TOPOLOGICAL TESTING*

Miroslaw Malek, Antoine Mourad,
and Mihir Pandya**

Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712-1188

TR-89-03 March 1989

* This research was supported in part by AT&T and by the Office of Naval Research under Grants N00014-86-K-0554 and N00014-88-K-0543.

** All of the authors are affiliated with: The Department of Electrical and Computer Engineering, The University of Texas at Austin, Austin, Texas 78712. Telephone: 512-471-5704, ARPANET: malek@emx.utexas.edu

Abstract

A new concept, *topological testing*, which uses graph optimization methods applicable to each level of system hierarchy is presented. Examples demonstrating orders of magnitude improvement in testing time are given.

Brief summary

A concept of *topological testing* is introduced and its applications are presented. Topological testing uses graph theoretic optimization methods such as the Traveling Salesman Problem, the Chinese Postman Problem, path covering and partitioning to minimize the test time. The topological testing techniques can be applied to each level of system hierarchy, namely, circuit, logic, register transfer, instruction and processor-memory-switch levels. Specifically, the topological testing approach is demonstrated by developing tests for the multistage interconnection network and the hypercube network. Time optimization for the testing of these networks gives very promising results by taking advantage of inherent parallelism and removing test redundancy. Three orders of magnitude improvement is achieved by applying topological testing techniques to the testing of an existing multistage interconnection network.

Key Words : Testing, graph theory, optimization methods, multistage interconnection networks, hypercube.

1 Introduction

As digital systems become more and more complex, testing becomes even more time consuming process during manufacturing and maintenance. An exhaustive approach putting the system in every state and applying all possible inputs to it, leads to an exponential growth in the number of tests as the system complexity increases and hence the testing time becomes prohibitive. In this paper, we introduce a new approach called *topological testing* which applies primarily at the system integration level. The approach uses the "topology" of the system to minimize test time. By topology we mean a graph representation of the system reflecting either its physical layout or a logical description in the form of a Finite State Machine (FSM).

The approach is based on defining the objectives of the test and then trying to find an optimal way to perform the test in order to achieve the specified objectives. For example, if the nodes in the graph representation of the system are to be tested, then the optimal way of performing the test is by sending a testing packet along a shortest Hamiltonian cycle which traverses every node exactly once. If the graph does not have a Hamiltonian cycle then a shortest closed walk traversing every edge at least once should be found.

The target systems include multiprocessor architectures, microprocessors, distributed systems, communication networks as well as protocols.

The next section presents the system models and the domains of application of topological testing. Section 3 introduces the optimization methods that are typically used in this approach. Section 4 details an example where topological testing is applied to generate tests for the multistage interconnection network (MIN) and Section 5 provides another example in which the method is used to test the communication capabilities of a hypercube. Finally, Section 6 presents some conclusions.

2 System models

Three system models, where topological testing is applicable are identified :

- *Behavior:* the approach can be used to test the behavior of a system modeled by a finite state machine. Testing protocol conformance [1, 2] is one example in this area.
- *Organization:* the organization or topology of a system will be taken into account in the test generation process in order to minimize the

number of tests or the distance traveled by the testing packets. Section 4 provides a detailed illustration of this approach through the testing of the multistage interconnection network.

- *Hierarchy:* the objective here is to partition the system into subsets that can be tested concurrently. In both homogeneous and heterogeneous environments, a hierarchical approach can be used to parallelize the testing procedure, hence reducing the test time. Section 5 shows how partitioning can be effectively used in generating tests for the hypercube network.

Topological testing can be applied at different levels in a computer system. It can be used at the circuit, logic, register transfer, instruction and processor-memory-switch (PMS) levels. However, it is most useful at the system integration level, when different components are put together and a test has to be conducted to ensure that the system is operational. In Sections 4 and 5, we detail two examples of application of topological testing at the PMS level. The application of the method at the register transfer level (RTL) can be illustrated by the model developed by Thatte and Abraham in [4] for microprocessor testing where the system is described by a graph in which nodes are registers and arcs represent dataflow between registers. In this case, the test generation process involves traversing every arc in the graph using a minimal number of tests. This problem can be solved using the path-covering technique discussed in the next section. In the paper, we will be concerned only with finding an "optimal" way for traversing the graph modeling the system. We will not discuss what instruction or sequence of instructions will produce the transition from a given node to the other or the traversal of a given edge. This particular problem is discussed in [4] for microprocessor testing and in [1] for protocol conformance testing. Examples of applications of topological testing at the logic level are mainly in the area of parallel testing of VLSI circuits [5]. The Euclidian test method for semiconductor random access memories [6] also falls in the category of topological testing.

The fault models will be specific to each system and the level of testing to be achieved.

Figure 1: a) graph with Euclidian costs b) graph with non Euclidian costs

3 Methods of topological testing

Topological testing makes use of optimization methods to find a minimal test sequence and eliminate redundancy in testing. As shown previously, Hamiltonian circuits can be used to send a packet to test every node in the system. When the distance between the nodes or the cost of traversing an edge is not uniform, then the Traveling Salesman Problem (TSP) [7] can be used to find the Hamiltonian cycle with the minimal cost. In the case where no Hamiltonian exists in the system graph or when the cost on the edges is not equal to the Euclidian distance, a variation of the TSP, called Traveling Salesman Problem with Repetition of Cities [7], can be used. It replaces the graph by another complete graph with the same nodes but the cost on the edges is the length of the shortest path in the original graph between the nodes on which that edge is incident. Figure 1 gives two examples of graphs where the nodes are to be tested. In a) the costs satisfy the Euclidian property and hence the shortest Hamiltonian (ABCDA) cycle is the least cost tour. In b) the costs do not satisfy the Euclidian property and the least cost tour (ABDCDBA) is not a Hamiltonian cycle.

In the case where both links and nodes in the graph have to be tested, an Eulerian circuit can be used to traverse all edges of the graph exactly once. Also in the case of behavioral models, such as Finite State Machines, an Eulerian can be used to perform all transitions without repetition if the graph is symmetric. In the case where no Eulerian exists, the Chinese Postman Problem (CPP) [1, 2, 3] can be used to find a minimal cost cycle that traverses every edge at least once.

The above methods assume that the system graph is strongly connected but that is not always the case. Some models use digraphs with one or more source nodes and one or more sink nodes with unidirectional links connecting the source nodes to the internal nodes, the internal nodes to each other and the internal nodes to the sink nodes. The flow graph of a program is an

example of such graphs. In this case, the problem of traversing all arcs and nodes is solved by finding a *path-covering* of the graph, i.e., a minimal number of paths from source nodes to sink nodes that cover every arc in the graph. This number of paths was also proposed as a measure of program complexity in [8].

