TEST SCHEDULING AND CONFIGURATION IN A SELF-REPAIRING COMPUTER

by

WALTER LEE GREGORY, JR.

Thesis submitted to the Graduate Faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

Electrical Engineering

Approved:

R A Thompson

Dr. R. A. Thompson, Chairman

F F Gray

Dr. F. G. Gray

C W Bostian

Dr. C. W. Bostian

May, 1977

Blacksburg, Virginia

LD
5655
V855
1977
G74
c.2

ACKNOWLEDGMENTS

The author wishes to express his sincere thanks and gratitude to Dr. R. A. Thompson who was generous with both his time and his thoughts in assisting me with this thesis, and to Drs. F. G. Gray and C. W. Bostian for providing guidance and suggesting corrections for the original manuscript.

The author would further like to thank Steve Walters for his advice and counsel concerning Tessellation Automata and for the use of his tessellation simulator computer program. Also, thanks go to Don Stewart for his photographic talent, and to my typist Mary Brent whose fast and excellent work was responsible for me being able to complete this thesis on schedule.

This research was partially supported by the National Science Foundation under grant number DCR75-06543.

TABLE OF CONTENTS

	Page
ACKNOWLEDGEMENTS	ii
LIST OF FIGURES	vii
1.0 INTRODUCTION	1
1.1 Purpose	1
1.2 Philosophy of a Self-Repairing Computer	1
1.3 Applications	1
2.0 A SELF REPAIRING COMPUTER	3
2.1 Computational Structure	3
2.2 Cellular Arrays	3
2.3 Tessellation Automata	6
2.4 Tessellation Dimension	8
2.5 Computational System Within the TA	9
2.6 Reconfiguration	9
2.7 Faulty Cell Removal	11
2.7.1 Quarantined Cells	11
2.7.2 Phagocytosis	12
2.8 Seed Fault Considerations	15
2.9 Pragmatics	16
3.0 CELL COMMUNICATION	17
3.1 Ports	17
3.2 Port Testability	17
3.2.1 Single Port Testability	19
3.2.2 Independent Port Testability	19

Page

3.2.3 C-Operative Port Testability 21
3.3 Port Technology Types 21
3.3.1 Electronically Isolated Ports. 22
3.3.2 Optically Isolated Ports 22
3.3.3 Electromagnetically Isolated Ports 25
3.4 Linkage Faults. 27
3.4.1 Electronically Isolated Port Faults. . . . 27
3.4.2 Optically Isolated Port Faults 28
3.4.3 Electromagnetically Isolated Port Faults . 28
4.0 TEST CONFIGURATION. 30
4.1 Cell Testing. 30
4.2 Cell Faults 30
4.3 Single Diagnoser Testing. 32
4.3.1 Single Port Testing. 32
4.3.2 Independent Port Testing 33
4.4 Multiple Daignoser Testing. 36
4.4.1 Cooperative Port Testing 37
4.4.2 Cooperative Port Testing with Indirect Links. 38
4.5 Comparison Testing. 41
4.6 Reconfiguration Inhibitor 43
4.7 Testing Hyperplane. 44
5.0 SEQUENTIAL TEST SCHEDULING. 45
5.1 Sequential Test Scheduling. 45

Page

5.1.1 Test Propagation 45
5.1.2 Array Geometry 46
5.1.3 Propagation Mechanism. 47
5.1.4 Reconfiguration. 48
5.1.5 Configuration Propagation. 50
5.2 Test Propagation in Infinite Arrays 51
5.2.1 Concentric Test Propagation in E^2. 51
5.2.2 Spiral Test Propagation in E^2. 55
5.2.3 Modified Spiral Test Propagation in E^2 . . 60
5.2.4 Sequential Test Generators 61
5.3 Test Propagation in Modified Geometry Arrays. . . 66
5.4 Propagation Perturbation. 71
5.5 Faults in Test Propagation. 75
6.0 PARALLEL TEST SCHEDULING. 76
6.1 Parallel Testing. 76
6.2 Timing Considerations 76
6.3 System Preservation 77
6.4 Testing Hyperplane. 78
6.5 Semistructure 79
6.6 The Parallel Testing Procedure. 80
6.7 Advantages of the Parallel Testing Procedure. . . 88
6.8 Quarantined Cells 89
6.9 Modification of the Parallel Testing Procedure. . 90

Page

7.0 FAULT RESISTANT TECHNIQUES 92

7.1 Critical Faults. 92

7.2 Combined Testing Schemes 93

7.3 Redundant Architecture 93

7.4 Redundant Software 97

8.0 CONCLUSIONS. 99

8.1 Conclusions. 99

8.2 Recommendations. 99

8.3 Future Research. 100

BIBLIOGRAPHY. 102

VITA. 103

LIST OF FIGURES

FIGURE		PAGE
2.2-1	Parallel hyperplanes within the cellular array.	5
2.7-1	The neighborhood of cell j in E^2.	13
2.7-2	Phagocytosis of cell j in E^2.	14
3.1-1	Cell interconnectivity via ports.	18
3.2-1	Cell showing ORing of inputs from ports to form the input to the contained machine M.	20
3.3-1	Cell with electronically (input) isolated ports.	23
3.3-2	Cell with optically isolated ports.	24
3.3-3	Cell with electromagnetically isolated ports.	26
4.3-1	Single diagnoser testing showing staggered sequence diagnosis of single port testable neighboring cells.	34
4.3-2	Independent port testing carried out by a single diagnoser cell.	35
4.4-1	Multiple diagnoser testing configuration used in cooperative port testing with direct links.	39
4.4-2	Multiple diagnoser testing configuration using indirect links for cooperative port testing.	40
4.5-1	Comparison testing using three recently tested cells as output sequence generators.	42
5.2-1	Generations of the concentric test propagation in E^2.	54
5.2-2	Spiral sequential test propagation.	57
5.2-3	Modified spiral sequential test propagation.	63
5.3-1(a)	Conceptualized representation of a 4 x 4 skew torus.	67
5.3-1(b)	Cylindrical conceptualization of a 4 x 4 skew torus.	68
5.3-2	Boundary connections of a 4 x 4 skew torus.	69
5.3-3	Conformal mapping of a 4 x 4 skew torus onto a two dimensional infinite array.	70

LIST OF FIGURES (Cont'd)

FIGURE		PAGE
5.4-1	Illustration of a possible situation where the diagnoser of a spiral sequential test must propagate around a group of quarantined cells.	72
5.4-2	A situation in which a modified spiral propagating test must bounce off of a blockage of quarantined cells.	74
6.5-1	Structure of a single cell showing the semicell link between each semicell of the red semiarray and the black semiarray.	81
6.5-2	Subcell links between the three subcells within each semicell of a given semiarray.	82
7.3-1	Conceptualization of redundant computational subsystem and sequential test generators on a self-repairing array.	96

1.0 INTRODUCTION

1.1 Purpose

The purpose of this research is to suggest various testing architectures that can be implemented in an array of identical cells. The internal structure of such cells is intentionally ignored in order that the fundamental testing concepts be general enough so that they can be applied to a variety of physically realized cell types. The eventual utility of such a self-testing array is for the implementation of a self-repairing computer.

1.2 Philosophy of a Self-Repairing Computer

Is there really such a thing as a self-repairing computer? No. There is no finite physically-realizable computer structure that is perpetually self-repairing. For example, in any finite cellular array there exists a fixed probability that each cell in the finite array will simultaneously fail provided that the probability of individual cell failure is greater than zero. A catastrophic failure of this nature would surely neutralize any self-healing properties of such a system. When self-repairing computers are discussed, what is really meant is a computer system that can maintain itself in terms of the functions it performs within predictable physical limitations and specified failure probabilities.

1.3 Applications

There is application for such a self-repairing computer system whenever highly reliable computations must be performed. An example of

this application is in the control of spacecraft in deep-space unmanned missions. In this environment, no external maintenance could be performed on the computer and accurate operation might be required over a very long period of time. It is desirable to design a self-repairing computer that can operate within these constraints with an arbitrarily specified reliability during the time limits of the mission.

2.0 A SELF REPAIRING COMPUTER

2.1 Computational Structure

With the goal in mind of having a computing structure capable of performing information processing tasks while maintaining a fault tolerant environment, it is desired that a "fabric"[1] or generalized structure be developed in which a specific computing system can be implemented. This structure must further have the property by which faults and other permanent perturbations in the system be rendered invisible to the remaining operational structure. Furthermore, the functions once performed by the faulty system element are adopted by a spare, previously undedicated (quiescent), element of the structure. In this way, it can be said that this computational system is self-repairing provided sufficient spare elements are available. Following this line of thought, each such element of the structure must be able to perform any of the tasks delegated to any other element in the system. Thus, each element must be an identical unit or cell. The structure of identical cells arranged in multi-dimensional cellular arrays with uniform interconnectivity has been investigated by VonNeumann(1966)[2], Hennie (1964)[3], Holland (1970)[4], Unger (1962)[5], Yamada and Amoroso (1969-1971)[6,7,8], and by others. This structure lends itself as a suitable fabric in which to implement just such a computational system.

2.2 Cellular Arrays

Within this cellular array, each cell will consist of a finite state machine capable of performing any of a specific set of functions;

that is, each cell will be capable of emulating the behavior of one of a set of sub-machines, each designated as a particular function. Let $F = \{M_1, M_2, \ldots, M_i, \ldots, M_n\}$ be the set of functions which each cell can execute. With the formulation of an interconnection rule I which indicates how the functions intercommunicate, it is possible to define the desired computational system as $S = (F, I)$[1]. Therefore, S can be implemented in a cellular array, provided that each cell is capable of realizing any $M_i \in F$ and can determine which M_i to realize based on the realized functions of its neighboring cells. This function assignment capability (control) must be consistent with I, the system interconnection specification, and with the array interconnection pattern.

A conceptual separation of cell operation can be made[1]. Consider each cell as a two-part device, each part residing in one of two parallel hyperplanes. The two cell parts intercommunicate and each part communicates with the neighboring cells to which it is directly connected in their corresponding hyperplanes as is suggested in Fig. 2.2-1. The lower, or "control", hyperplane is responsible for determining the cell functional assignment of the system to be implemented. Thus, the control portion of the cell examines the functional assignments of its lower neighbors and determines an M_i, a specific system function, which its upper part is to be assigned. The upper, or "computing" hyperplane actually performs this assigned system operation. Data used in these computational functions is communicated between neighboring cells of this upper hyperplane. The properties of the control hyperplane operation can best be studied when it is mathematically modeled as a Tessellation Automaton.

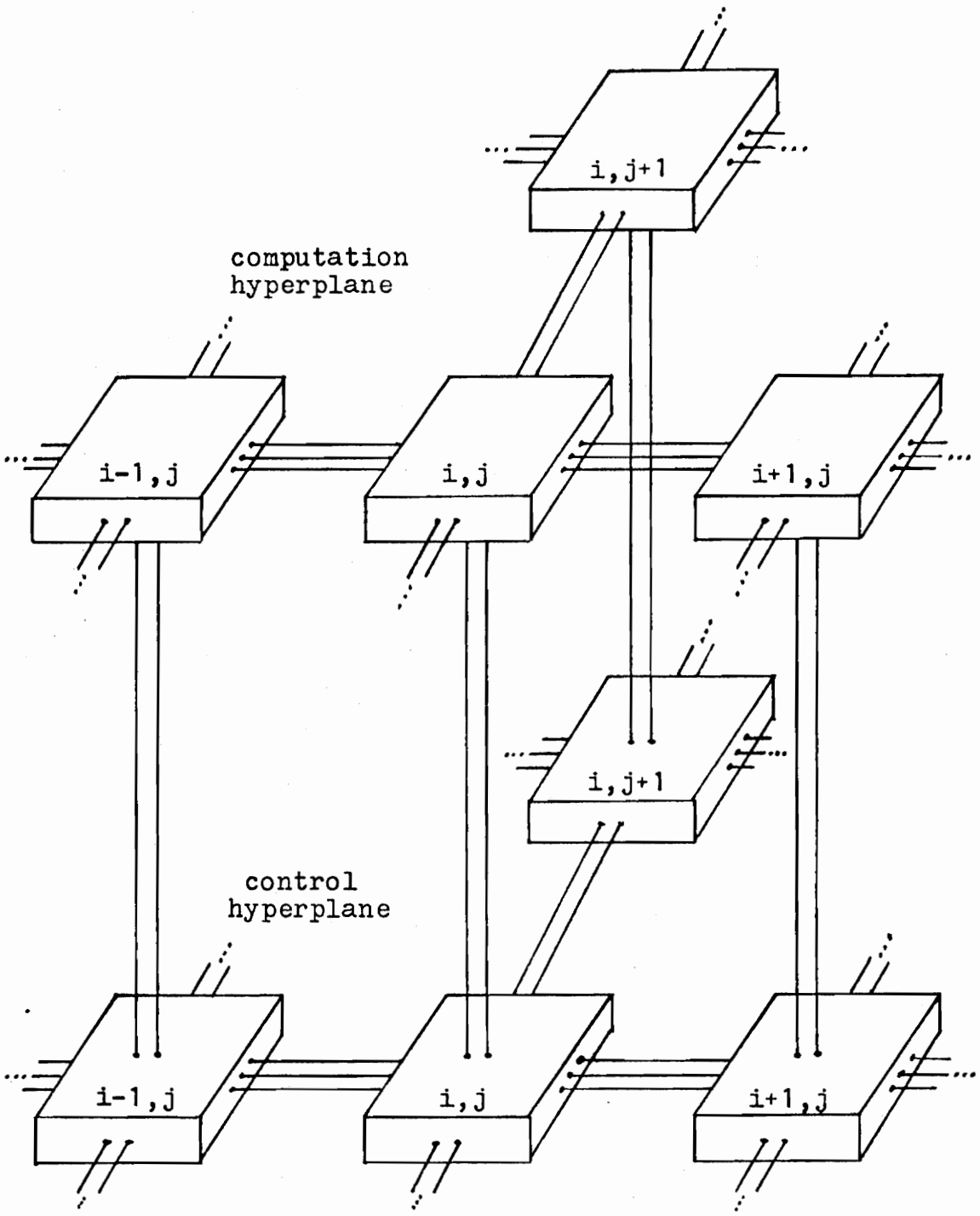


Figure 2.2-1. Parallel hyperplanes within the cellular array.

