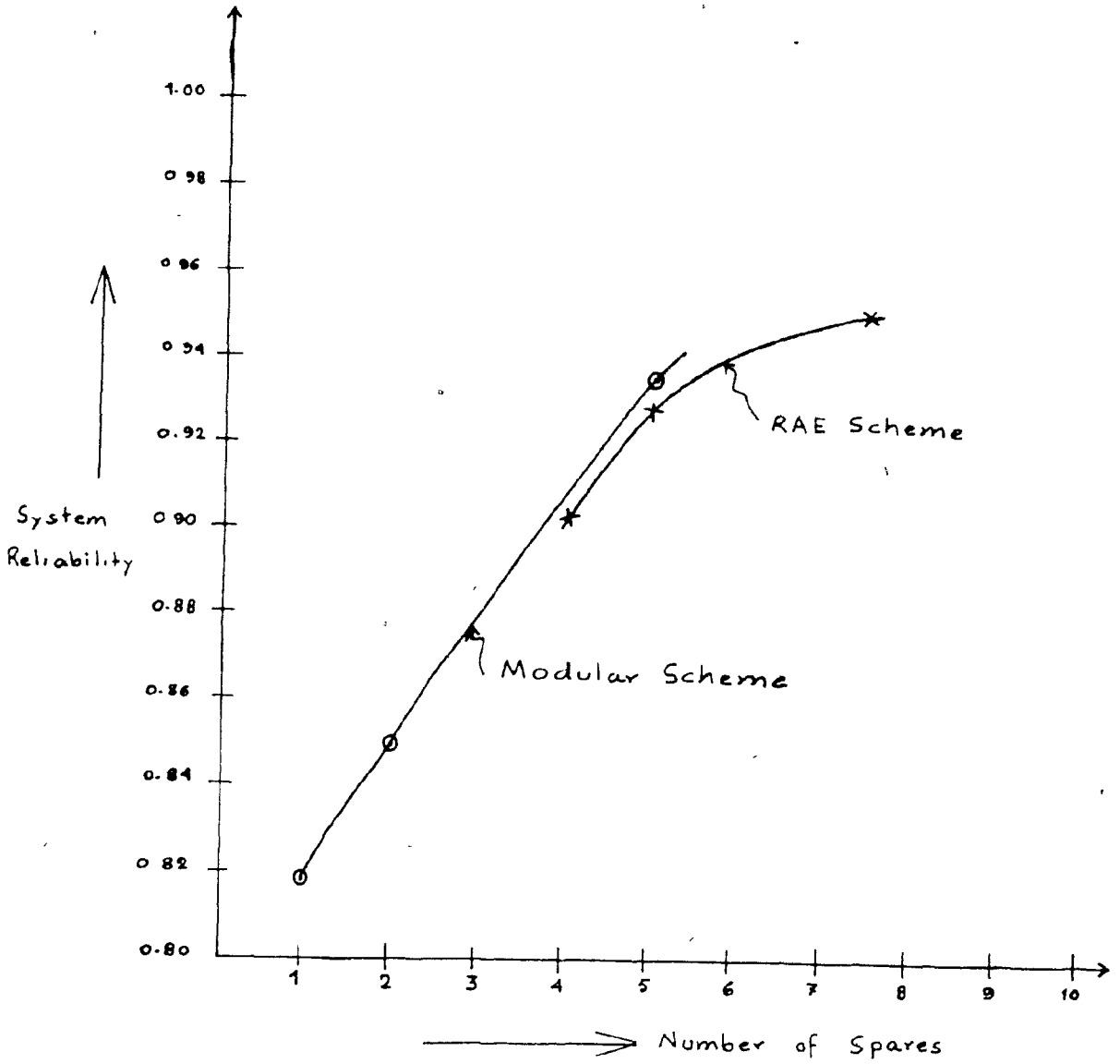

Fig. 5.2 Reliability curves for a four-level binary tree.