Partitioning may also be successfully applied in cases where parallel testing is possible. This is also one of the most powerful ways to reduce test time. Different types of parallel testing can be used. In a multiprocessor system, the interconnection network can be tested by having the processors send testing packets in parallel through different paths in the network. In a distributed system or a large communication network, the graph can be partitioned and a hierarchical test can be performed where the partitions are tested concurrently using one of the methods mentioned above. Figure 2 provides a graphical illustration of the different aspects of topological testing in the form of a Rubik's cube. The three dimensions are the level at which the system is tested (logic, RTL, PMS, ...), the type of representation of the system (behavior, organization, hierarchy) and the graph methods that are used in the test generation (Eulerian, Hamiltonian, partitioning, ...). Almost any combination of these concepts can be used in practice to model a system and generate tests for it.

4 An example using behavior and organization: testing of multistage interconnection networks

4.1 Motivation

In this section, we propose to apply the concepts of topological testing to reduce the test time of multistage interconnection networks. The approach uses the *behavior* of the switches and the *organization* of the network in the test generation process.

Multistage interconnection networks became popular with the advent of multiprocessor systems. But testing these networks, especially in the case of packet switching, remains a problem. For example, the testing of the multistage interconnection network of the Sigma I dataflow computer being developed at Electrotechnical Laboratories in Japan takes over twenty-two hours, [9].

The multiprocessor system is assumed to be synchronous and packet-switching is used. The objective of the test will be to detect stuck-at faults

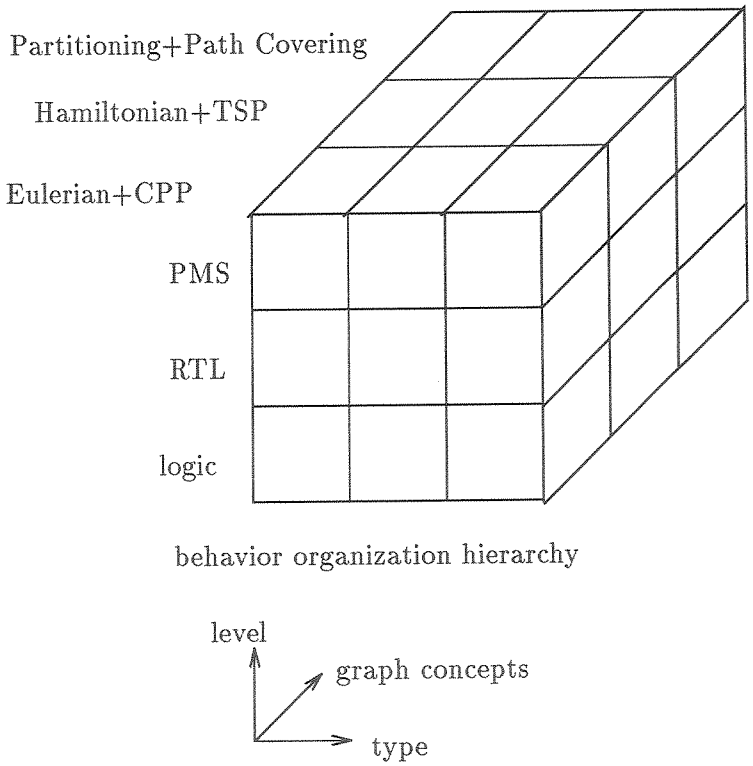

Figure 2: The Rubik's cube of topological testing.

and bridge faults in the data, routing and control lines of the network.

Some work has already been done in this direction in [11] where test techniques for detecting and locating faults in the data paths have been developed. In this study, we concentrate on testing routing, control and priority. We will use the rectangular SW-banyan, [10], to demonstrate our methods. Most of the results can be extended to other networks.

4.2 Description and properties of the network

The number of stages in the network is denoted by L. Each switch has f inputs and f outputs. The number of computers is $N = f^L$. Each connection between a computer and a switch or between two switches in the network includes a number of data lines, usually 8, a parity line, a data-valid (DV) line and a clear-to-send (CTS) line. Figure 3 shows the architecture of the network.

The graph representation of a rectangular SW-banyan contains a Hamiltonian circuit and an efficient way for constructing such a circuit was shown in [12]. Also since every node in the graph representation has an even degree, an Eulerian circuit can be constructed. These circuits can be used to implement serial on-line tests for detection of stuck-at faults.

It was also shown in [11] that f pairwise edge-disjoint test graphs, each with f^L disjoint paths between pairs of computers can be constructed. These test graphs contain every edge of the graph exactly once and can be used to apply N tests in parallel. Figure 4 shows the edge-disjoint test graphs that can be constructed in the network of Figure 3.

4.3 Testing the data paths

Using the above result, it was shown that two parallel tests are sufficient to detect any number of stuck-at-α ($\alpha = 0, 1$) faults in the nodes of the graph and that $2f$ parallel tests are sufficient to detect any number of stuck-at-α faults in the edges of the graph. Also fault location was studied and efficient methods were developed for fault location.

4.4 Testing the routing capabilities

Each switch uses part of the incoming address, exactly $\log_2 f$ bits, to route the incoming packet to the correct output. Therefore only f patterns need to be applied to each input of the switch to test the routing capabilities. This can be done in parallel for all the inputs of the switch using a sequence