2.3 Tessellation Automata

A Tessellation Automaton (TA) as defined by Yamada and Amoroso (1969-1971)[6,7,8], consists of a finite state set A (the state alphabet), the tessellation array which is the d-dimensional integer space E^d (the set of all integer d-tuples), and the neighborhood index X (an n-tuple of distinct d-tuples). The neighborhood of cell j corresponds to the finite set of cells in the array from which the cell at point j(j is a d-tuple) has direct connections. Thus, a cell of the TA exists at each point which is an element of E^d. Let ξ_i be a d-tuple. If $X = (\xi_1, \xi_2, \ldots, \xi_n)$ is the neighborhood index, then $N(X,j) = [(j+\xi_1), (j+\xi_2), \ldots, (j+\xi_n)]$ identifies the specific neighbors of the j^{th} (j is a d-tuple) cell. Let ξ_o be a d-tuple of zeroes. Then, if $\xi_o \varepsilon X$, the j^{th} cell is in its own neighborhood.

A mapping $c : E^d \rightarrow A$ is a "configuration" and indicates the cells of its argument within the array. The notation c(j) denotes the local configuration or state of the j^{th} cell, c(N(X,j)) is the neighborhood configuration of the j^{th} cell, and $c(E^d)$ is the current global configuration of the entire TA. The set of all global configurations $c(E^d)$ that the TA can achieve is denoted by C. A mapping $\sigma : A^n \rightarrow A$ is a "local transformation" which determines the next state of the j^{th} cell as a function of its neighborhood configuration, and is written $c'(j) = \sigma[c(N(X,j))]$. Yamada and Amoroso[6,7,8] allowed for a multitude of local transformations; the selection of the one used at a particular time is analogous to a TA input at that time. The theory developed around the existence of sequences of local transformations

that could take the TA from an initial configuration to a final configuration in nondeterministic fashion. A TA is called "autonomous"[1] if there is a unique local transformation σ. Corresponding to the local transformation σ is a "global transformation"[1] τ: $C \rightarrow C$ defined by the simultaneous application of σ to every local cell in the tessellation. Thus, $c'(j) = \sigma[c(N(X,j))]$ for all cells j causes $c'(E^d) = \tau[c(E^d)]$.

A cell is "quiescent"[1] if it is in the unique state 0 ($0 \in A$). Initially, all cells are quiescent. The TA can be configured to an initial configuration c_i by externally forcing each cell comprising c_i to its designated function. It is also possible to specify σ such that a single cell, or "seed"[1], may be externally forced out of quiescence so that the TA behavior can be observed as the resulting sequence of global configurations grows towards a desired configuration as τ is applied in synchronous sequential fashion. If $c'(j) = c(j)$ at some time, then this j^{th} cell is said to be in "local equilibrium"[1] at that time. If this is true for all cells j, then $c'(E^d) = c(E^d)$ and the TA is said to be in "global equilibrium"[1]. It must be assumed that $\sigma[0, \ldots, 0] = 0$ to ensure the local equilibrium of quiescent neighborhoods provided that $\xi_o \in X$.

A TA is called "nonerasing"[1] if a nonquiescent cell can never become quiescent except by some form of external reset. Formally, this is stated as $c(j) \neq 0 \Rightarrow c'(j) \neq 0$. Thus, in a nonerasing TA, the sequential behavior of global configurations is characterized by a nondecreasing inquiescence into all tessellation dimensions from

the single seed cell or larger imposed initial configuration c_i.

A configuration c_f is called a "final configuration"[1] if: (1) $c_f \varepsilon C$ when the TA is initialized to a specific initial configuration c_i and (2) c_f is in global equilibrium. Let a TA be initialized to c_i and let τ be applied sequentially to all successive configurations. If the TA never assumes a final configuration, it is said to be "unstable"[1]. Or, if the TA eventually assumes a final configuration c_f it will stay in equilibrium and never change its configuration. Thus, a final configuration c_f is defined by:
(1) $c_f \varepsilon C$ and (2) $\tau [c_f] = c_f$. An Initialized Autonomous Tessellation Automaton (IATA)[1] is formally defined as a five-tuple $\langle A, E^d, X, \sigma, c_i \rangle$. Furthermore, C is formally defined as:

$$C = \{c \mid c = c_i \text{ or } c = c'(E^d) = \tau [\tilde{c}] \text{ for } \tilde{c} \varepsilon C\} .$$

Unless otherwise stated, it is assumed that the TA contains no laminations[6]. That is, any cell j may be considered able to communicate with any other cell i by way of an arbitrary path of connected neighbors compatible with the neighborhood index specified by X.

2.4 Tessellation Dimension

Although the results of this paper are not restricted to a specific tessellation dimensionality d, unless specifically so stated, a dimensionality of d = 2 will be assumed unless otherwise noted. At this time a two-dimensional array seems suitable for most suggestable forms of computational array architecture. This seems a particularly reasonable assumption in view of the result derived by Thompson, Walters, and Gray (1977)[1] whereby a Turing machine defined in the classical

sense was emulated by a one-dimensional IATA with the particularly simple neighborhood index X = { -1, 0, +1 }. With such a powerful computing device being implemented in only one dimension, a two-dimensional TA should provide the necessary computational power needed by most desired systems.

2.5 Computational System Within the TA

By merging the notion of our computational system $S = (F,I)$ with the definitions put forth as part of the TA, it can be seen that each finite state machine of the set F will be in the form of a subset of the TA state alphabet A; that is, $M_i = \{A_1, A_2, \ldots, A_n\}$ where $A_i \in A$. For example, if a cell is to be capable of performing a test of some sort, this function M_t might be implemented within the control hyperplane TA as $M_t = \{T, D\}$ where $T \in A$ is the control state indicating that the test is being done and $D \in A$ will indicate the control state where the test has been completed. Moreover, the intercommunication rule I will be largely specified by the neighborhood index X which will designate communication paths in the control hyperplane, and thus usually in the computational hyperplane as well.

2.6 Reconfiguration

Returning to the goal of implementing our system in such a way so that faulty cells can be removed from having any effect on the computational configuration within the array, and that spare cells can be used to take over the tasks previously performed by the faulty cells, the notion of reconfiguration is introduced. Should a cell in the

control hyperplane TA detect a faulty cell, it is possible to have designated σ so that the configuration be changed in a way to avoid this faulty cell while again regaining the functional integrity of the desired configuration. Thus, if the control TA contains a final desired configuration c_f before the fault occurred, the perturbated configuration c_p resulting from the faulty cell will induce repeated applications of τ to successive configurations until another final configuration c_f' is obtained in such a way as the computational abilities of this configuration c_f' are identical to those of the previous configuration c_f with the only difference being the existence of a faulty cell which no longer takes an active functional operation as part of the working array. In this reconfiguration process, the physical layout of the computational array may have changed in such a way as to avoid the faulty cell by assigning functions to previously quiescent cells (spares) to replace the functions once performed by the faulty cell. For this process to successfully occur in the presence of an indefinite number of faulty cells, it is necessary to have an infinite number of spare cells as was suggested by the infinite array dictated by the TA definition. Thus, the degree of redundancy can be thought of as infinite given an infinite cellular array. Even if an infinite array is not physically realizable, an array much larger than that needed for the desired computational configuration in a fault free array can supply the necessary redundancy for a specified length of time provided the statistical reliability of the individual cell is known.

2.7 Faulty Cell Removal

Through reconfiguration it is possible for the computational array to simply "walk away" from a faulty cell. However, it is desirable to remove all effects of this faulty cell from the array so that it cannot: (1) spew random and erroneous data throughout the array and possibly to the outside world, (2) initiate false and hazardous local transformations within its neighborhood, nor, (3) interfere with any future reconfigurations necessitated by future faulty cells. It has even been suggested that a cell be armed with the capability to physically destroy any neighbor found faulty. Despite the fact that this notion is suggestive of what one might expect to find in a Nazi science fiction novel, it is necessary to render a faulty cell incapable of causing perturbations to be propagated throughout the array. There are two ways this can be accomplished within the TA.

2.7.1 Quarantined Cells

If the nature of the cell fault is such that it is located in the computational part of the cell when detected by the cell neighbors, it is possible that the functionally sound control part of the faulty cell map itself into a quarantined state (Q) based on control state information passed on by one or more of these neighbors.

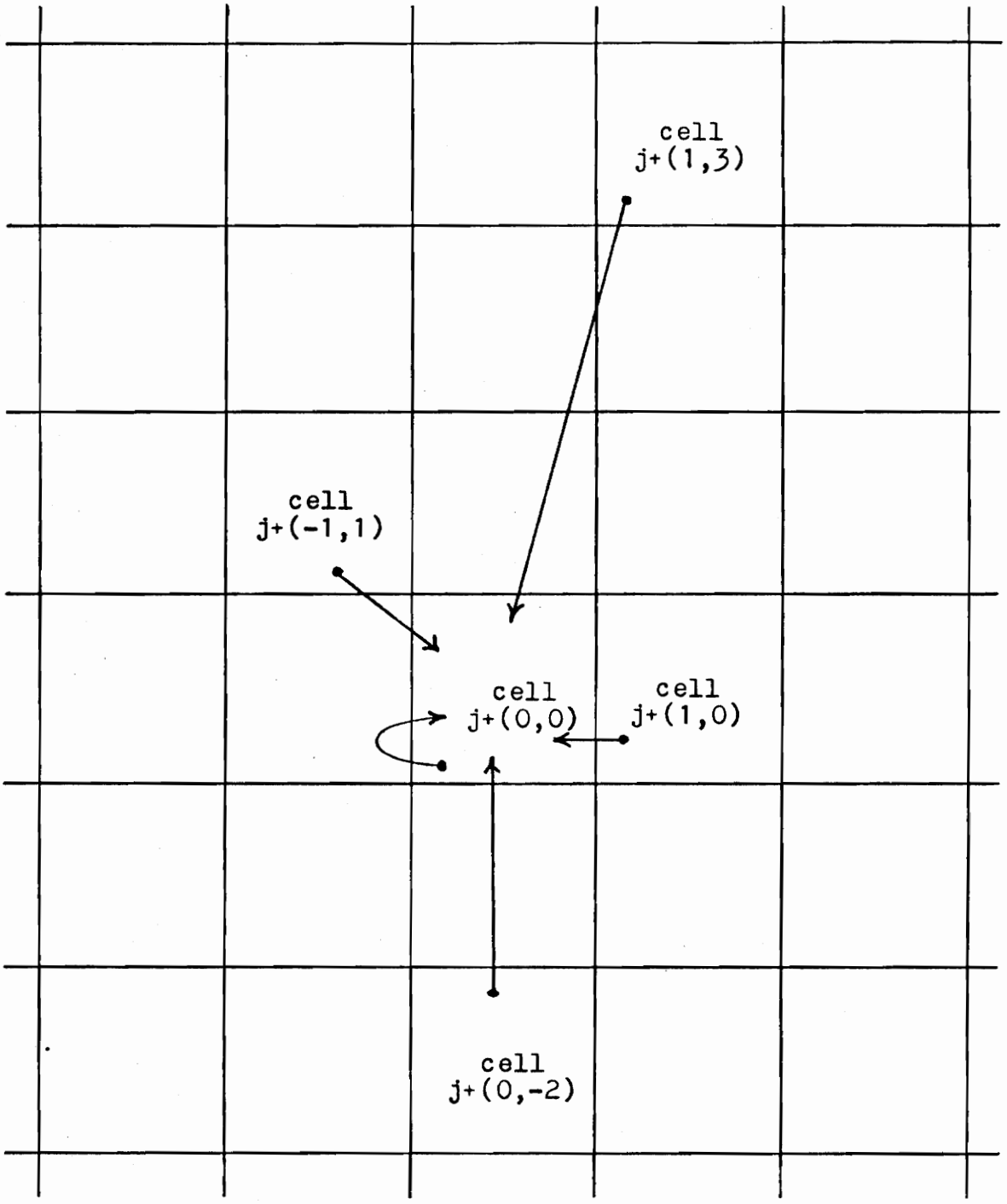
Once a cell enters the quarantined state, it must remain in permanent local equilibrium; that is, once a cell is quarantined, it will always remain in that state: $c(j) = Q \Longrightarrow c'(j) = Q$. The state outputs from the cell identifying its quarantined status to its neighbors can be used as an indication that all outputs of the computation hyperplane

segment of the quarantined cell should be ignored. If the cell to be quarantined refuses to go into this state, or if the fault was found to be in the control portion of the cell in question, it will be necessary to isolate the state outputs of the control plane so that false configurations will not be initiated by the faulty cell.