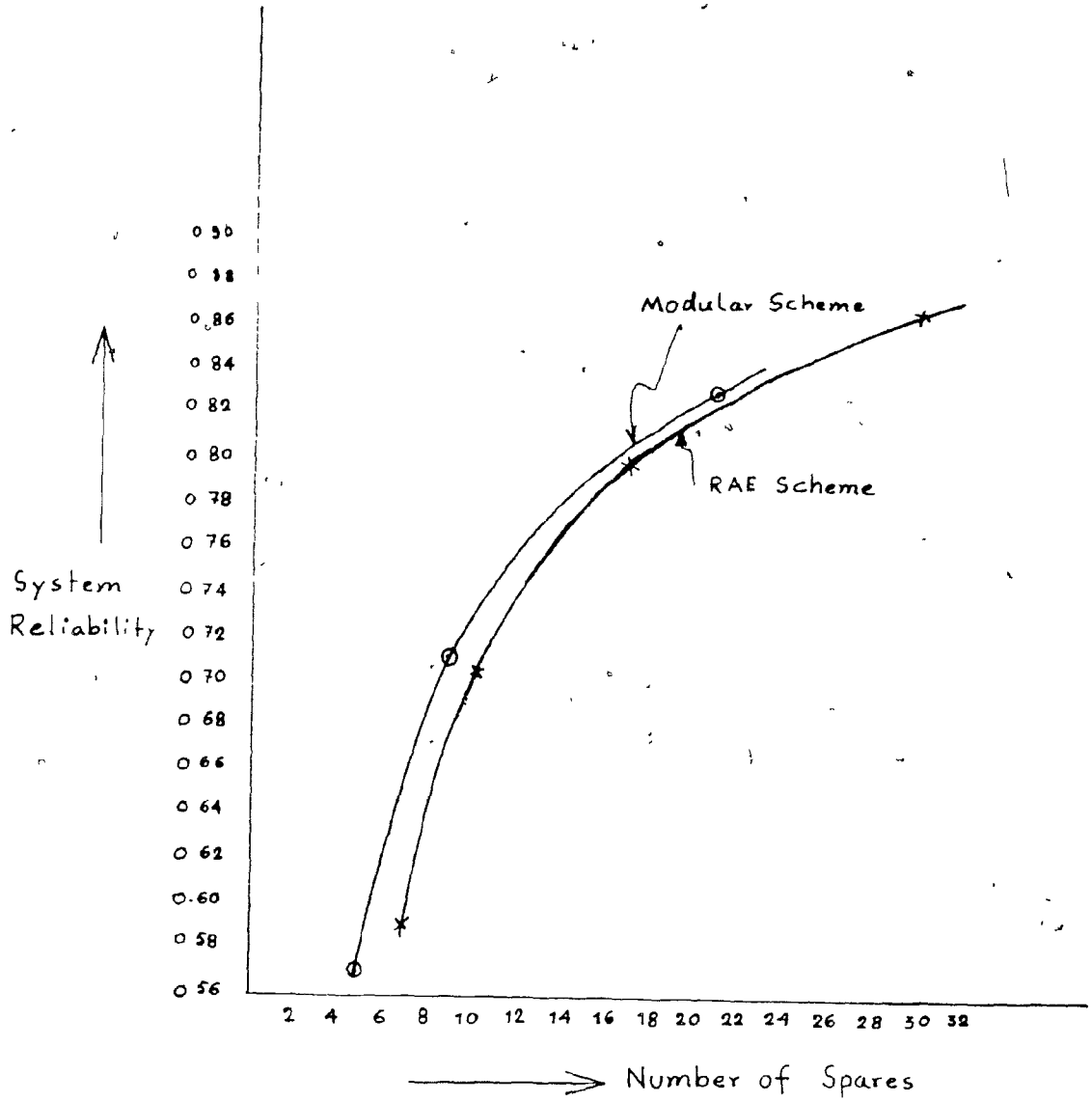

Fig. 5.3 Reliability curves for a six-level binary tree.