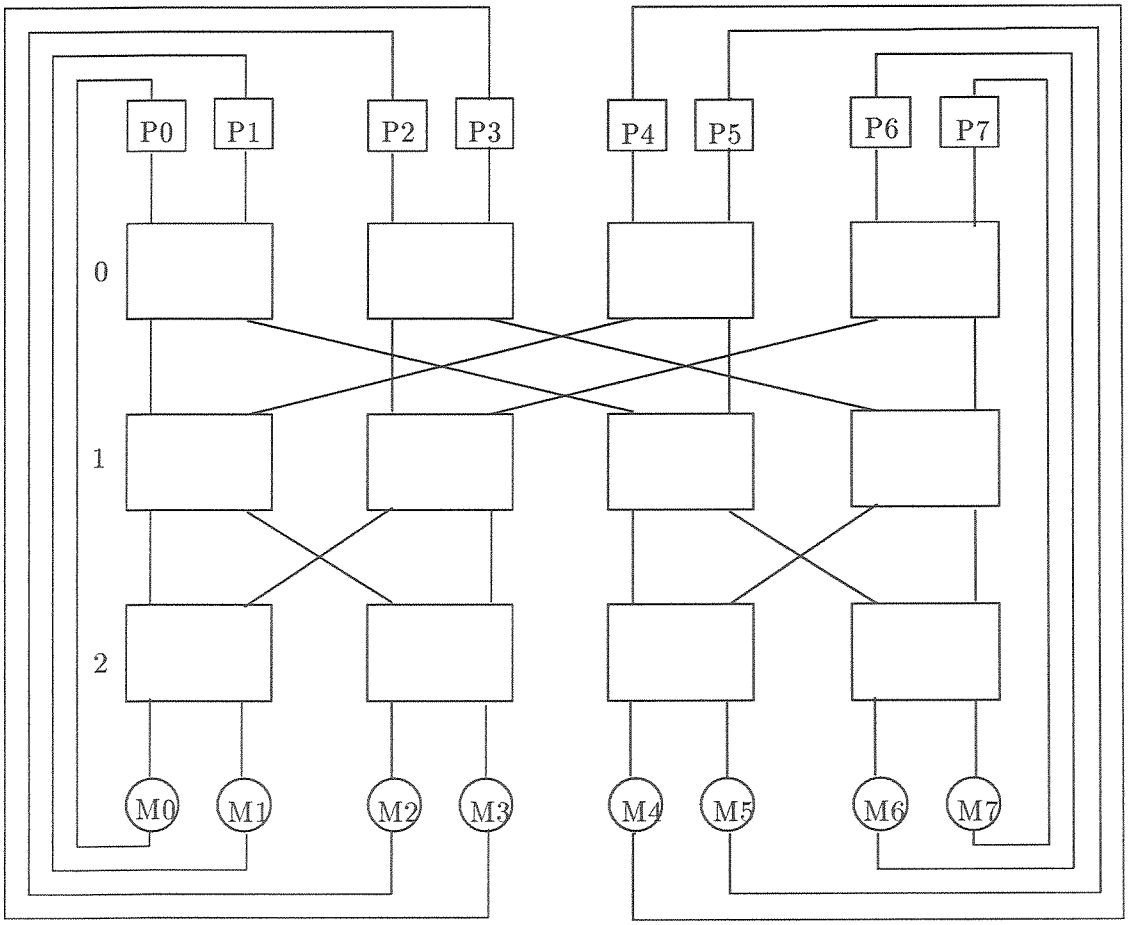

Figure 3: Example of banyan network with 2×2 switches and 3 levels.

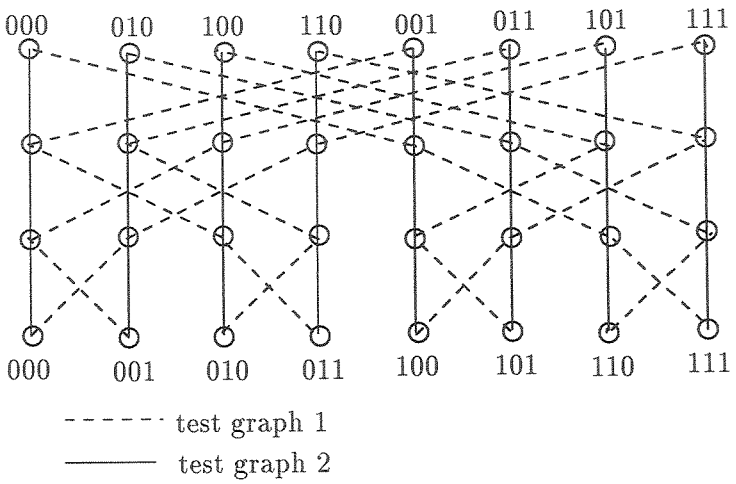

Figure 4: Graph model of the network and the f edge-disjoint test graphs.

of tests where the outputs corresponding to the inputs $1 \ldots f$ would be successively $1\,2 \ldots f, 2\,3 \ldots f\,1, \ldots, f\,1 \ldots f-1$. In other words, we perform all circular permutations of the outputs.

To test the whole network, the f edge-disjoint test graphs, mentioned above, can be used. It turns out that the intersection of each one of these test graphs with a switch corresponds to one of the above f patterns. Therefore, applying parallel tests successively along each of the f graphs would test routing on the whole network. A computer will decide that a routing fault has occurred if the address in the received packet is different from its own address or if it does not receive a total of f packets.

Theorem 4.1 *Only f tests are needed to detect any number of routing faults in the network.*

The proof of the theorem follows from the above.

4.5 Testing priority and control

The objective here is to test the correct behavior of each switch under any type of input pattern. The above tests for data and routing do not test the behavior of the switches under contention on one or more of their outputs. Also some stuck-at-1 faults and bridge faults on the control lines inside the switch would not be detected by the above tests. The priority scheme used in case of contention should also be tested. Both fixed priority and round-robin priority schemes will be considered in the proposed testing methods.

These methods make extensive use of the synchrony of the system especially for testing the behavior of the switches under contention.

4.5.1 Number of tests per switch

If an exhaustive approach applying all possible patterns is used, the number of tests to be applied to each switch would be:

$$m = \sum_{i=1}^{f} {}^{f}C_i f^i = (1+f)^f - 1$$

In the above formula i is the number of active input lines in the switch and ${}^{f}C_i$ is the number of different combinations of i active inputs out of the f inputs, f^i is the number of different choices of outputs for i active inputs. By active input, we mean an input with an active data valid (DV) line. Also, in the case of round-robin priority, each of the above tests should be repeated for every state of the arbitration units used to resolve the contentions that appear in the test. The above number of tests grows superexponentially with the number of inputs f. For a 4×4 switch, $m = 624$, for an 8×8 switch, $m = 43,046,720$.

In order to minimize the number of tests we need to consider the internal structure of the switch. As described in [13], the control part of an $f \times f$ switch comprises usually one routing and storage block at each input, called RSB, and one arbitration logic block (ALB) at each output. Every RSB is connected to every ALB with one SELECT line and one DESELECT line. The RSB is completely tested by the tests for data paths and routing. The SELECT and DESELECT lines and the ALB's remain to be tested. The SELECT line signals to the ALB that the corresponding RSB has a packet to send on tho output of the ALB. The DESELECT line signals to the RSB that the transmission is completed and it can receive another packet. Figure 5 shows the design of the switch.