2.7.2 Phagocytosis

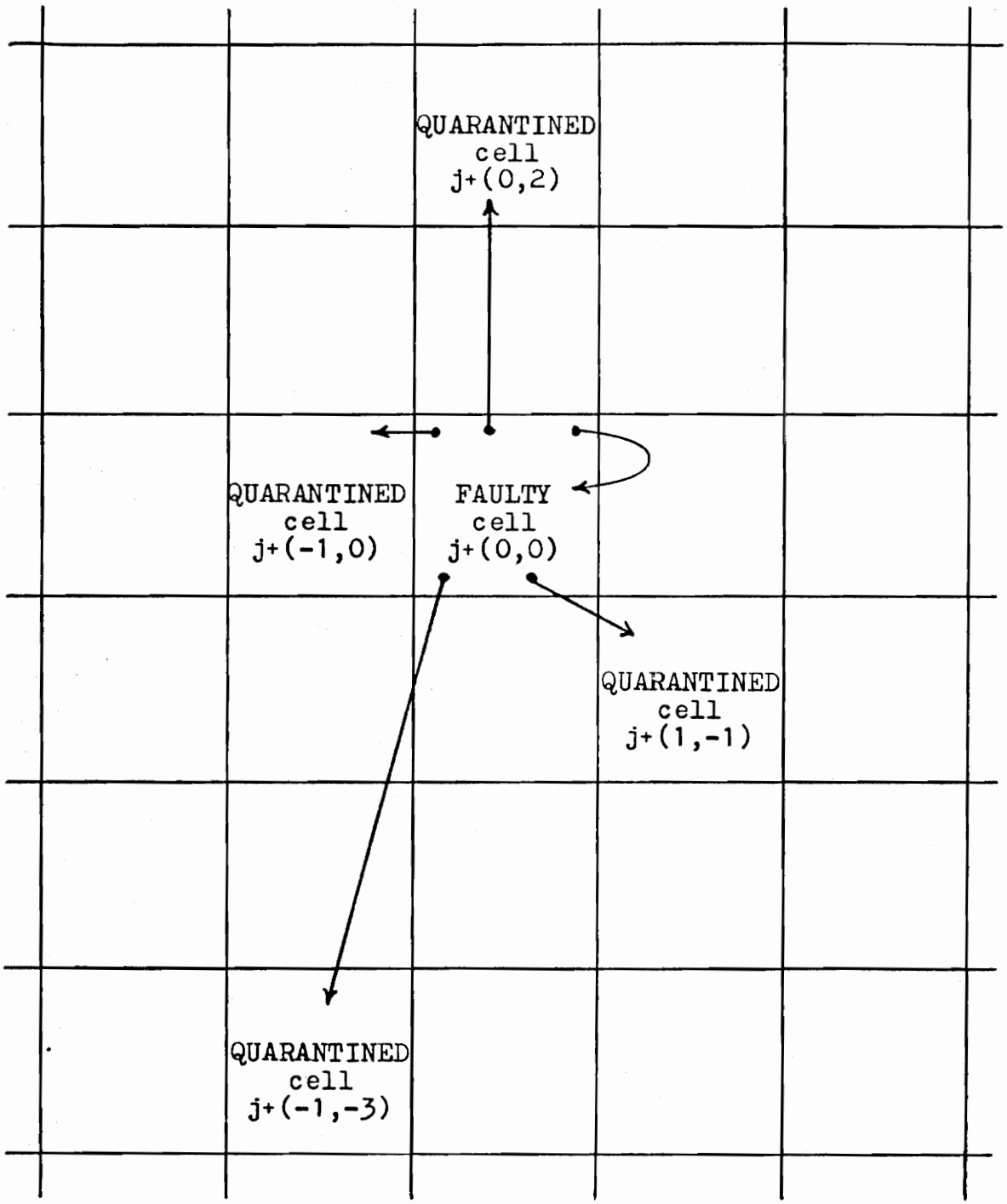
Phagocytosis is a term used in the science of immunology derived from Greek meaning the engulfing of particulate matter by phagocytes. Phagocytes are cells that characteristically engulf foreign matter and consume debris and foreign bodies (faulty cells). Thus, the term phagocytosis is used to describe the process of literally "walling-off" a faulty cell from the rest of the working array by causing each neighbor that receives any type of output connection from the faulty cell to go into the quarantined state permanently.

As an example, the j^{th} cell in E^2 (two-dimensional TA) with the neighborhood index X = { (0,0), (1,0), (0,-2), (1,3), (-1,1) } is used. Figure 2.7-1 shows cell j and its neighborhood which consists of four other cells which output state information to cell j. From this neighborhood pattern, it can be determined which cells in E^2 receive outputs from cell j; that is, all cells which have cell j in this neighborhood can be determined. It is these healthy cells that must be quarantined so that the outputs from faulty cell j are not propagated any further into the array in either hyperplane. Thus, when these cells { j + (-1,0), j + (0,2), j + (-1,-3), j + (1,-1) } are quarantined, cell j will have been phagocytized; that is, consumed by the phagocyte



$X = \{(0,0),(1,0),(0,-2),(1,3),(-1,1)\}$

Figure 2.7-1. The neighborhood of cell j in E^2.



$c(j+(0,0)) = c(j) = \text{faulty}$

$c(j+(0,2)) = c(j+(-1,0)) = c(j+(-1,-3)) = c(j+(1,-1)) = Q$

Figure 2.7-2. Phagocytosis of cell j in E^2.

quarantine cells which have cell j in their neighborhoods as is illustrated in Fig. 2.7-2.

2.8 Seed Fault Considerations

It has been previously mentioned that it is possible to specify σ such that a single cell can be forced out of quiesence and thus cause the growth of configurations in the TA towards the goal of a desired computational configuration. This is certainly a convenient method of initializing a fully quiesent array; however, in view of possible fault occurrences, this property may be wholly undesirable. Consider the fault that occurs in the cell surrounded by quiescent cells that on its own accord assumes the single seed state. Under the current σ, this cell will induce the growth of a completely new and duplicate computational configuration.

To avoid this situation, a σ can be specified that requires a function $F \neq 0$ be assigned only to a cell whose neighborhood must contain at least one non-quiescent cell. Thus, when a cell surrounded by quiescent cells assumes an arbitrary state due to a fault, no reconfiguration will occur until the testing procedure discovers the fault and phagocytosis is initiated to isolate this faulty cell from the rest of the array. This can be expressed as $\sigma[c\ (0,\ldots,0)] = 0$ where $\xi_o \notin X$. Thus, an initial configuration in a TA must consist of at least two neighboring cells being forced out of quiescence simultaneously.

2.9 Pragmatics

An unusual set of problems pertaining to system input, output, clocking, and power, and other concepts not normally studied are generated by the fact that these functions must be carried out in some distributed fashion so that no single centralized function becomes the weak link in our fault tolerant computer. Some aspects of these problems can be solved with relative ease. For example, to distribute power to the array, it can be assumed that each cell has its own solar cell which is fed power from a highly reliable broadcast source of photon energy (the sun, for example). Other problems of this nature are not so easily solved and are also not investigated in this paper. However, it is assumed that there is a system clock available to all cells such that state changes of reconfiguration as well as computation can be carried out in a synchronous manner. It is further assumed that the only system input/output (I/O) is performed by way of communication through existing neighborhood links (or ports) between cells of the array.

3.0 CELL COMMUNICATION

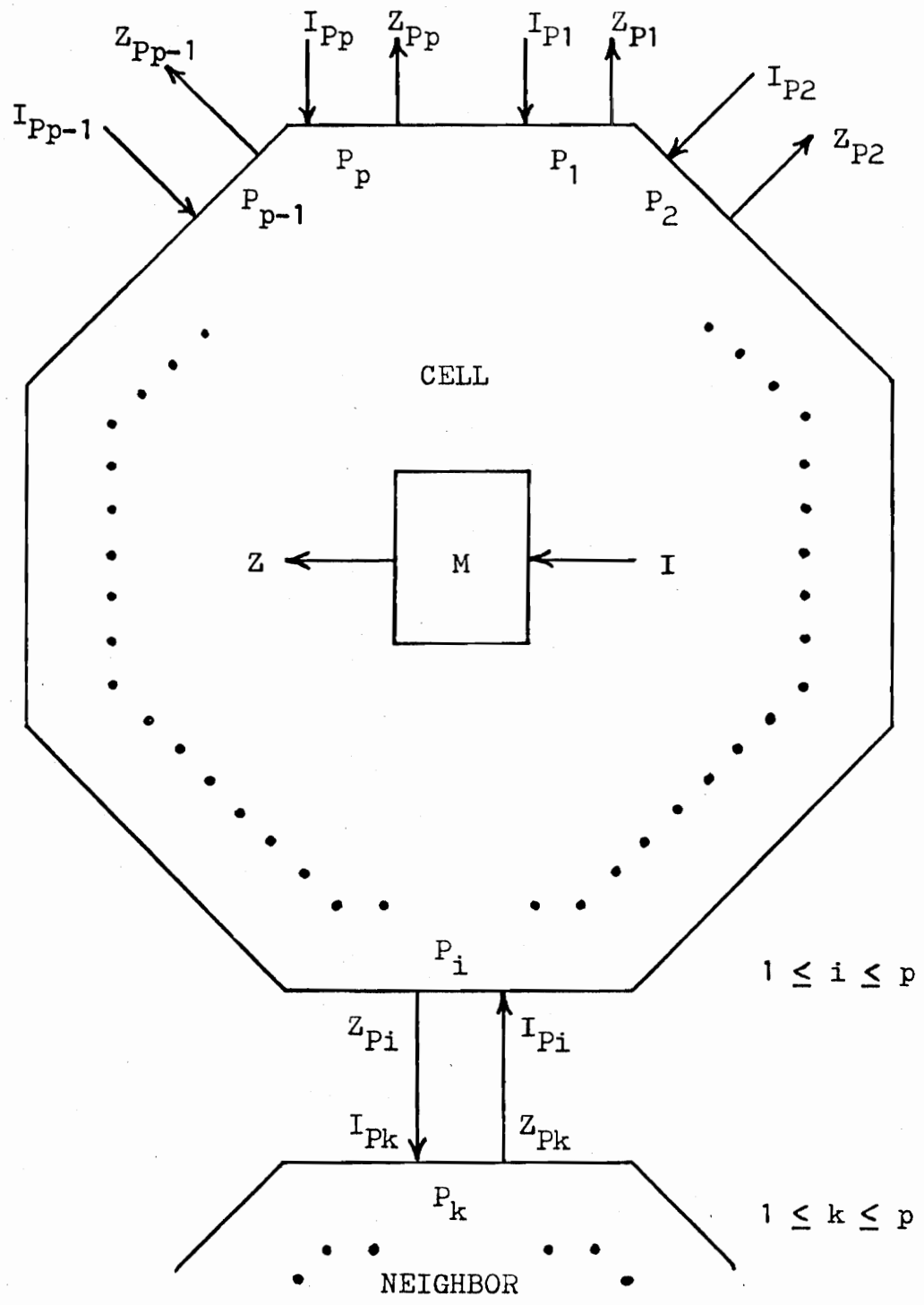
3.1 Ports

Each cell of a tessellation automaton contains an identical finite state machine $M = \langle Q,I,Z,\delta,\omega \rangle$ where Q is the state set, I is the input set, Z is the output set, and δ and ω are mappings such that δ: $Q \times I \rightarrow Q$ and ω: $Q \times I \rightarrow Z$.

Furthermore, each cell is connected to a finite number of neighbors in a uniform manner throughout the array. These connections necessitate some form of communication linkage between each cell and its neighbors. A port, P, is defined as the pair (I_p,Z_p) where $I_p \subseteq I$ and $Z_p \subseteq Z$ such that each port represents a communication linkage between each cell and its neighbors. The number of ports, p, which is identical for all cells, is equal to the number of neighbors that have unidirectional or bidirectional information moving to or from the cell. This means that the number of ports, p, is greater than or equal to the number of neighbors, n. The number of neighbors, n, is defined as those cells that provide only outputs to the cell j. The notion of a port is illustrated in Fig. 3.1-1.

3.2 Port Testability

Cell testability requires that the contained machine as well as all the ports of the cell be diagnosed from a source external to the cell. In order to test a port, all the communication links within the individual port must be used at some point during the diagnostic procedure being performed on the contained machine M.



$$I_{Pi} \subseteq I \qquad Z_{Pi} \subseteq Z$$

Figure 3.1-1. Cell interconnectivity via ports.

3.2.1 Single Port Testability

It is desirable that the finite state machine contained in a cell be fully diagnosable through each of its ports. This requires $I_p = I$ and $Z_p = Z$ so that access to all possible inputs and outputs of the finite state machine within the cell are accessible externally via any of the cell's ports. A cell diagnostic procedure that induces all possible state variables per linkage line (i.e. both 1 and 0 for a binary line) to occur when executed via a port tests the operability of both the contained machine and of the port in use. Thus, each time a diagnostic is performed, any existing fault that is present anywhere in the contained machine M or any fault that presently exists in the port in use for the test will be detected. Figure 3.2-1 illustrates that all outputs of the contained machine are bussed to each port, and the union of the input sets of each port becomes the input set to the contained machine. This union of input sets is accomplished by "ORing" the like input lines from each port to form the input line to the contained machine M.

3.2.2 Independent Port Testability

In the case where $I_p \subseteq I$ and $Z_p \subseteq Z$, it is still possible to test the contained machine provided that $I = \bigcup_{i=1}^{p} I_{Pi}$ and $Z = \bigcup_{i=1}^{p} Z_{Pi}$, and that $I_p = \{ x \in I \mid \text{for some } q \in Q,\ \omega(q,x) \in Z_p \}$. This requirement states that for a given set of outputs available at a given port, all inputs that can affect these outputs must also be available at this same port. Thus, a portion, or submachine, of the contained machine M can be tested through each port, and the contained machine M is considered to be