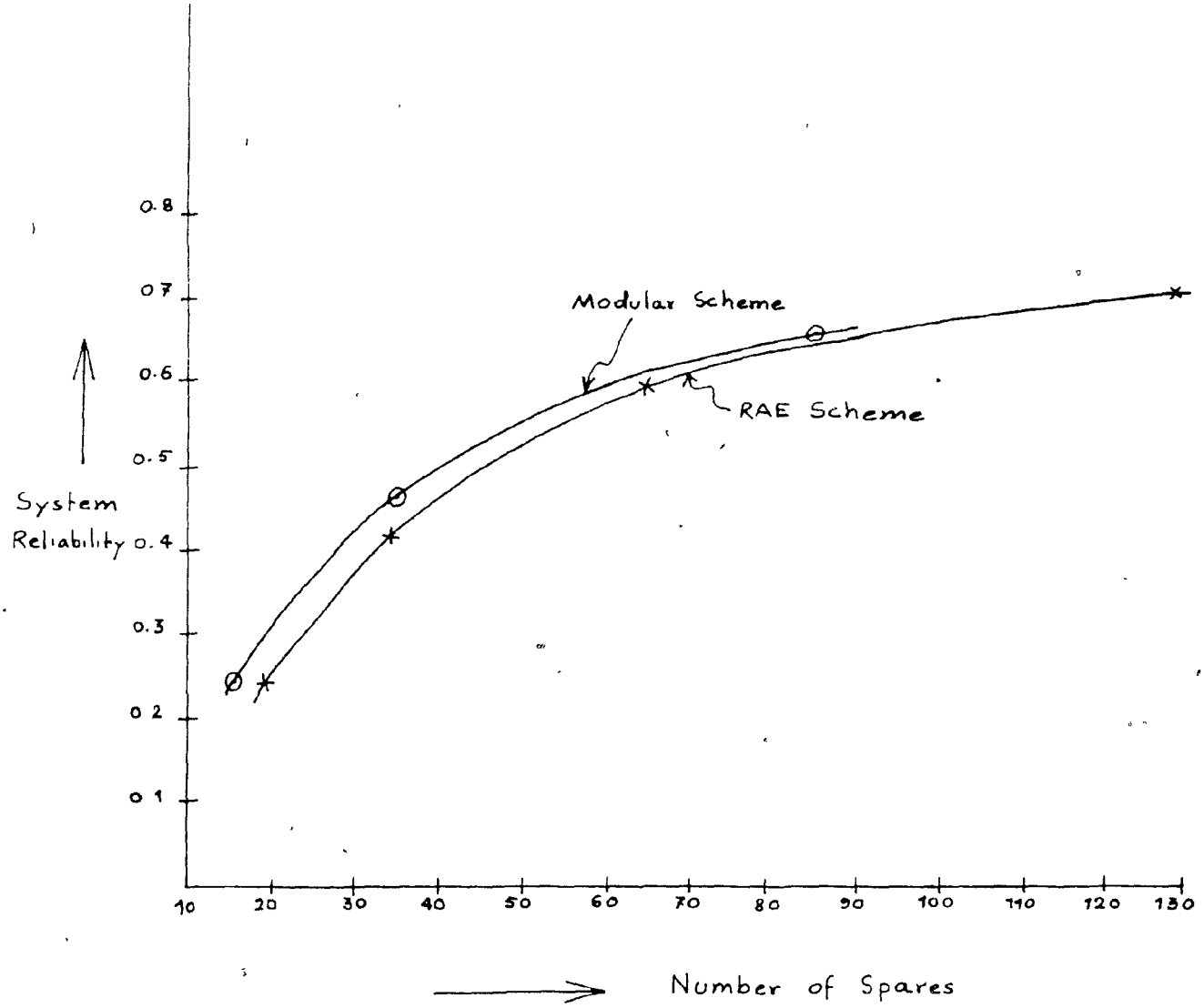

Fig. 5.4 Reliability curves for an eight-level binary tree

6. Conclusion

A modular fault-tolerant scheme is proposed in this thesis for the binary tree architecture. Repetition of the fault-tolerant module as many times as required gives the fault-tolerant binary tree. The module interconnection is so easy and straight forward that partitioning of the tree among several chips becomes very convenient. An H-tree like layout is described for this modular scheme. The layout structure is conveniently extensible and well-suited for VLSI design methodology. The layout is optimized in terms of area and time. Detailed analysis of the performance and reliability of the fault-tolerant tree has been studied.

The modular scheme is compared with RAE scheme in different terms. First of all, the link ratio, which is a measure of complexity, has significantly improved for the modular scheme. Secondly, the modular structure, due to its regularity, is more convenient for increasing the tree depth and for layout than RAE scheme. RAE scheme requires more layout area and tree depth cannot be increased simply by adding extra module or chip. Thirdly, for same amount of redundancy, the modular scheme is shown to be more reliable than RAE scheme. Fourthly, the modular scheme has a local restructuring scheme which does not require any external control as in case of the decoupling networks in RAE scheme.