Step 1. Testing of SELECT and DESELECT lines
The presence of stuck-at-1 faults of SELECT lines will produce the activation of the DV line at the output even if there are no messages in the switch. The DV signal will propagate through the network and produce a DV signal at one of the computers with an empty message, and therefore will be detected by the computer. The presence of stuck-at-0 faults of SELECT lines will be detected by a sequence of tests similar to the one used for testing stuck-at-α

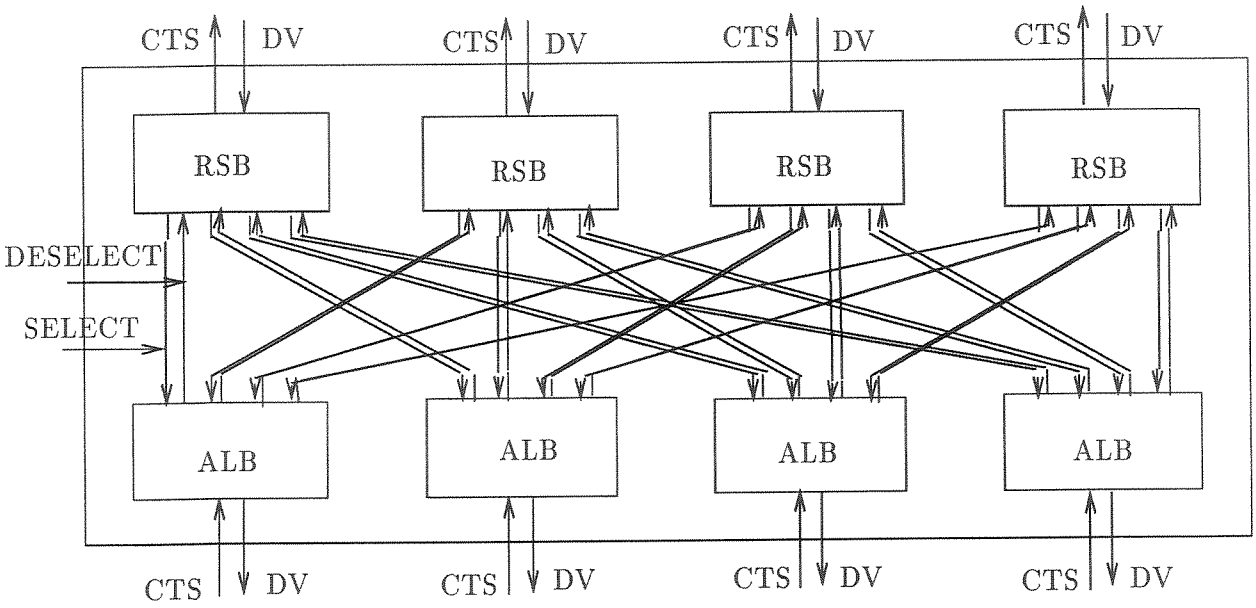

Figure 5: Design of the control part of a 4×4 switch.

edges in the data part. Basically, every input to output connection inside the switch should be requested. The only difference will be that in this case, faults will be detected if messages are not received at the other end of the network. Exactly f parallel tests will be needed to test the whole network for these types of faults. Stuck-at-0 faults of DESELECT lines will produce the blocking of all incoming messages and can be tested by the same tests as those for stuck-at-1 faults of SELECT lines. Stuck-at-1 faults of DESELECT lines will cause the RSB to accept new messages even when those stored in the switch have not been relayed yet. These faults can be tested by sending two consecutive packets, one with all 0's and one with all one's along every edge-disjoint test graph in the network. In all, $2f$ tests will be needed to completely test the network for these types of faults.

Bridge faults on SELECT lines can be tested by activating each line separately and then attempting to detect any activation of other lines through the propagation of DV signals with empty messages. The number of tests needed in this case is f^2. For bridge faults on DESELECT lines 2 tests are needed for each line, as in the stuck-at-1 case, but since the lines cannot be tested in parallel in this case, the total number of tests will be $2f^2$. Note that faults on DESELECT lines will not be handled by exhaustive testing

of all patterns because they involve interaction between stages.

Step 2. Testing the ALB's
In the case of fixed priority, the Arbitration Logic Block is simply a priority encoder. It can be tested with f tests. Assuming the priority decreases from input 1 to input f, the tests are: $11\ldots1$, $01\ldots1$, ..., $00\ldots01$. The f ALB's have to be tested serially. Therefore the total number of tests per switch would be f^2.

In the case of round-robin priority, the design of the ALB is more complex. A possible design is shown in [13]. The ALB implements a finite state machine with f states, each state corresponding to a particular setting of the relative priority of the inputs. From each state, 2^f possible transition arcs exist. Some of these transitions do not change the state. The finite state machine in question is completely symmetric and, therefore, has an Eulerian. The Eulerian can be used to generate an optimal sequence of tests for the ALB and the number of tests required is equal to the number of transition arcs in the finite state machine which is $f2^f$. This optimization can be carried a step further by applying some of the tests not involving all inputs of the switch, in parallel, to all or some of the ALB's. In any case, the number of tests needed for the f ALB's will be bounded by f^22^f. This number is still exponential with respect to the number of inputs, but it is several orders of magnitudes smaller than the number of tests required in the exhaustive method. For an 8×8 switch, $f^22^f = 16384$ while, in the exhaustive method, more than 43 million tests are needed. Figure 6 shows the finite state machine representation of the ALB.

Adding together the number of tests for the SELECT/DESELECT lines and the number of tests for the ALB's, we get the following result:

Theorem 4.1 *$4f^2 + 3f$ tests are sufficient to test the control part of a switch, in the fixed priority case, and, $f^22^f + 3f^2 + 3f$ are sufficient in the round-robin priority case.*

4.5.2 Network testing

Optimizing the number of tests for the entire network is more complex in the case of testing control than in the case of testing data paths because, when contention occurs at one level, it becomes difficult to use the same tests at the next level. However we were able to develop a scheme that makes it possible to test all levels at the same time. The approach applies to the fixed priority case. The method is as follows:

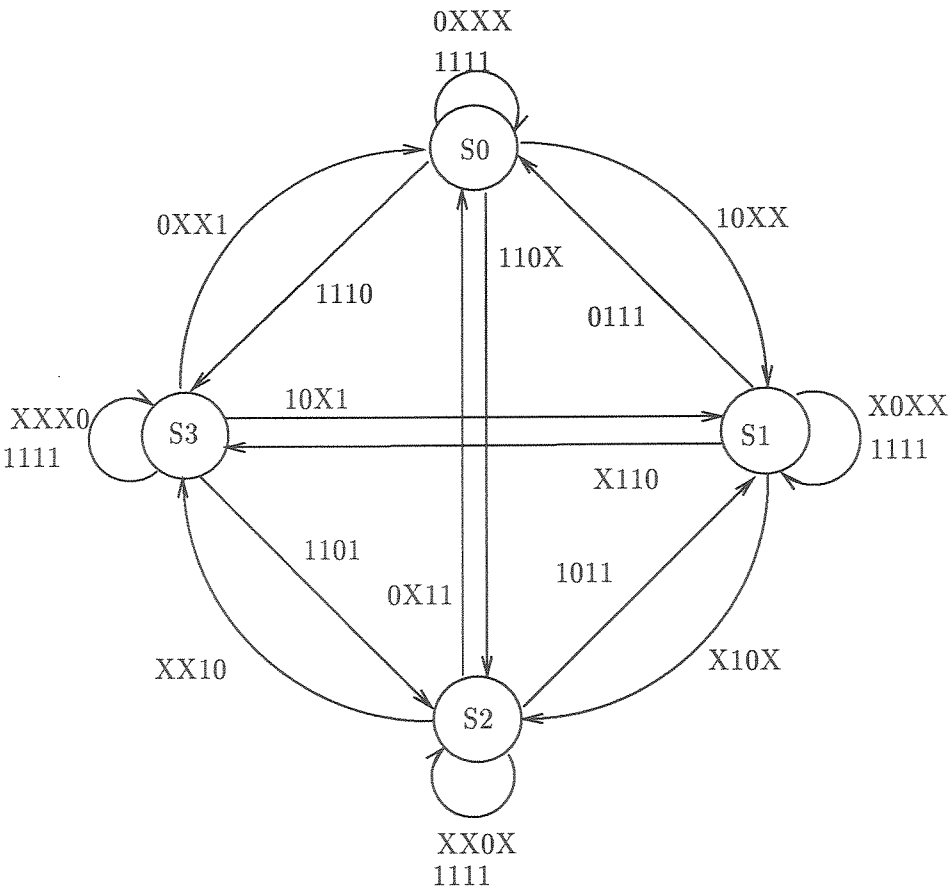

Figure 6: Finite state machine of the ALB for a 4×4 switch.

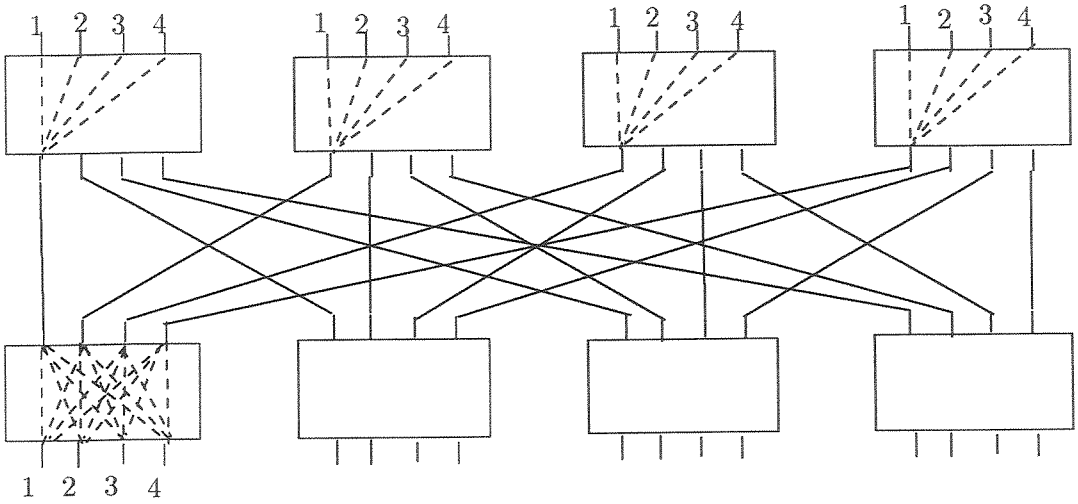

Figure 7: Routing of a test involving contention in the fixed priority case.

1. If a test does not involve contention, then it will activate as many outputs as inputs and these outputs in combination with the outputs of the other switches at the same level will be used to apply the same test to the next level,

2. If the test involves contention, then the packets that will go through in the first cycle are known since fixed priority is used, and the addresses of these packets can be chosen so that the same test will be applied to some of the switches at the next level, in the next cycle, another set of outputs will go through and can be used to apply the same test at some switches at the next level. Ultimately all testing packets will go through and will be used to apply the test to all the switches of the next level.

Figure 7 shows an example of the above scheme. The implementation of this method requires also adjusting the length of the testing packet to ensure that the desired pattern will occur at the input of the switch.

This approach can also be used in the round-robin priority case if the ALB's are in the same state at the beginning of the test. The ALB's can be in the state after powering up the system and the test can be applied at that time. This approach should be used if the number of stages in the network is high. Otherwise testing the switches, stage by stage, is better since it multiplies the number of tests only by a factor of L and it is much safer.

4.6 Estimation of testing time

Here, we consider the network used in the Sigma I computer, which has two stages and uses, for fault-tolerance purposes, 10×10 switches configured as $8 \times 8's$. The network is duplicated and one side is used to write into memory and the other to read from it. We make an estimate of the testing time using first the exhaustive method and then our approach. We assume the time taken for traversing the network and accessing the memory to be $t = 120ns$, and the average contention factor F_c is taken to be 2 in both methods. The number of different possible settings of a switch is $s = ({}^{10}C_8)^2 = 2025$ and f is equal to 8.

The testing time in the exhaustive approach would be:

$$T = 2sLtF_c((1+f)^f - 1) = 23.2hours$$

The testing time in our approach is:

$$T = 2sLtF_c(f^2 2^f + 2f^2 + 7f + 2) = 32s.$$

In the above number, we include the number of tests for data paths, routing and control. This represents over three orders of magnitude improvement! The testing time for the network is about 2610 times shorter!

In this section, we demonstrated how to apply the methods of topological testing to test the multistage interconnection networks. The three domains of application of topological testing have been illustrated: *behavior* testing has been used in the Arbitration Logic Blocks, the *organization* of the network has been extensively used to minimize the number of tests, also *hierarchy* and partitioning have been exploited in the parallel testing of the network.

The results obtained for testing control and priority are especially encouraging since they lead to reducing the test time of complex networks by several orders of magnitude. In the following section, the method will be applied for testing the communication capabilities of the hypercube computer.