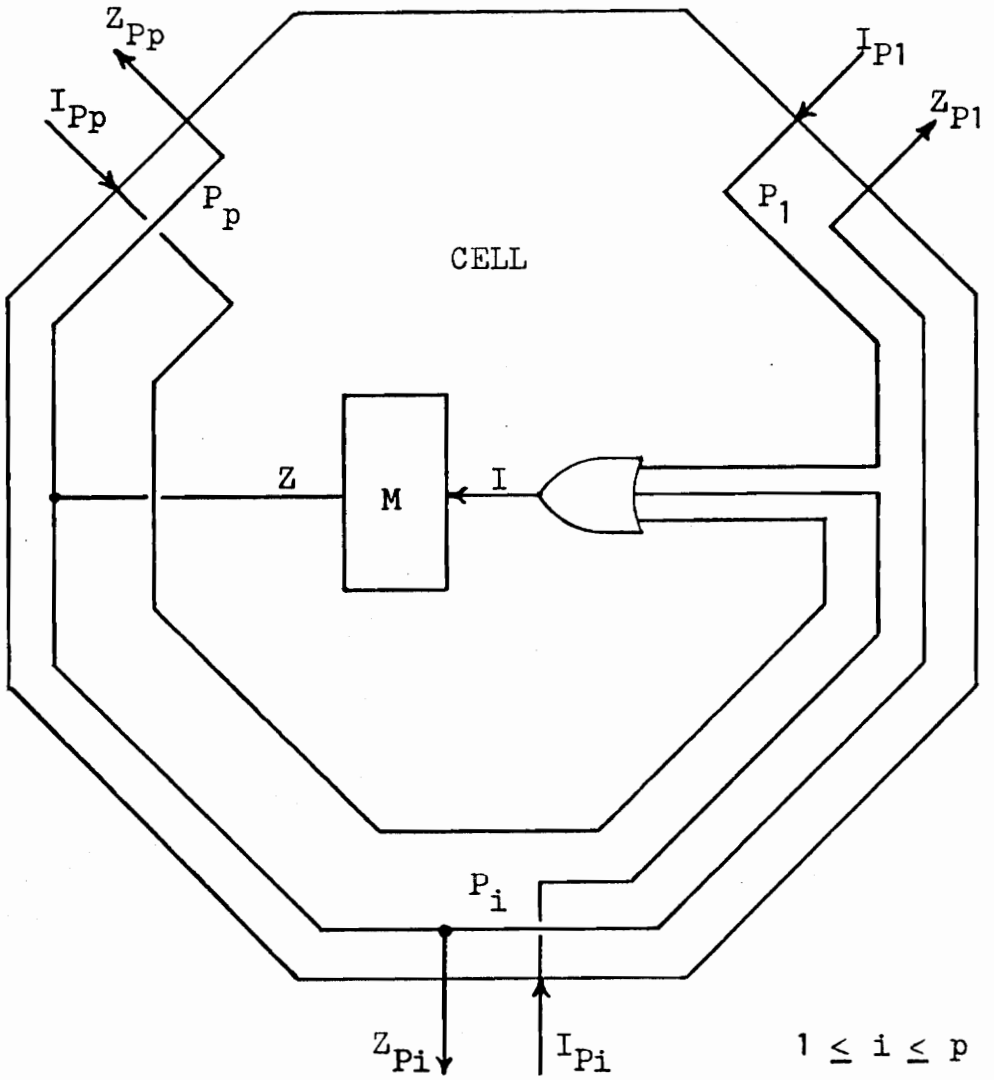


Figure 3.2-1. Cell showing ORing of inputs from ports to form the input to the contained machine M.

tested when all of its submachines have been tested. Independent port testing consists of a sequence of diagnostistics applied to all ports on a separate basis necessary for diagnosis of the contained machine and all the ports of the cell.

3.2.3 Co-Operative Port Testability

There exists a more general case where $I_p \subseteq I$ and $Z_p \subseteq Z$, and $I = \bigcup_{i=1}^{p} I_{Pi}$ and $Z = \bigcup_{i=1}^{p} Z_{Pi}$, with no other restrictions than these. To diagnose the contained machine M it may be necessary to apply inputs through one port and observe outputs from a different port. Thus, co-operation of neighbors is needed on a simultaneous basis to test the contained machine or a portion thereof through more than one port at the same time. Cell testing must then consist of one of the following procedures:

(1) a sequence of tests of submachines through multiple ports until the whole contained machine and all the ports of the cell have been tested, or (2) the internal machine is diagnosed via all ports of the cell simultaneously.

3.3 Port Technology Types

Some form of physical technology must be implemented to realize the ports of a cell so that the contained machine will have a definite communication channel over which information transfers can be made. For this discussion of technology types, it is assumed that the outputs of the contained finite state machine be electrical so that variations in voltage potential or electron flow can be used to represent elements

of the output set. Further, it is assumed that inputs to the contained machine be of the same electrical type as its outputs. The three technologies given are not the only possibilities, but they do represent three classes of technologies each with different fault resistent properties. Furthermore, in spite of the assumption of electrical linkage, there is no reason why a different medium or technology such as optical (photon) coupling could not be used which might present different fault tolerant properties.

3.3.1 Electronically Isolated Ports

One possible form of an electronically isolated port is shown in Fig. 3.3-1 where outputs from the contained machine consist of voltage levels bussed to each port (or wires). Like inputs from each port are "ORed" through electronic gates and form the input voltage level for each input of the contained machine. Electronic isolation could have just as easily been applied to the outputs of the cell as it has been applied here to the cell inputs.

3.3.2 Optically Isolated Ports

Figure 3.3-2 illustrates a possible circuit configuration making use of optically-coupled electronic devices. Inputs and outputs of the contained machine and the ports take on the form of a flow of current to carry information. For example, when a current flows out of the contained machine M in Fig. 3.3-2 on output wire W_1, photon emission occurs from the light emitting diode. This light is optically coupled to the three output phototransistors which are thus stimulated into the active region and pass a current to the output ports. A

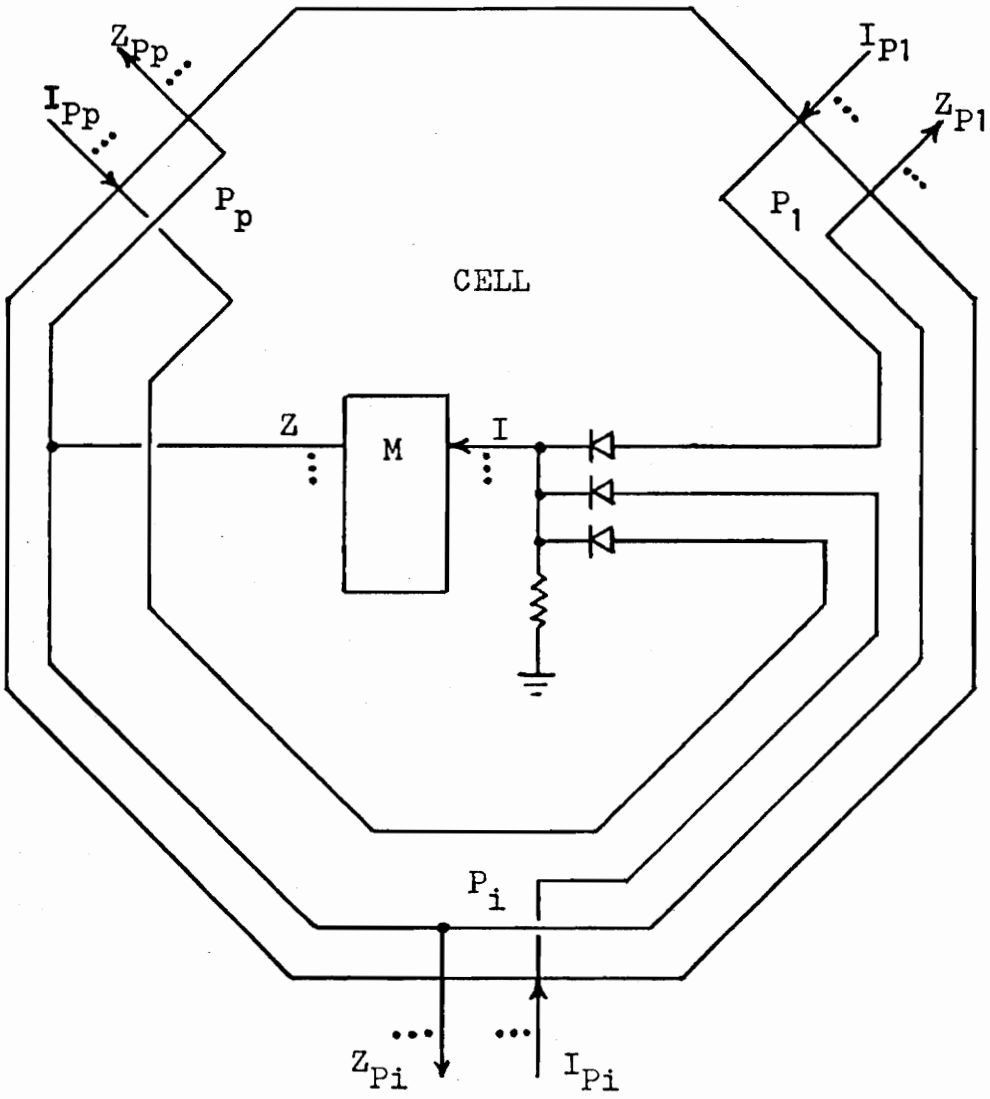
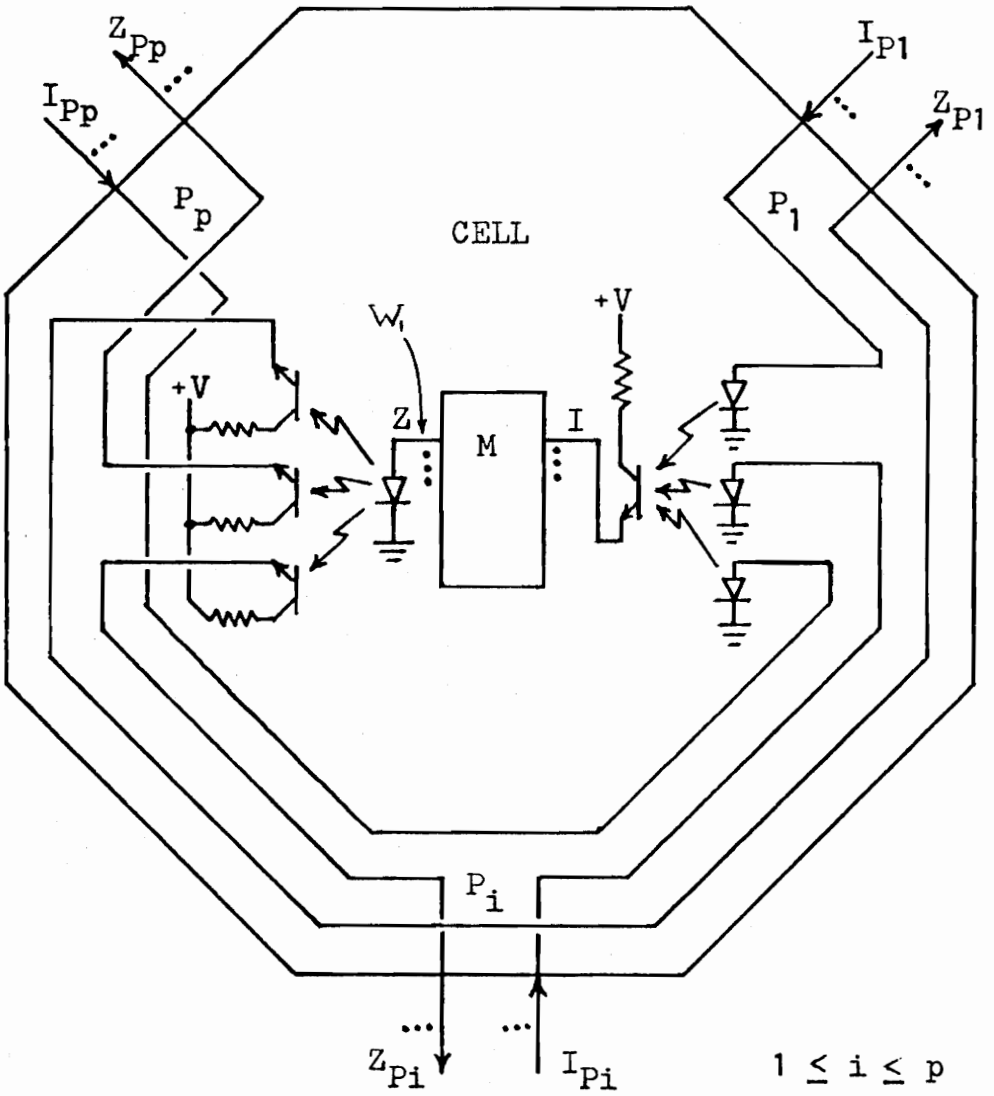


Figure 3.3-1. Cell with electronically (input) isolated ports.



+V is some positive voltage

Figure 3.3-2. Cell with optically isolated ports.

similar situation occurs for the inputs from the ports which activate light emitting diodes , anyone of which can cause the phototransistor to go into conduction. This type of technology allows the contained machine to be electrically isolated from all other machines while communications are maintained via an optical path.

3.3.3 Electromagnetically Isolated Ports

Electromagnetic technology can be used to implement signal coupling for the ports as shown in Fig. 3.3-3. In this case, input and output signals consist of pulses of current generated or to be sensed by the contained machine M. In the case of inputs, some type of pulse transformer is used so that a current pulse in a coil will induce a magnetic flux in a toroid or some core made of a substance that will allow magnetic flux to be propagated. This flux induces a current pulse in the remaining coils and is sensed by the contained machine M. Outputs from machine M cause a transistor to conduct thus setting up current paths to flow through the reverse isolation network of diodes. The diodes are necessary to negate the effects of the current pulses existing on the linkage wires due to induction in an input toroid which is in contrast to the only desired source of output current pulses which is the output from the contained machine.