The discussion is mainly based on a two-level module. The extension of the module is also shown here. A three-level, four-level or even higher level module can be constructed by using the basic module. Although the fault-tolerant capability reduces with the higher level modules, these can be used in systems where fault-tolerance can be traded for cost.

However, one very significant issue of fault-tolerance is still remained unsolved. That is the revival of data from the faulty element. To our knowledge, no scheme or

paper, so far, has discussed this problem. But it is very much essential that the data in the faulty node should be recovered and stored in the replacing node before the tree begins to function again. This is an open problem of interest. It seems that the modular scheme will be advantageous for such a data-recovery procedure. The modular scheme treats the fault locally and hence, data recovering and shifting would be restricted within the faulty module. But, in most of the existing schemes data-shifting would be rigorous to restore the previous data structure.

A method of testing and fault diagnosis has not been worked out for the modular scheme. Although the existing standard methods can serve the purpose, a method which does the testing and reconfiguration without any interruption in the normal operation should be of interest.

In the modular scheme there are four nodes N1 through N4. Out of these, N1, N2 and N4 are identical in terms of node complexity, restructuring scheme etc. Remaining N3 is more complex than the rest. It has to replace any one of the other three nodes depending upon the fault condition. That is why it has higher degree of complexity and it requires more controls for restructuring. The fact that the module as a whole is considered as the building block for the tree, N4 does not break the regularity of the structure. The complete binary tree is the regular repetition of the same block. However, it would have been more convenient if the block itself were more regular in structure in terms of the four nodes.

In some of the existing schemes, the root node of the binary tree is the bottleneck. It is found that the entire tree becomes inoperative when the root fails. So, some schemes have to consider the root node to be specially designed and protected. Other schemes provide an extra redundant node to take care of the root. The modular scheme does not suffer from this drawback. The failure of the root node is nothing but a normal fault

in the highest level module and the faulty module is readjusted accordingly However, some binary tree machines like dictionary machine require the root node to serve as the global controller for the tree In that case there is no choice but to design a special purpose node.

In fault-tolerance design methodology it is a usual practice to consider the possibility of link failure to be negligible compared to that of node failure Also, some link failures can be considered to be equivalent to the failure of one of the corresponding link connected nodes In the modular scheme all the link failures have equivalent node failure representation excepting L1 (See Fig 3 1) Notice that L1 connects a module to another module Thus the failure of L1 can neither be treated as the failure of N1 nor be transferred to the connected module Thus link L1 of every module is very sensitive We can think of some special protection or duplication of L1 in every module.

The results of comparison show that the proposed fault-tolerant scheme is more reliable, modular and easier to implement than the existing fault-tolerant schemes However, the scheme is not free from shortcomings But the advantages it provides are sure to outplay the accompanying drawbacks We hope further works and developments on this scheme will make it more attractive and useful as a fault-tolerant design approach

7. References

[1] Atallah, Mikhail J., and Kosaraju, S. Rao, *A Generalized Dictionary Machine for VLSI* The John Hopkins University, Department of Electrical Engineering and Computer Science, Baltimore, Maryland, 21218

[2] Chung, Fan R. K., Leighton, F. Thomson, and Rosenberg, Arnold L., *Diogenes: A Methodology for Designing Fault-Tolerant VLSI Processor Arrays* In Proceedings The Thirteenth International Conference on Fault-Tolerant Computing, Milano, Italy June 28-30, 1983, pp 232-239

[3] Fortes, J. A. B. and Raghavendra, C. S., *Dynamically Reconfigurable Fault-Tolerant Array Processors* In Proceedings The Fourteenth International Conference on Fault-Tolerant Computing, Florida, USA, June 20-22, 1984, pp 386-392.