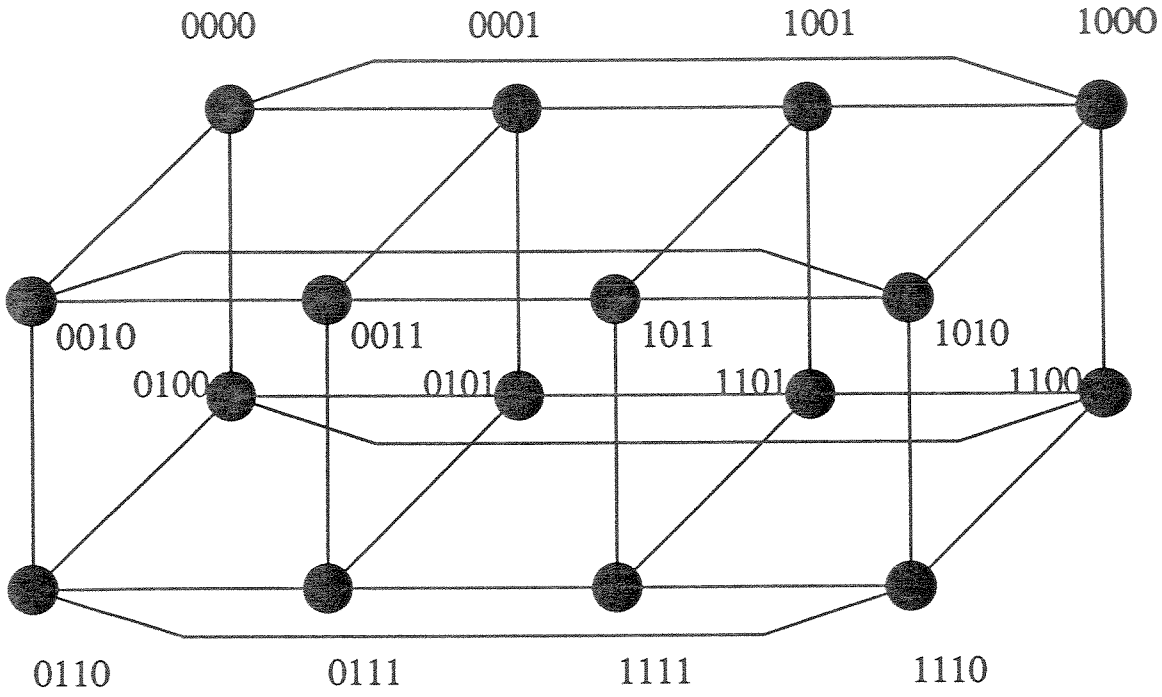

Figure 8: A hypercube of dimension four.

5 An example of hierarchical testing: testing a hypercube

5.1 The hypercube topology

A hypercube is a multiprocessor with $N = 2^n$ nodes where n is called the dimension of the hypercube. It is also referred to as n-cube or Q_n. These N processors are placed at the vertices of an n-dimensional cube. Each processor is addressed by a unique binary integer from 0 to $2^n - 1$. Each address has n bits and there is a link between two processors if their binary address differs by exactly 1 bit. Hence the degree of each vertex is n, the dimension of the cube.

The hypercube is a homogeneous machine meaning that all processors are alike. It has distributed memory and communication takes place by message-passing. Taking two cubes of a given dimension and connecting the nodes with the same addresses yields a cube of the next higher dimension. A very important and useful property of a hypercube is that it can be easily partitioned into smaller sub-cubes. A hypercube of dimension four (4-cube) is shown in Figure 8. Further details on the hypercube graphs are presented in [14].

5.2 Communication in a hypercube

There is no shared memory in a hypercube and all processors communicate by passing messages. If a message has to be sent to a processor that is not adjacent to the sender, the message is routed through intermediate processors until it reaches it's ultimate destination. A store-and-forward function is implemented at each node. The receiving processor checks the address of the message and re-routes the message if not intended for it.

5.3 Routing on a hypercube

There exists a very simple and efficient algorithm for routing on a hypercube. At each stage, the address of the destination is scanned from left and compared with the address of the current node. When we encounter a bit that is different in the addresses of the destination node and the current node, the message is sent to Alternatively,
$source = s_n s_{n-1} ... s_2 s_1$
$destination = d_n d_{n-1} ... d_2 d_1$
$Bit - wise\ EXOR = x_n x_{n-1} ... x_2 x_1$
where $x_i = s_i \oplus d_i\ for\ i = 1, ..., n$

The values of i for which $x_i = 1$ indicate the dimension that must be traversed to transfer a message from source to destination. There are d disjoint paths between two processors that are separated by a distance d.

5.4 Test Technique

Topological testing may be, in this case, applied to test the topology of the network. We would like to test all processors, all links and possibly all routes from each processor to all other processors. The latter would imply testing switching of messages from one link to another at a node. A simplistic approach would be to test a Hamiltonian linking all nodes and an Eulerian. Since all the links in the hypercube are bidirectional, there always exists an Eulerian in the graph. But this would test only the processors and the links whereas we would like to test the routing too. Besides the topology of the hypercube presents us with an excellent opportunity to exploit parallelism to reduce testing time and increase coverage. Hence an alternate technique taking advantage of ease of partitioning and parallelism in hypercube is presented next.

First partition the hypercube into node and edge disjoint rings of size four i.e. Q_2's or C_4's. A Q_2 or a C_4 is a cycle with four nodes and four edges.

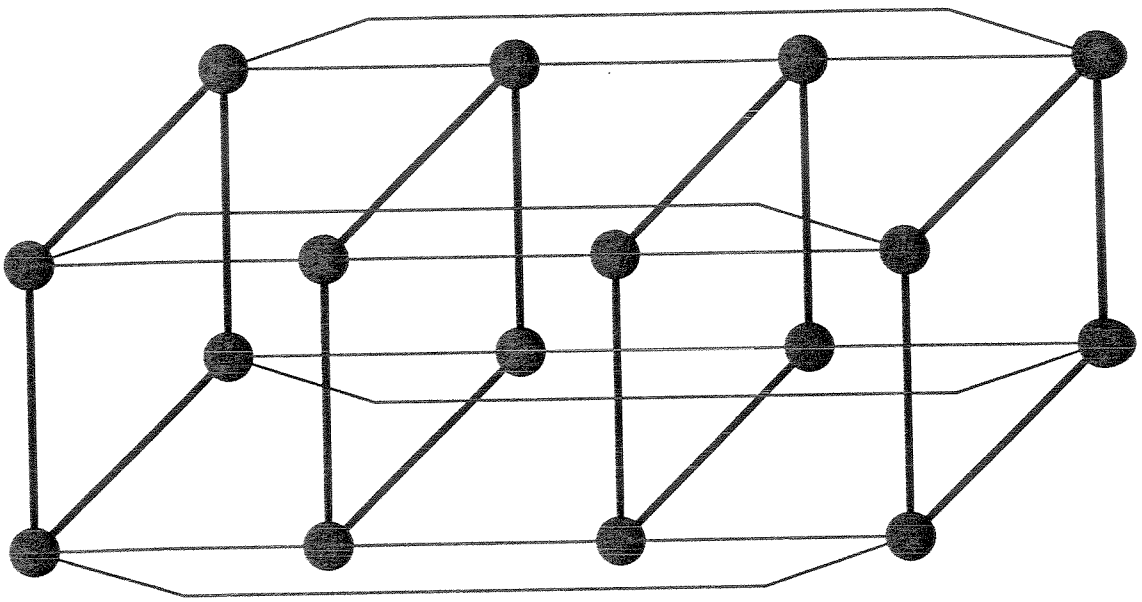

Figure 9: A partition of the hypercube.