Several fault and failure resistent properties are obtained when the input pulse transformer and the input current pulse sensors in the contained machine are properly designed. The core material and the number of turns on each coil should be selected so that the flux induced is at its saturation point in the toroid when a current pulse

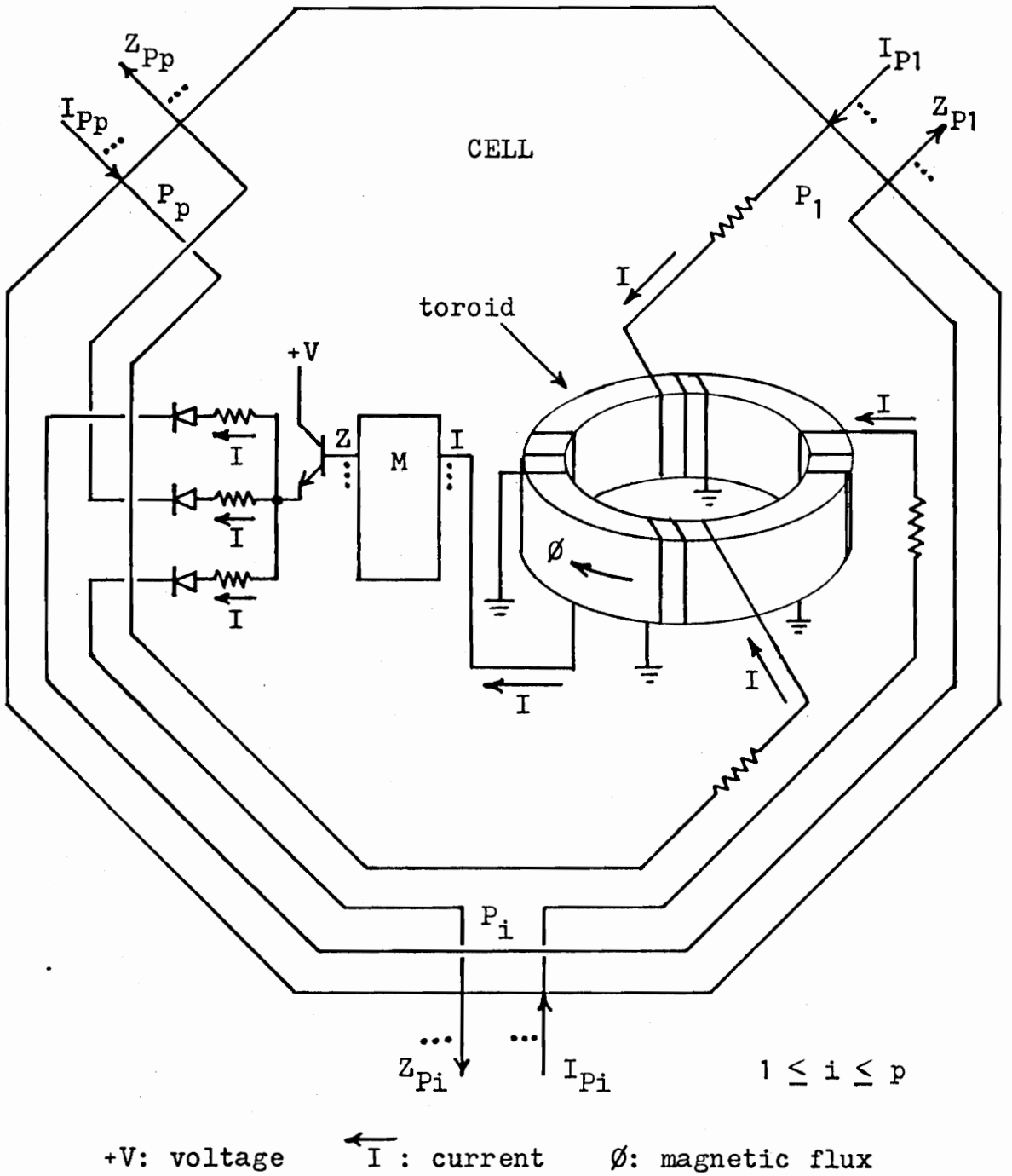


Figure 3.3-3. Cell with electromagnetically isolated ports.

is simultaneously present on each coil from all the ports. The current pulse sensor should be capable of detecting a current pulse induced by an input pulse from any number of ports that have windings on the toroid without incurring internal damage.

3.4 Linkage Faults

Two classes of faults can occur on the linkage wires that connect port to port: (1) stuck-at faults, and (2) intermittent faults. Stuck-at faults consist of the situation where a wire will come in permanent contact with some foreign voltage or current cource or by some other component failure that will appear as a permanent fault during some phase of testing procedure. In general, stuck-at faults are usually stuck-at-ground (s-a-0) or stuck-at-V (s-a-1) type voltage level conditions. Intermittent faults are any type of nonpermanent fault that can often not be distinguished from an actual valid signal that would normally occur on this same wire. None of the discussed port technologies have any intermittent fault resistant properties. However, each type of technology discussed exhibits different stuck-at fault resistant properties as well as different catastrophic failure propagation resistant properties.

3.4.1 Electronically Isolated Port Faults

The electronically isolated port cell as illustrated in Fig. 3.3-1 exhibits a resistance to stuck-at-ground (s-a-0) faults on any input wires to its cell ports. Thus, when one port input is grounded, input signals from other ports are still received correctly by the contained machine M. However, a stuck-at-V (s-a-1) fault on an input wire of any

port causes the voltage V to be permanently applied to the input of machine M. Furthermore, the output wires of the ports have no fault resistant properties. Thus, when a linkage wire develops a stuck-at-V (s-a-1) fault, the cell from which this is an output and all neighbors of this cell to which this output is connected are caused to be inoperative. This type cell has no failure propagation resistance properties because a failure inducing voltage or current surge might be passed electrically from one port to another right through machine M or on a port-to-port interconnection bus wire within the cell.

3.4.2 Optically Isolated Port Faults

The optically-coupled port cell which is shown in Fig. 3.3-2 has improved fault resistance properties over those of the electronically isolated port cell. The input circuitry has resistance to stuck-at-ground (s-a-0) faults but has no resistance to stuck-at-V (s-a-1) faults. However, the output circuitry exhibits resistance to all stuck-at faults existing on port output lines. Thus, the worst case effects due to a single stuck-at fault is the inoperability of only one cell. Furthermore, the electrical isolation properties of the optically-coupled devices can eliminate the cell-to-cell propagation of any injurious voltage or current surges.

3.4.3 Electromagnetically Isolated Port Faults

The electromagnetically isolated port cell shown in Fig. 3.3-3 incorporates fault resistant properties beyond those of both the electronically and optically isolated types. Because the current induced

in the coils depends only on a flux change in the toroid, all stuck-at faults are ignored by the input pulse transformer. The output diode and resistor networks also are uneffected by any stuck-at faults. The worst failure that can happen due to a stuck-at fault on a linkage wire is the inoperability of the wire only, which may not necessarily cause inoperability of any cells. Furthermore, the fact that the input toroid has been designed to saturate at excessive input current levels prevents the propagation of cell-to-cell voltage and current surges.

4.0 TEST CONFIGURATION

4.1 Cell Testing

Each cell, and the ports associated with each cell, must be diagnosed within the array in order to provide the property of "self-repair". The exact nature of a diagnostic procedure is dictated by the internal design and structure of the individual cell. In spite of this fact, it is still possible to describe general test configurations and scheduling schemes which allow for the application of the internal cell test without having to specify a particular cell design. Thus, with the goal of developing diagnostic procedures for a general cell, the following sections suggest various methods in which cells can be used to test other cells.

It is also desired that the testing procedures developed should be capable of performing their diagnostic tasks even in a faulty environment; that is, the test procedures must be fault tolerant. Some procedures offer superior fault tolerance over others; therefore, it is suggested that a procedure be selected not only for its efficiency and ease of implementation, but for its fault tolerant properties.

4.2 Cell Faults

Faults within cells are of two basic classes: (1) permanent, or (2) intermittent. Intermittent faults can only be detected if the fault is present in the cell during the exact phase of the test procedure that is designed to detect a fault in that precise location. Because intermittent faults can often not be differentiated from ordinary signals that might occur in the cell, no special effort is made

to detect them. Permanent faults will be present during the phase of a test that is designed to detect them. It is desirable that the specific cell testing procedure be structured so that any type of permanent fault that might occur within the cell be detected during some phase of a testing cycle.

Faults that occur within cells can have an adverse effect on the overall operation of the system. A fault (or faults) that occurs in the computational part of the cell can cause the loss of computational state information. This can lead to errors in the computational calculation that is taking place in this cell when the fault occurs. A single or multiple fault can also occur in the control part of the cell. Should this happen, it is possible that all of the control state information could be destroyed.

When the testing procedure discovers the faulty cell, the control hyperplane TA will initiate reconfiguration so that the functional integrity of the system can be restored. Thus, a spare cell will be mapped into the assigned function of what was previously the task of the now faulty cell. In this way, the assigned function in the form of control state information that was lost in the faulty cell is regained by the spare cell due to the mapping property based on the control state information held by the neighboring cells.

In the control part of a cell, any state information that is not recognized by the mapping σ of the TA can be considered a "substate". Once the control state information of a cell has been assigned by the TA, it might be necessary for the control part to assume internal

substates that will allow detailed control over the computational part so that this function assigned by the TA can be performed. If the control state information contained substates, a fault could destroy this information, and it would not be regained due to reconfiguration. Furthermore, reconfiguration will not restore any of the computational state information that was lost due to a fault.

4.3 Single Diagnoser Testing

This testing configuration involves the use of a single diagnoser cell to carry out the testing procedure. From a stationary position, the single diagnoser will be able to communicate with neighboring cells over only the unique port that connects them. Thus, no more than one port per neighbor can be tested when this neighbor is being exercised by this stationary diagnoser. Furthermore, because all cells are identical, the testing abilities of the diagnoser cell must be sufficient to diagnose another cell which exhibits the exact same internal structure as is contained in the diagnoser itself.

4.3.1 Single Port Testing

Single diagnoser testing is best suited for operation within an array of cells that exhibit the property of single port testability. In this way, any neighboring cell that offers the type of port necessary for single port testability (section 3.2.1) can be tested by the diagnoser. It is even possible to allow more than one neighboring cell to be diagnosed simultaneously by the same testing procedure. However, it may be easier to implement a procedure in which each each of neighboring cells is tested one by one in a staggered sequence as is

suggested in Fig. 4.3-1.

Let $X_t \subseteq X$ be the testing neighborhood index that is required to carry out a particular test or to allow a particular test procedure to be propagated through an array. Single port testing carried out by a single diagnoser imposes no requirements on the testing neighborhood index X_t. Any cell in the testing neighborhood index X_t of the single diagnoser is eligible to be tested.

4.3.2 Independent Port Testing

Single diagnoser testing is also capable of operation within an array of cells that have the property of independent port testability. Cells that offer the type of port required for independent port testability (section 3.2.2) can be tested in this way provided that each submachine or partitioned cell that is available through each separate port is capable of conducting a testing algorithm on the generally different submachine that is connected to this same port but is in a neighboring cell under test. In other words, the port that connects the single diagnoser to a neighboring cell under test really only connects a submachine of the diagnoser to a potentially different submachine of the cell under test. Thus, the submachine of the diagnoser must contain sufficient complexity to test the submachine of the cell under test at the opposite end of the port.

The notion of independent port testing via a single diagnoser is illustrated in Fig. 4.3-2. It is possible to conduct the testing procedure so that any number of neighboring submachines are tested by this single diagnoser. Furthermore, it is possible to design the

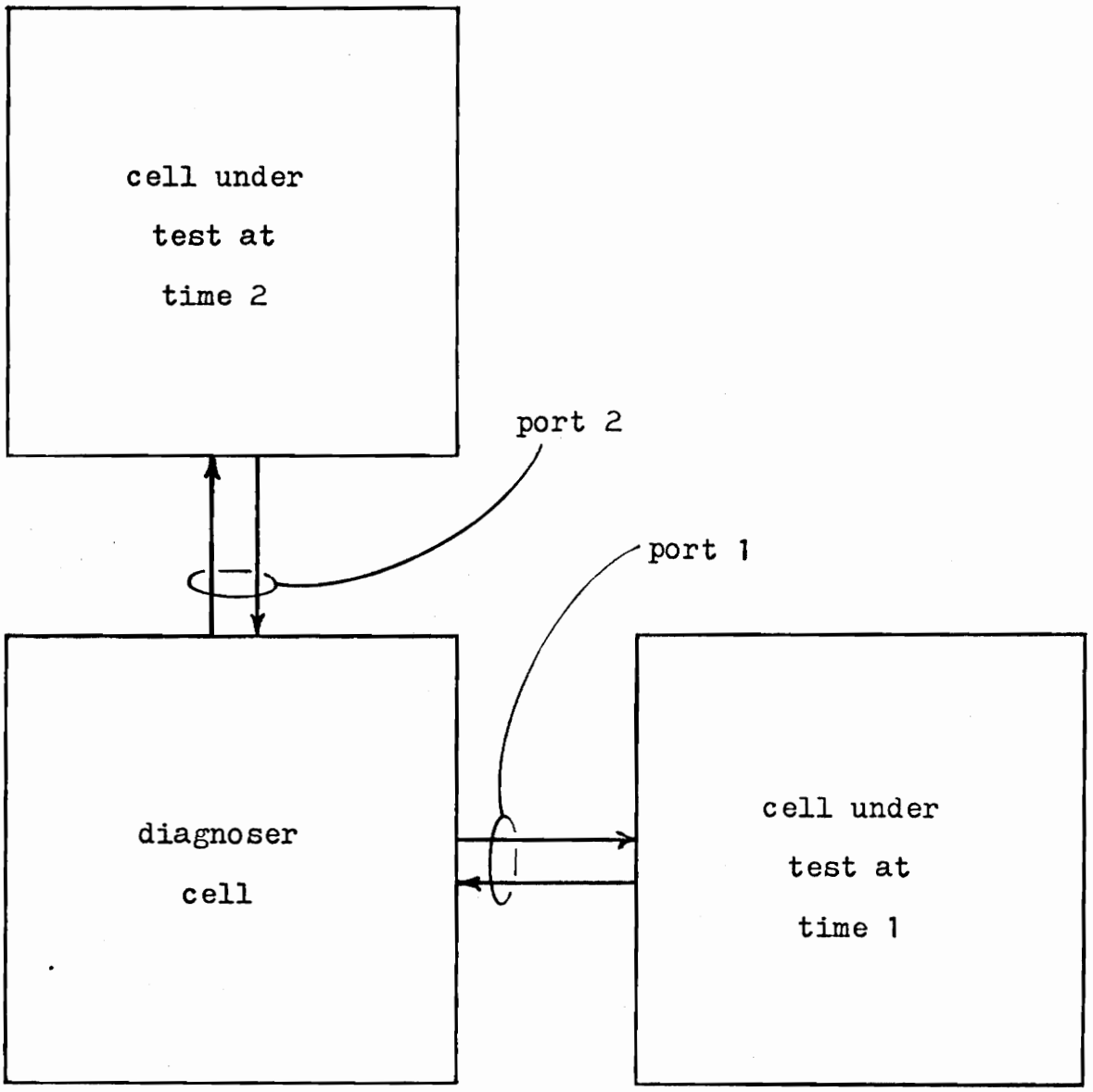


Figure 4.3-1. Single diagnoser testing showing staggered sequence diagnosis of single port testable neighboring cells.