[4] Grey, Bason O. A., Avizienis, Algirdas, and Rennels, David A., *A Fault-Tolerant Architecture for Network Storage Systems* In Proceedings. The Fourteenth International Conference on Fault-Tolerant Computing, Florida, USA, June 20-22, 1984, pp 232-239

[5] Hayes, J. P., *A Graph Model for Fault-Tolerant Computing Systems* IEEE Transactions on Computer, Vol C-25, No 9, September 1976, pp 875-884

[6] Kwan, C. L. and Toida, S., *Optimal Fault-Tolerant Realizations of Hierarchical Tree Systems*, Proceedings 1980 International Symposium on Fault-Tolerant Computing, Kyoto, October 1980, pp 176-178

[7] Leiserson, Charles E., *Area-Efficient VLSI Computation*. ACM Doctoral Dissertation Award 1982

References

[8] Ottmann, Thomas A., Rosenberg, Arnold L., and Stockmeyer, Larry J., *A Dictionary Machine (for VLSI)* IEEE Transactions on Computers, Vol. C-31, No. 9, September 1982, pp 892-898

[9] Raghavendra, C. S., Avizienis, A., and Ercegovac, M., *Fault-Tolerance in Binary Tree Architectures* IEEE Transactions on Computers, Vol. C-33, No 6, June 1984, pp 568-572

[10] Rosenberg, Arnold L., *The Diogenes Approach to Testable Fault-Tolerant Arrays of Processors* IEEE Transactions on Computers, Vol C-32, No 10, October 1983, pp 902-910

[11] Samatham, M. R., and Pradhan, D. K., *A Multiprocessor Network suitable for Single-Chip VLSI Implementation* In Proceedings International Computer Architecture Conference 1984, pp 328-337

[12] Somani, Arun K., and Agarwal, V. K., *An Efficient Unsorted Dictionary Machine* Accepted for publication in IEEE Transaction on Computers

[13] Swarz, R. S., and Siewiorek, D. P. *The Theory and Practice of Reliable System Design*

[14] Ullmann, Jeffery D., *Computational Aspects of VLSI*

Appendix A

Calculation of Mission Time MT(r) for a redundant binary tree system:

The mission time function MT(r) gives the time at which the system reliability falls below the level r

In chapter 5 the system reliability for a redundant system is given as:

$$R_{sys} = \prod_{p=0}^{M} [R^{x+1} + (xc - 1)R^x(1 - R)]$$

where R=node reliability=$e^{-\lambda t}$

x=number of active nodes in a sub-tree

M=number of subsystems in the tree.

c=coverage factor.

For finding the mission time of the system, some arbitrary value can be assigned to R_{sys} and the corresponding mission time is obtained by solving the above equation.

The equation simplifies to :

$$R_{sys} = \prod_{p=0}^{M} [R^{x+1} + (xc - 1)R^x(1 - R)]$$

$$= \prod_{p=0}^{M} [R^{x+1} + R^x(1 - R) + xcR^x(1 - R)]$$

$$= R^{2^n - 1} \prod_{p=0}^{M} [(xc - 1) - xcR]$$

For a modular four-level tree. M=5 and x=3 Assuming c=1 and $\lambda = 0.1$

$$R_{sys} = R^{15}(4 - 3R)^5$$

$$= e^{-1.5t}(4 - 3e^{-0.1t})^5$$

$$y = 1024e^{-1.5t} - 3840e^{-1.6t} + 5760e^{-1.7t} - 4320e^{-1.8t} + 1620e^{-1.9t} - 243e^{-2.0t}$$

Therefore,

$$R_{sys} = [1024y^{15} - 3840y^{16} + 5760y^{17} - 4320y^{18} + 1620y^{19} - 243y^{20}]$$

where $y = e^{-0.1t}$

Solving this polynomial eqn. for different values of R_{sys} gives the corresponding value of mission time.

Appendix B

Calculation of Mean Time to Failure for a redundant binary tree

From eqn. 5.8,

$$MTTF = \int_0^\infty R(t)dt.$$

For a redundant binary tree

$$R(t) = \prod_{p=0}^{M} [R^{x+1} + (xc+1)R^x(1-R)]$$

From Appendix A, for a modular four level binary tree with c=1 and $\lambda = 0.1$,

$$R(t) = 1024e^{-1.5t} - 3840e^{-1.6t} + 5760e^{-1.7t} - 4320e^{-1.8t} + 1620e^{-1.9t} - 243e^{-2.0t}$$

Therefore,

$$MTTF = \int_0^\infty R(t)dt$$

$$= \frac{1024}{1.5} - \frac{3840}{1.6} + \frac{5760}{1.7} - \frac{4320}{1.8} + \frac{1620}{1.9} - \frac{243}{2.0}$$

$$= 2.033539.$$