There are 2^{n-2} such cycles in the partition such that they have no nodes and edges in common. One such partition is shown in Figure 9 and another partition in Figure 10. Now for each cycle of four processors, perform a test which is a slight variation of 'Ring Test'. In the ring test, the processors are configured in the form of a ring and all processors simultaneously pass messages to their neighbors. Instead, in our case, each processor passes a message to the processor that lies at the diagonal end through its neighbor. All messages are passed in one direction, either clockwise or anti-clockwise. An example is shown in Figure 11. In the figure, processor routes the message from D to B, processor B routes message from A to C, processor C routes message from B to D and processor D routes message from processor C to A. After the messages have been received, the direction of transmission is reversed. The message that is passed may also be used to test the parity of the links. This test is carried out in parallel on all disjoint rings in the partition.

After the particular configuration is tested, another partition is formed and the test is repeated. This is done till all possible partitions are exhausted. There are ${}^nC_2$ possible partitions and hence all these ${}^nC_2$ are tested in sequence one after another. Hence the total time to test the hypercube is of the order of $O(n^2)$ where $n = \log N$. Hence the time to test

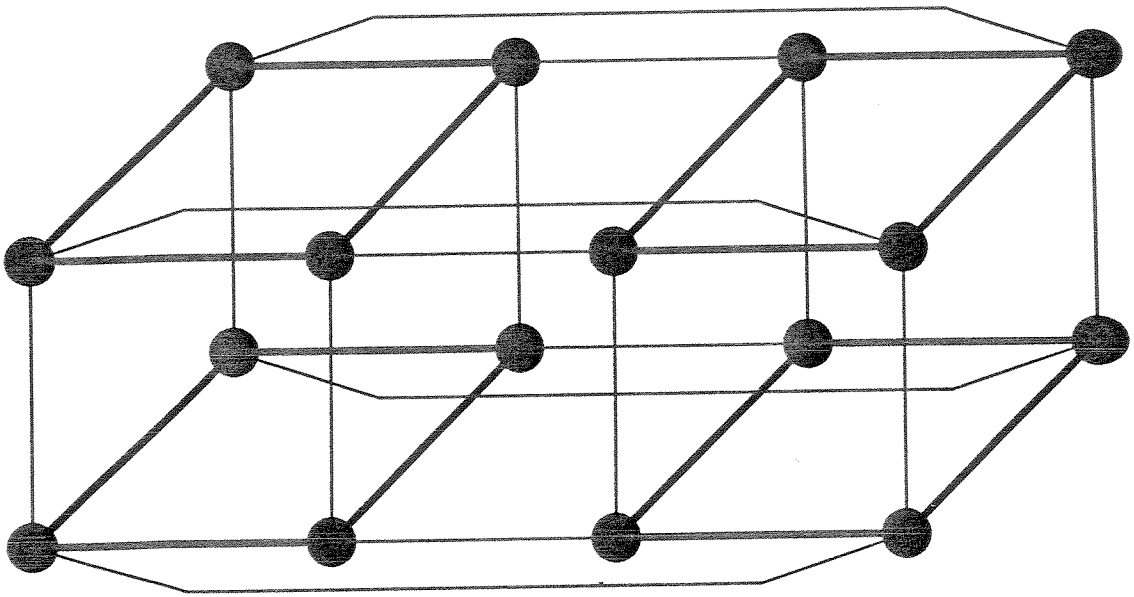

Figure 10: Another partition of the hypercube.

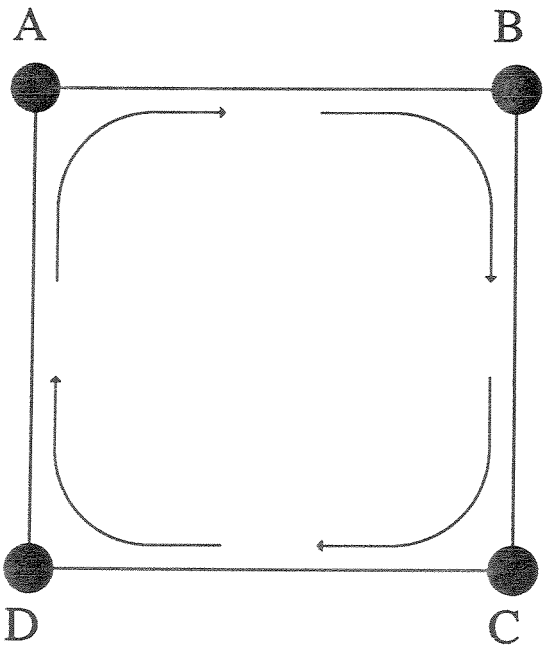

Figure 11: Routing messages on a ring.

the hypercube increases sublinearly with n and so it seems quite acceptable. Intuitively this seems to be the minimal test time though we do not have a proof for that.

5.5 Test Coverage

The above test procedure confirms whether all processors are working. To obtain more thorough testing of processors, the routing processor might be required to perform a set of operations on the data that it is routing. All links are also tested. The test also determines whether a processor can route a message from edge i to edge j for all incident edges i and j. Since during routing, a processor just reads in a message from one link and sends it to other, hence we can see that all paths between any pair of processors on the hypercube are tested.