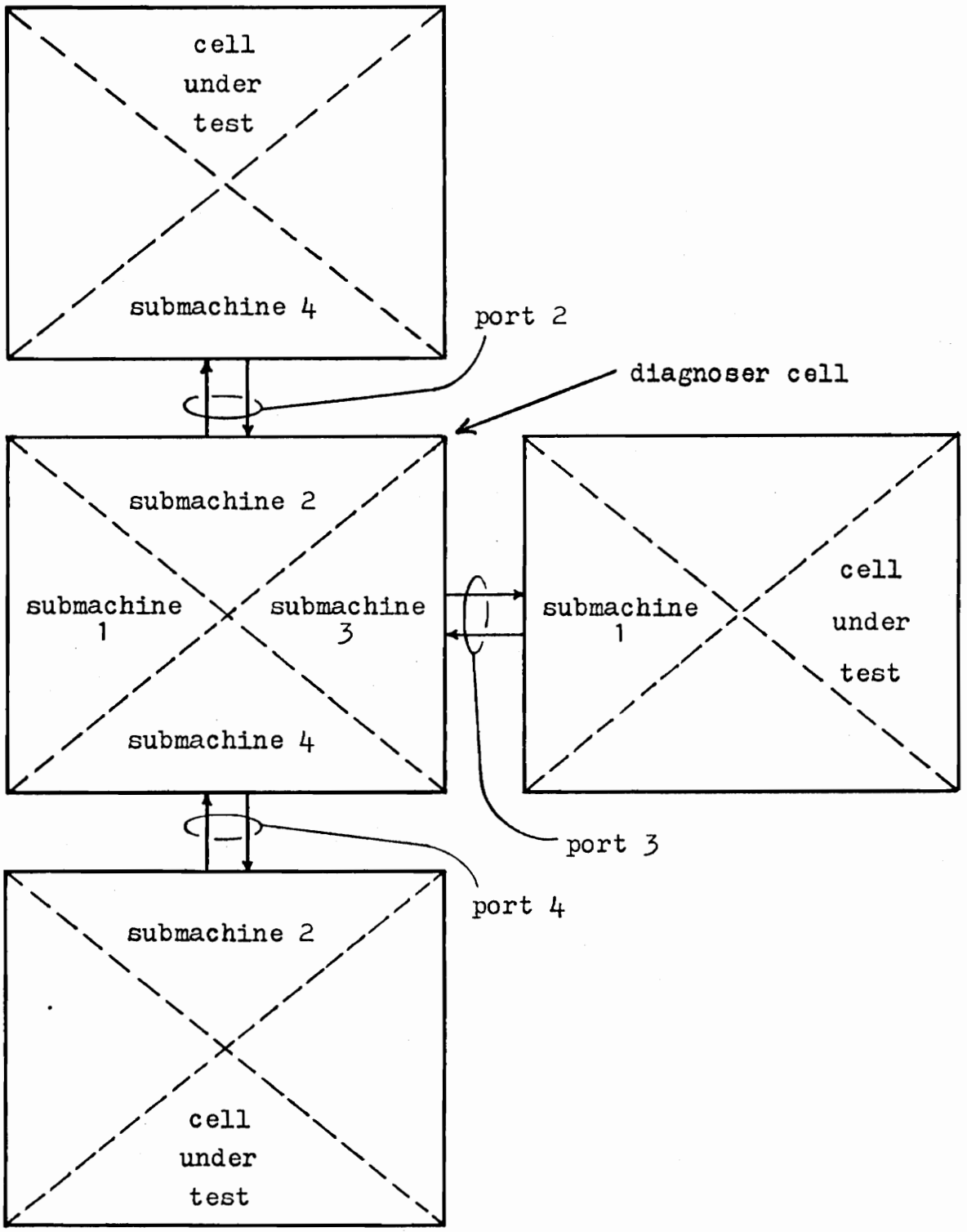


Figure 4.3-2. Independent port testing carried out by a single diagnoser cell.

specific test to either be carried out simultaneously on these neighboring submachines, or the testing can be sequentially staggered from one submachine to the next. Because of the possible differences in submachines to be tested, different tests must be provided by the diagnoser, and the proper test must be applied to the proper submachine in a predetermined sequence. This testing configuration does not impose any special requirements upon the testing neighborhood index X_t.

4.4 Multiple Diagnoser Testing

Multiple diagnoser testing is a configuration in which more than one cell is used to diagnose a particular cell under test. Because single diagnoser testing is simply a subset of multiple diagnoser testing, the latter is fully capable of performing any task that the former can execute. Furthermore, multiple diagnoser testing is capable of testing more than one port of the cell under test from the stationary test position. In fact, all of the ports, and the cell itself, of an array unit under test can be diagnosed simultaneously if the cell exhibits single port testability. This same configuration can be applied to cells that have the property of independent port testability so that each submachine and its port of the cell under test can be diagnosed by the stationary testing coalition.

In these cases where all ports of the cell under test are diagnosed from a stationary test configuration, it is necessary that the diagnoser cells occupy all neighboring positions of the cell under test. Thus, a restriction is placed on the testing neighborhood index X_t such that the testing neighborhood index must equal the

neighborhood index: $X_t = X$. If a multiple diagnoser testing configuration is used to diagnose a cell through a number of its ports which is less than the total number of its ports, no particular restriction is imposed on the testing neighborhood index X_t.

Multiple diagnoser testing can be used to test a wide range of possible cell internal structures. This is true because of the increased complexity of the testing configuration. By using multiple cells as diagnosers, cooperation can be established between these cells so that the testing abilities of each of these diagnoser cells is combined to form a testing unit which has much more testing power than the single diagnoser cell. Thus, this multiple diagnoser testing coalition has much more complexity than is contained by the single cell that is under test by this configuration.

4.4.1 Cooperative Port Testing

Cells which have a port structure which is of the cooperative port testability type (section 3.2.3) can only be tested via multiple diagnoser testing. At some point in the testing procedure of a cooperative port testability cell, it will be necessary that all of its neighbors be involved in the testing procedure. Thus, it is required that the testing neighborhood index be equal to the neighborhood index: $X_t = X$.

In the case of single and independent port testing via multiple diagnosers, it is not necessary that these multiple diagnosers actually be in communication with each other. The tests performed by this multiple diagnoser configuration do have to be sequenced in time so that they do not interfere with each other, but this can be made a built-in

function of the testing procedure without having to establish communication between diagnosers. When one of these diagnosers independently finds a fault, it can initiate reconfiguration on its own. However, in the case of cooperative port testing, it may be necessary to allow the multiple diagnosers to communicate with each other due to the possible need to apply the test inputs through one port to diagnose the test outputs from a different port.

Figure 4.4-1 can be used to illustrate the following example. Let $X_t = X = \{(0, 1), (1, 0), (0, -1), (-1, 0), (1, 1), (-1, -1), (-1, 1), (1, -1), (0, 0)\}$. Thus, for cooperative port testing, all eight neighbors of the cell under test are active diagnosers and must exercise the cell under test, as well as all eight of its ports. Communication between the diagnosers is provided by the rectangularly oriented ports that are between them.

4.4.2 Cooperative Port Testing with Indirect Links

Because of a particular neighborhood index X, it may be necessary to use cells that are not direct neighbors of the cell under test as indirect communication links between diagnosers. Let $X = \{(0, 1), (1, 0), (-1, 0), (0, -1), (0, 0)\}$. Figure 4.4-2 can be used to illustrate the following example. Because of the neighborhood index X, all the neighboring cells of the cell under test are diagnosers but they do not have direct communication paths between them. In this case, it is necessary to assign the corner cells with the function of indirect communication links as is illustrated by the dashed lines. In a sense, $X \subseteq X_t$ because the effective testing neighborhood index

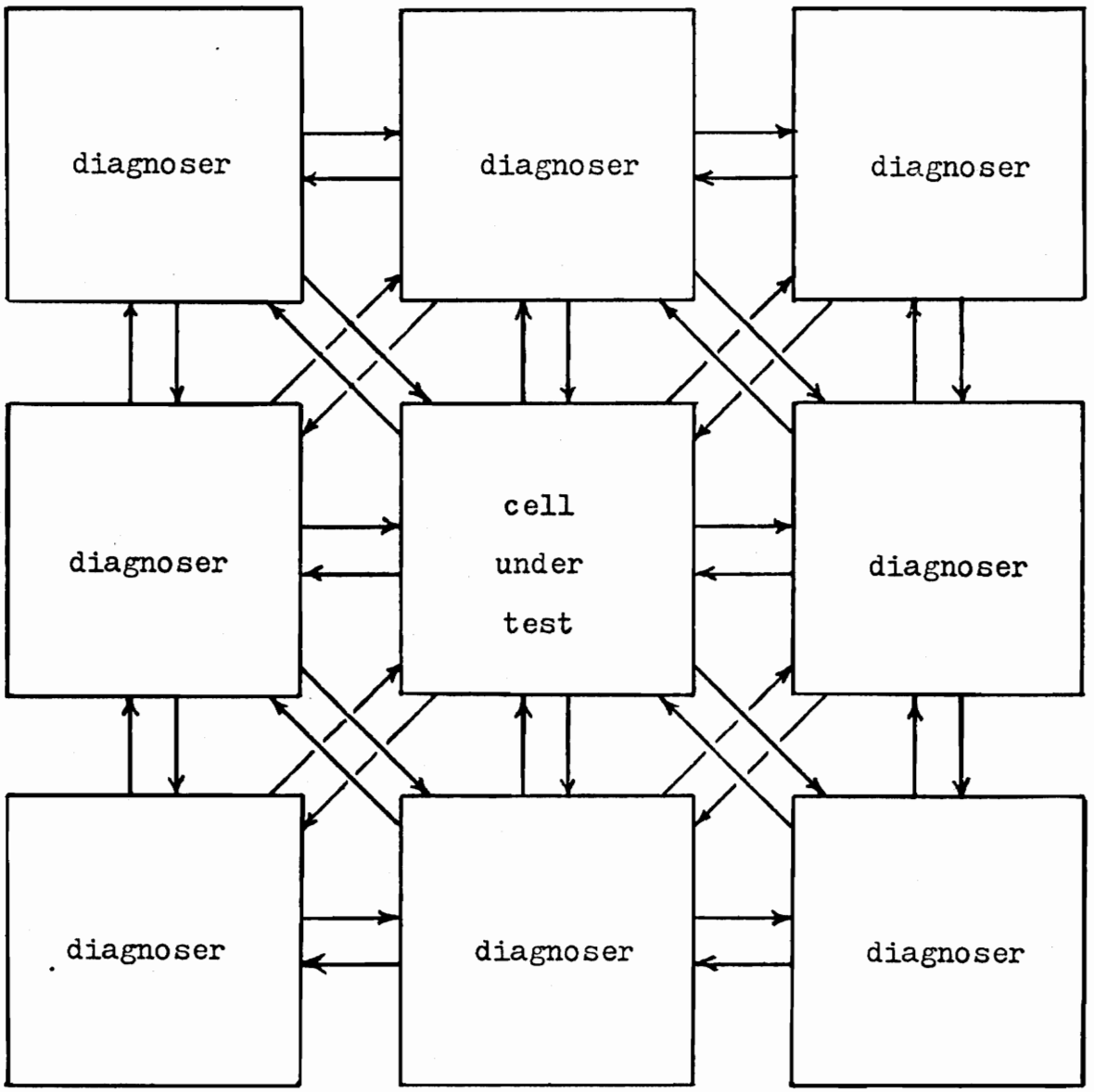


Figure 4.4-1. Multiple diagnoser testing configuration used in cooperative port testing with direct links.

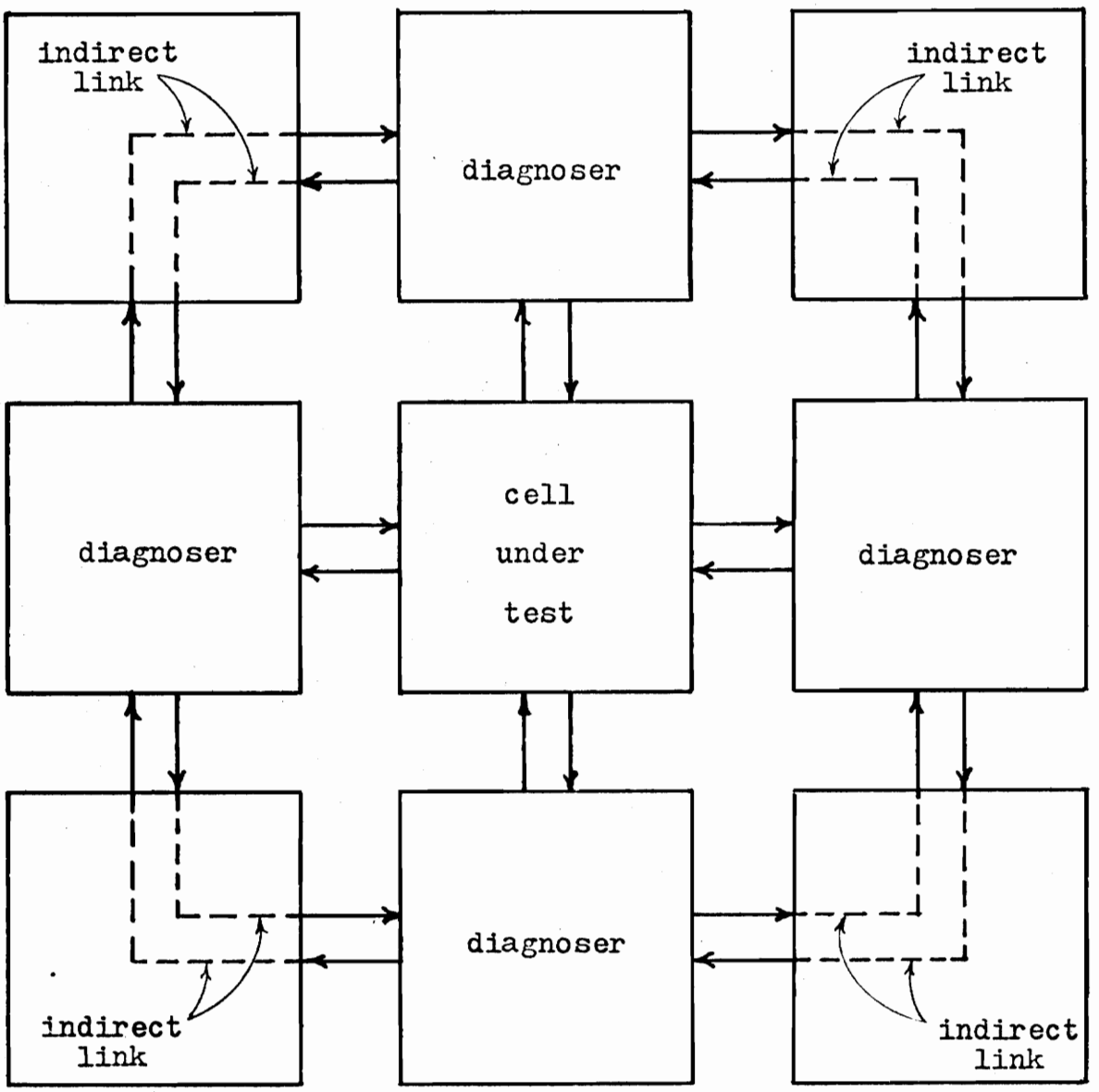


Figure 4.4-2. Multiple diagnoser testing configuration using indirect links for cooperative port testing.

is $X_t = X \cup \{(1, 1), (-1, -1), (-1, 1), (1, -1)\}$ due to the participation of the corner cells in the test configuration.

4.5 Comparison Testing

A heavy burden is placed upon the diagnoser cell which not only must generate the test inputs to a cell under test, but must also know the desired output due to each test input so that fault analysis can be made. To reduce this burden, it is possible to use a testing configuration where the diagnoser need not know the exact sequence of outputs that is supposed to result from its generated sequence of test inputs. Figure 4.5-1 illustrates a possible configuration, a single diagnoser cell generates a test input sequence which it applies to the cell under test and to three recently tested cells.

This diagnoser also contains a majority decision element which compares the output sequence of the three recently tested cells to produce a majority choice among them. Since these three template cells were recently tested and found fault free, it is assumed that a majority of these cells is still fault free. The output of this majority decision element is assumed to be the correct and desired output sequence for the given test input sequence. Thus, the output of the cell under test and this majority choice output can be compared for fault detection.

Comparison testing is best suited for cells that exhibit single port testability. It is also possible to use this test configuration on cells that are independent port testable provided that each submachine is identical as seen from any port of the cell (this is

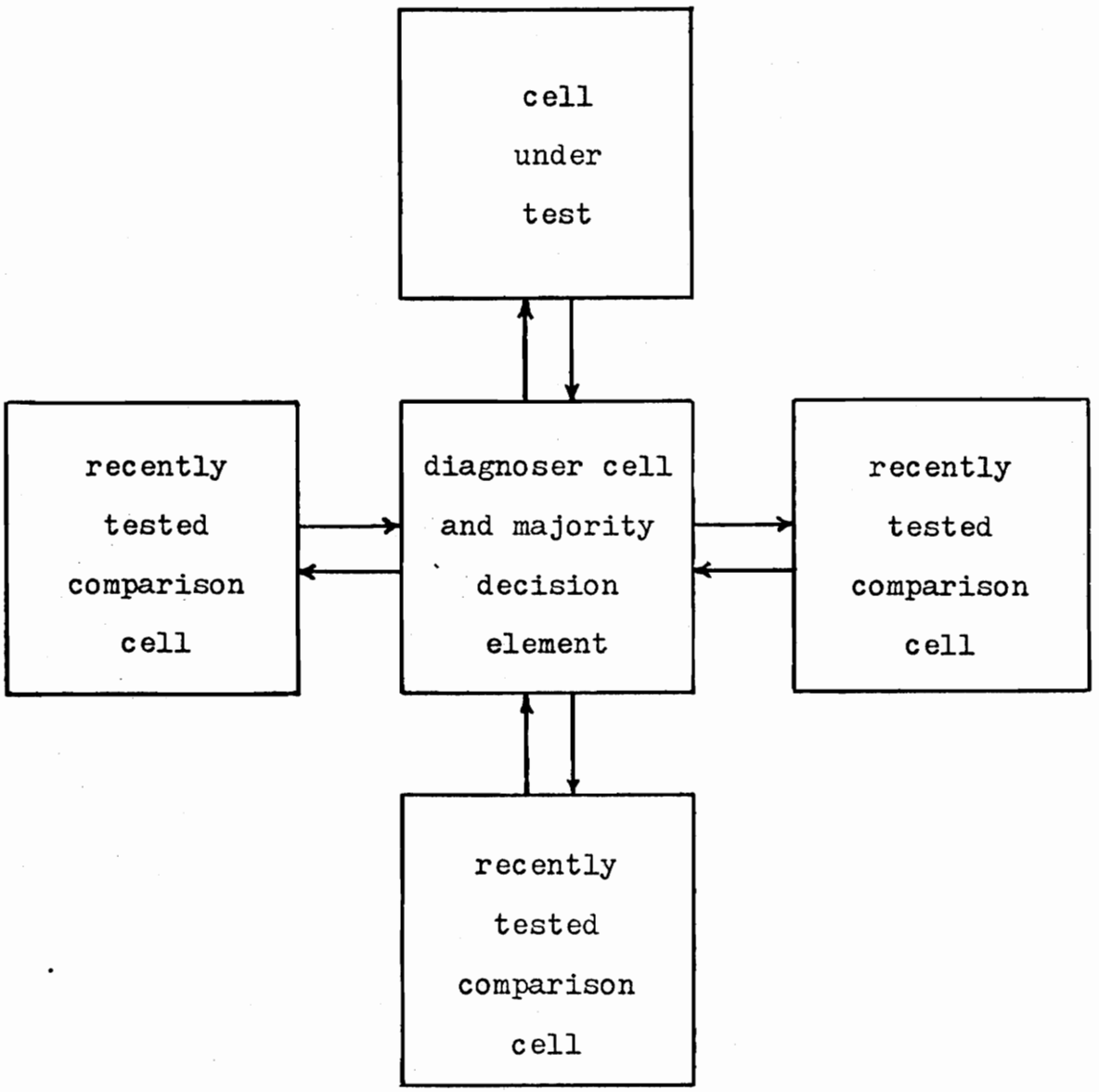


Figure 4.5-1. Comparison testing using three recently tested cells as output sequence generators.

unlikely). Comparison testing cannot be performed on cooperative port testable cells because it is necessary that the same test be applied to a cell via any of its ports to produce the same output sequence.

4.6 Reconfiguration Inhibitor

When a cell is under test, it can be expected to output a variety of control and computational states which is observed by the testing cell so that a determination can be made as to the faulty status of the cell based on the test inputs. This variety of outputs that result from the test inputs may also be present on ports to cells not involved in the testing procedure. Thus, neighboring cells could interpret the control state outputs of the cell under test as valid functional assignments within the TA. This could further lead to undesired reconfiguration of the neighboring cells not involved in the test due to the temporary control state outputs of the cell under test.

One method of solving this problem is to allow reconfiguration to take place only during specified time periods separated by longer intervals during which reconfiguration is inhibited. Thus, the testing procedure would have to be carried out during these reconfiguration inhibited periods so that temporary control state outputs could not initiate reconfiguration. When the reconfiguration "window" occurs following the test, the cell under test will have to have been left in a control state that will not initiate unwanted reconfiguration.

An alternate method involves the use of an additional control state output which serves the purpose of a reconfiguration inhibitor. When the testing procedure is initiated on a cell, the diagnoser must

first output a control state that causes the cell under test to map to a control state output that will not cause reconfiguration in any of its neighbors. Now the diagnoser can test the operability of this special reconfiguration inhibitor whose output line is common to all ports. Once the reconfiguration inhibitor has been found to be functional, it is caused by the daignoser cell to assume an inhibited state. This then signals all of the neighboring cells which receive outputs from the cell under test to ignore any control state outputs from this cell. Now the diagnoser can fully exercise the cell under test without the outputs that result from the test causing an unwanted reconfiguration. Of course, it is still possible for the reconfiguration inhibitor line of a port to a neighbor to contain a fault that will be undiscovered by the diagnoser. In this cause, unwanted reconfiguration could still be initiated by the testing procedure.

4.7 Testing Hyperplane

It may be necessary to alter the basic concept of our structural fabric to help facilitate the desired process of system self testing. An additional "testing" hyperplane could be placed in parallel to the already existing control and computational hyperplanes. Thus, each cell would contain a testing part that was connected to the control and computational parts of this same cell. This testing part could be assigned the specific and singular function of conducting the testing procedure. This would free some of the burden in terms of cell complexity from the control and computational parts that previously had to administrate and conduct the testing procedure.

5.0 SEQUENTIAL TEST SCHEDULING

5.1 Sequential Test Scheduling

In an array in which only one cell (or a few cells) is tested at a time, it is necessary that the testing procedure be propagated through this array so that each contained cell can be eventually tested. In this way, a given test configuration is moved from cell to cell in a predetermined pattern of motion until all the cells of the array have been tested.

5.1.1 Test Propagation

The propagation of a testing pattern is a function of the TA within the array. The cells that are part of a testing configuration are assigned control states that indicate the nature of their particular testing function. Le $A_t \subseteq A$ represent these testing control states whose only function is to represent the state of a cell which performs a task in a testing configuration or of states assigned to cells necessary in the propagation of this pattern. Thus, the test can be propagated simply by reconfiguration of the array by means of the mapping σ.

It is desired, however, that reconfiguration necessary for test propagation only occurs once the testing procedure is completed for the current cell under test. One way to accomplish this is to make use of the reconfiguration inhibitor output from the diagnoser cells. This would inhibit reconfiguration until the diagnosers had completed the testing procedure and released the reconfiguration inhibitor output. Now, the TA can reconfigure to propagate the testing configuration to

its next desired position. An alternate method is to assign a control state to a diagnoser that is internally modified depending on the particular phase of the testing procedure. For example, a cell is assigned the control state T which causes it to begin a testing procedure as a diagnoser cell. When it has completed the test, it internally alters its control state to state D which signifies the test is done. When the TA sees this control state output from the diagnoser, reconfiguration is initiated in its neighbors which results in propagation of the test configuration.

The final control state of the diagnoser cell also depends on the result of the applied testing procedure. If a fault were found, this final state should be one that would initiate reconfiguration around the newly discovered faulty cell. Two different control states are needed to differentiate between whether phagocytosis or quarantining is necessary for the faulty cell.

5.1.2 Array Geometry

In an ideal system, it would be necessary to propagate a given test configuration throughout an infinite array. In a practical system, however, it is necessary to propagate a test configuration within a finite array. A finite array can be constructed from an infinite array by simply cropping out a finite section of the infinite array. This truncated infinite, or quasi-infinite, array can still appear to be infinite to the test propagation scheme. This is because tests which propagate out to a boundary of this quasi-infinite array can be made to vanish just as if they had continued to propagate in the same direction

within the now nonexistent part of the array that would have existed if the array were infinite.

In a finite array it is possible to assign a function to the boundary cells that would "reflect" the propagating test pattern much like a mirror reflects a propagating photon. Thus, a test could propagate within this finite array and be bounced back into this array when it encountered a line of reflection cells at a boundary. In this way, a single propagating test configuration could be used to continually test the cells of the array. A major disadvantage of this scheme is that it is possible for reflection cells to occur in unwanted locations due to some form of fault. Thus, it is possible for the propagating test to get stuck between valid and invalid reflectors so that it is bounced back and forth locally and cannot escape to test the rest of the array.

It is also possible to alter the basic geometry of a finite array into some non-Euclidean form. For example, boundaries of a finite array can be connected externally to the array so that they appear as neighbors with no discontinuity. Thus, a test which propagates to one boundary would simply disappear there and reappear at the boundary which is externally connected to this other boundary. A Karnaugh map is a good example of this because boundary rows are considered adjacent and boundary columns are considered adjacent.

5.1.3 Propagation Mechanism

When propagation is implemented via the TA, there exists a question as to exactly what information need be moved so that the new testing configuration can be established. Should the propagated information be

a "test me" or "test them" type of signal? Either of these two types of state information could be used to propagate a test. However, there exists a danger in the "test me" form of signal. If this information is passed to a cell to be tested that is already faulty, it is possible that this information would be lost due to the fault. Thus, the propagation of the test would be halted. For this reason, it is safer to propagate a "test them" mechanism from cells conducting the testing procedure to recently tested cells which are to assume the testing procedure. Thus, when the test configuration is to be propagated to a new location, the task of diagnoser is assigned to cells that have been recently tested. This reduces the chances of halting a propagating test due to a fault.

5.1.4 Reconfiguration

While a test is propagating through an array, it will eventually encounter the current active computational configuration of the array. Because it is necessary that these cells of the computational configuration be tested, the mapping σ must allow the test configuration to move into this computational configuration to test its cells without destroying the current calculation in progress.

This can be done by reconfiguration of the computational configuration around the testing configuration.

During this reconfiguration, it is necessary that the current computational state information of all cells of the computational configuration being moved, be shifted to the cells that will now assume the previously performed tasks of these cells. In this way, the current

calculation of the computational configuration will not be destroyed. Once this reconfiguration has taken place, the testing configuration can move in and test a cell that was once part of the computational configuration, but is now unassigned. Before the next cell of the computational configuration is tested, this process of reconfiguration must be repeated. Furthermore, it is desired that the computational configuration not grow beyond its initial size once the propagating test has moved away. If this were not so, the computational configuration might grow larger with every pass of the propagating test and would quickly outsize the finite array within which it was situated. Therefore, the mapping σ should be designed so that the computational configuration be returned to its exact previous structure once the propagating test has moved away provided no faulty cells were discovered.

These requirements can invoke the necessity of a huge mapping σ to carry them out. Furthermore, a fairly complex neighborhood index may be necessary to allow this reconfiguration around the propagating test. The severity of these disadvantages is a function of the complexity of the computational configuration and of the type of propagating test being used.