5.6 Testing for Contention

Contention may occur if two messages arrive simultaneously at a processor. We would like test that the processor can handle such cases properly and according to specifications. Hence during testing, more than one messages are sent simultaneously to the processor under test. Since a processor is linked to n neighbors in an n-dimensional cube, for complete test, we successively test for contention of two to n messages arriving simultaneously. Note that this test may be extremely difficult, if not impossible, to execute on the current generation of systems that are asynchronous. For the case where no advantage of partitioning and parallelism is taken and all processors are tested serially, all 2^n processors are tested successively for contention of i messages where $2 \leq i \leq n$. Hence the time required to test all processors is $2^n\ t\ \sum_{i=2}^{n}\ {}^nC_i\ i = n(2^{n-1}-1)t$ where t is the time required for one message.

However, if we partition the cube into subcubes of dimension i where $2 \leq i \leq n$, within each partition, we can test at least two processors simultaneously for contention of i messages and all the 2^{n-i} subcubes can be tested in parallel. There are ${}^nC_i$ different ways of partitioning a hypercube into subcubes of dimension i. Hence the upper bound on the time required to test for contention is $[8\ {}^nC_2 + \sum_{i=3}^{n}\ {}^nC_i 2^{i-1} i]t = (8\ {}^nC_2 + n(3^{n-1} - 2n + 1))t$. The second term appears because in partitions of subcubes of dimension two, only one processor may be tested at a time.

Based on the data in [15], we observe that the time to test contention in the hypercube without topological test techniques would be in the range

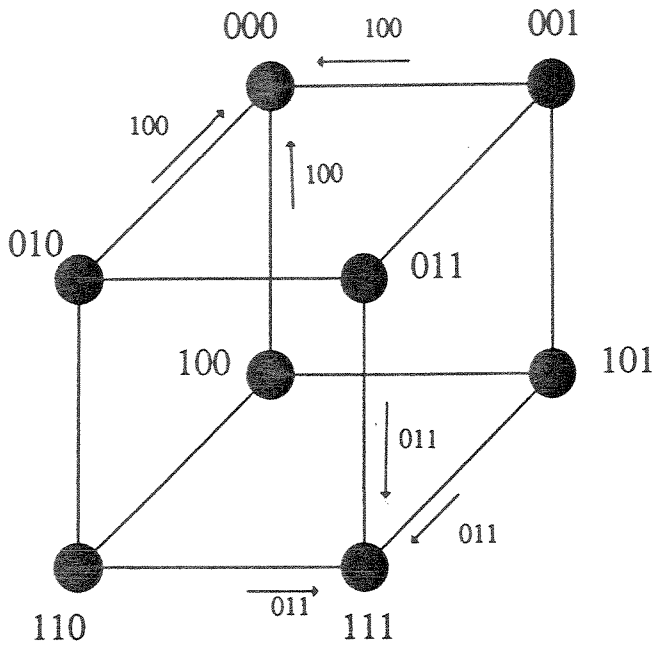

Figure 12: Testing for contention on the three dimensional hypercube.

from 392 seconds (approximately 7 minutes) to 31186 seconds (approximately 8 hours 40 minutes) for a hypercube of dimension ten. The test time reduces to the range from 15 seconds to 1192 seconds (approximately 20 minutes) when topological test technique is applied. topological test techniques are applied.

6 Conclusions

A new, powerful testing method, called topological testing was presented. Topological testing can be applied to the entire spectrum of a system hierarchy from a circuit to a multiprocessor level and to all forms of graph system descriptions that represent behavior, organization or hierarchy of the system. Once appropriate model is developed, basic graph theory concepts such as Hamiltonian, Eulerian, Traveling Salesman Problem, Chinese Postman Problem, partitioning and path covering can be used. The potential of the topological testing techniques has been demonstrated on practical examples where orders of magnitude testing time reductions have been achieved. The universality and power of the proposed approach may have a lasting effect on forthcoming generations of testing algorithms and system integra-

tion.

References

[1] Aho, A. V., A. T. Dahbura, D. Lee and M. U. Uyar, "An optimization technique for protocol conformance test generation based on UIO sequences and Rural Chinese Postman tours," *Proceedings of IFIP WG 6.1 8^{th} International Symposium on Protocol Specification, Testing, and Verification,* Atlantic City, June 1988.

[2] Uyar, M. U. and A. T. Dahbura, "Optimal test sequence generation for protocols: the Chinese Postman algorithm applied to Q.931," *Proc. of Globecom, 86.*

[3] Opper, E., *Fault diagnosis of banyan networks,* Ph.D. dissertation, Department of Electrical and Computer Engineering, the University of Texas at Austin, 1984.

[4] Thatte, S. M. and J. A. Abraham, "Test generation for multiprocessors," *IEEE Transactions on Computers,* Jun. 1980.

[5] Jone, W.-B., M. Pereira and C. A. Papachristou, "A new test scheduling method and its hardware support," *VLSI Technical Bulletin,* Dec. 1988, pp. 85-103.

[6] Hayes, J. P., "Testing memories for single-cell pattern-sensitive faults," *IEEE Transactions on Computers,* Mar. 1980, pp. 249-254.

[7] Lawler, E. L., J. K. Lenstra, and A. H. G. Rimnooy Kan, eds., *The Traveling Salesman Problem,* North-Holland, 1985.

[8] Nejmeh, B. A., "NPATH: A measure of execution path complexity and its applications," *Communications of the ACM,* Feb. 1988, pp. 188-200.

[9] Hiraki, K., Private communication, Tokyo, July 1988.

[10] Goke L. R., and G. J. Lipovski, "Banyan networks for partitioning multiprocessor systems," *Proc. of the First Annual Computer Architecture Conference,* 1973, pp. 21-28.

[11] Malek, M. and E. Opper, "Multiple fault diagnosis of SW-banyan networks," *Proc. of the 13^{th} Annual Int. Symp. on Fault Tolerant Computing,* Jun. 1983, pp. 446-449.

[12] Opper, E. and M. Malek, "Real-time diagnosis of banyan networks," *Proc. of the Real-Time Systems Symp.,* Los Angeles, Dec. 1982, pp. 27-36.

[13] Hung, A. C. and M. Malek, "A 4×4 modular crossbar design for the multistage interconnection networks," *Proc. of the Real-Time Systems Symp.,* Dec. 1981, pp. 3-12.

[14] Harary F., J. P. Hayes and H.-J. Wu, "A survey of the theory of hypercube graphs," *Comp. Math. Applic.* Vol. 15, No. 4, pp 277-289, 1988.

[15] Kolawa, A. and S. W. Otto "Performance of the Mark II and Intel Hypercubes," *Proceedings of the First Conference on Hypercube Multiprocessors,* August 1985, pp. 272-275.

[16] Heath, M. T. (ed.), *Hypercube Multiprocessors 1987, Proceedings of the First Conference on Hypercube Multiprocessors,* Sept. 1986.