As an alternative to reconfiguration, it may be possible to design the computational configuration in redundant "macrostructures". A propagating test could then be formulated to test only one macrostructure at a time. As the test propagates through a macrostructure, the state information of its cells will be destoyed; therefore, the current calculation taking place in that macrostructure will also be

destroyed. Once the propagating test has left this first macrostructure, the state information that is held in its duplicate macrostructure can be transferred to an area already tested by bridging over or moving around the propagating test in a sort of "macroreconfiguration." In this way, the calculation that was destroyed in the first macrostructure is recovered from the duplicate calculation of the redundant macrostructure. The propagating test can now start testing this redundant macrostructure and will thus destroy its calculation. However, this calculation is preserved by the new macrostructure caused by the macroreconfiguration into the already tested area.

5.1.5 Configuration Propagation

Certain test configurations offer better fault tolerant properties than others when they are propagated. Single port testing via a single diagnoser advances to a cell just tested by the previous test configuration. When this is done, the reliability of test propagation is high because only cells recently tested are used in the process of propagation. This also applies to single port testing via multiple diagnosers provided the new diagnosers are chosen only from recently tested cells. Comparison testing also fits into this category provided that the cells used to generate the comparison output sequences are chosen only from recently tested cells.

Independent port testing and cooperative port testing require that the function of diagnosis be assigned to cells that have not been recently tested or contain ports that have not been recently tested. Thus, cells that have not been recently tested and may contain faults

are critically involved in the propagation process. For this reason, the probability of these testing configurations being propagated without causing a halting situation is less than that of the single port testing and comparison testing configurations.

5.2 Test Propagation in Infinite Arrays

Any scheme of test propagation that allows all the cells of an infinite array to be diagnosed is acceptable. Furthermore, any propagating test suitable for an infinite array and any propagating test suitable for use in a quasi-infinite array are fully interchangeable. Because of the very nature of an infinite array, every propagation scheme will be a "one shot" affair. The test will be initiated at some location in the array and will then have to propagate out and away from this point in order to test all the cells of this array. Two examples of these type tests are given in the following two sections.

5.2.1 Concentric Test Propagation in E^2

The testing scheme presented here involves the propagation of a single diagnoser cell which performs single port testing in an infinite or quasi-infinite array of dimensionality 2. (It is also possible to design a similar test for an array of arbitrary dimensionality.) The propagation is started by a single seed initial configuration at any given point of an infinite array or at the center cell of a quasi-infinite array. The test configuration propagates in the form of concentric diamond shaped rings that expand away from the location of the initialization seed. In this way, all the cells of an infinite or

quasi-infinite array are eventually covered during each propagating test initiated per seed. Table 5.2-1 lists the specifications for a TA in E^2 that will perform this test propagation in an array of all quiescent cells. Figure 5.2-1 shows the progress of this propagating test from its initial seed through several generations.

Only a fairly small testing control state alphabet A_t is needed for this scheme. In spite of the relatively simple testing neighborhood index X_t listed in Table 5.2-1, a much more complex neighborhood index will be required so that the computational configuration will be able to reconfigure by "bridging over" this propagating test pattern. This is a problem because the expanding concentric rings are continuous and will split a computational configuration into partitions which must maintain ports over the partitioning ring for continued functioning of the computational configuration. This propagation scheme can further be described as being of the indeterminate multiplicity type because the number of diagnoser cells is not constant in spite of the fact that one port testing is being carried out by single diagnosers; it is just that there are many of these single diagnosers operating at the same time.

As the pattern propagates, the following test configuration must be carried out during each generation. The diagnoser cells designated S must test the cells which are above, below, left, and right of their present positions. The diagnoser cells designated U must test the cells which are above, left, and right of their present positions. The diagnoser cells designated D must test the cells which are below, left,

Table 5.2-1

TA Specifications for Concentric Test Propagation in E^2

$A_t = \{0,S,U,D\}$

$X_t = \{(0,1), (0,-1), (1,0), (-1,0)\}$

$c'(j) = \sigma \left[c\ (N(X_t, j))\right]$

σ: SOOO $\rightarrow$ D
OSOO $\rightarrow$ U
OOSO $\rightarrow$ S
OOOS $\rightarrow$ S
OUOO $\rightarrow$ U

DOOO $\rightarrow$ D
OUUO $\rightarrow$ U
OUOU $\rightarrow$ U
DOOD $\rightarrow$ D
DODO $\rightarrow$ D

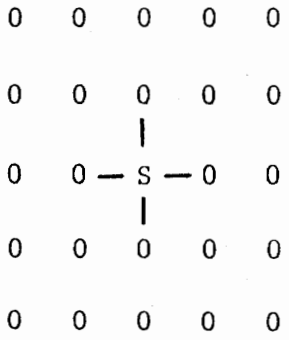
SODO $\rightarrow$ D
SOOD $\rightarrow$ D
OSUO $\rightarrow$ U
OSOU $\rightarrow$ U

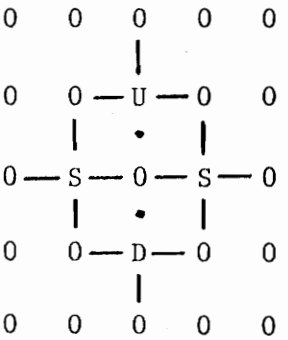
all other possibilities map to 0

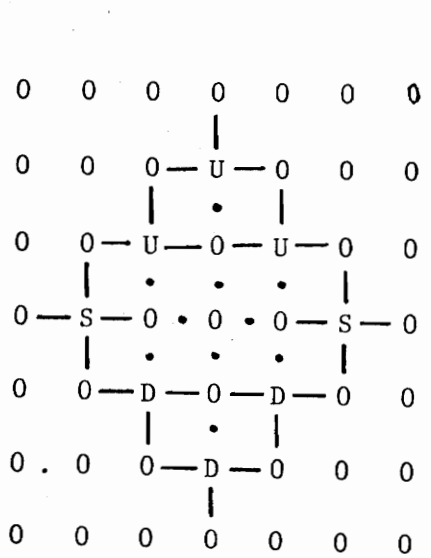
In figure 5.2-1:

— = a port tested during the present generation

• = a port tested during a previous generation







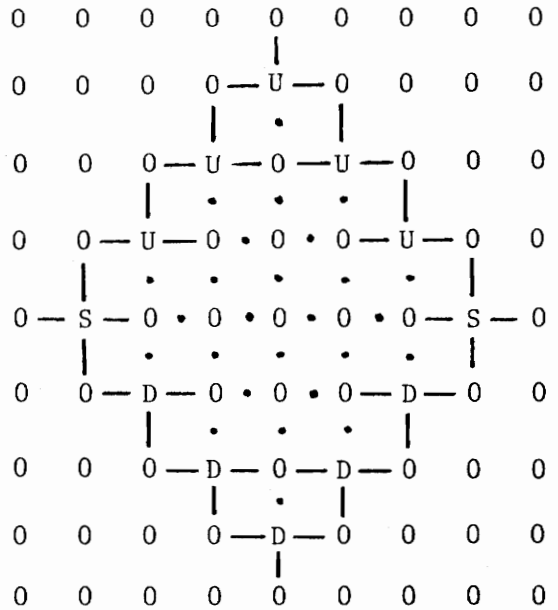


Fig. 5.2-1. Generations of the concentric test propagation in E^2.

and right of their present positions. This testing sequence will diagnose all the ports of the array and all the cells of the array in such a way that the next generation of diagnosers occupies only cells that were just tested by the previous generation. This sequence is valid only for the testing neighborhood index X_t given in Table 5.2-1. Once a more complex neighborhood index is specified, it will be necessary to alter the test configuration requirements so that these additional ports will also be tested as the procedure propagates through the array.

5.2.2 Spiral Test Propagation in E^2

This testing scheme involves the propagation of a single diagnoser cell which performs single port testing in an infinite or quasi-infinite array of dimensionality 2. (Again, it is also possible to design a similar test for use in an array of arbitrary dimensionality.) The propagation is started by a single seed initial configuration at any given point of an infinite array or at the center cell of a quasi-infinite array. The test configuration propagates in the form of a spiral starting at this initial point and moving clockwise while forming rings of increasing radius. In this way, all the cells of an infinite or quasi-infinite array are eventually tested during each test sequence per seed initialization. Table 5.2-2 lists the specifications for a TA in E^2 that will perform this test propagation in an array of quiescent cells. Figure 5.2-2 shows several generations of this propagating test starting with the initial seed configuration.

Like the concentric test propagation scheme, it will be necessary to add more complexity to the neighborhood index X than is suggested

Table 5.2-2

TA Specification for Spiral Test Propagation in E^2

$A_t = \{0,D,U,L,R,d,u,l,r\}$

$X_t = \{(-1,1),(0,1),(1,1),(-1,0),(0,0),(1,0),(-1,-1),(0,-1),(1,-1)\}$

x is the don't care state: $x \in A_t$

t is any test "guidance" state: $t \in \{d,u,l,r\}$

$c'(j) = \sigma \left[c(N(X_t, j)) \right]$

σ:
0000D0000 → D
00000D000 → d
tD0x00x00 → D
xxt00L000 → L

00x00x0Ut → U
000R00txx → R
xt00D0000 → L
00x0Lx000 → U
0000U00tx → R

000xR0x00 → D
xxxxd0xxx → d
xxxx1xx0x → 1
xxx0uxxxx → u
x0xxrxxxx → r

xtxxLxxxx → l
xxxxUtxxx → u
xxxxRxxtx → r
xxxtD0xxx → d

all other possibilities map to 0

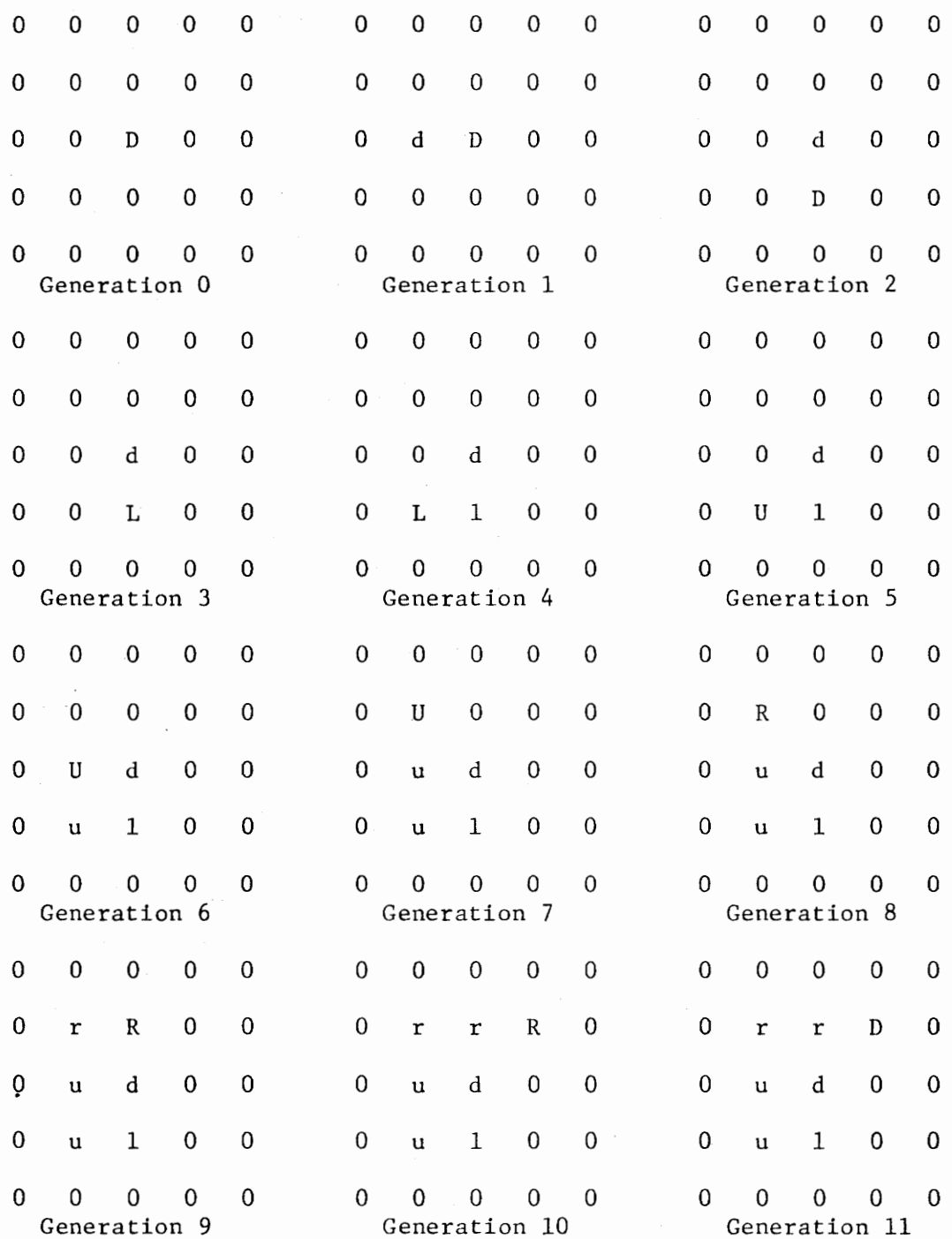


Fig. 5.2-2. Spiral sequential test propagation.

```
0 0 0 0 0      0 0 0 0 0      0 0 0 0 0
0 r r d 0      0 r r d 0      0 r r d 0
0 u d D 0      0 u 0 d 0      0 u 0 d 0
0 u 1 0 0      0 u 1 D 0      0 u 1 d 0
0 0 0 0 0      0 0 0 0 0      0 0 0 D 0
0 0 0 0 0      0 0 0 0 0      0 0 0 0 0
  Generation 12      Generation 13      Generation 14

0 0 0 0 0      0 0 0 0 0      0 0 0 0 0
0 r r d 0      0 r r d 0      0 r r d 0
0 u 0 d 0      0 u 0 d 0      0 u 0 d 0
0 u 1 d 0      0 u 1 d 0      0 u 0 d 0
0 0 0 L 0      0 0 L 1 0      0 L 1 1 0
0 0 0 0 0      0 0 0 0 0      0 0 0 0 0
  Generation 15      Generation 16      Generation 17

  0 0 0 0 0 0 0          0 0 0 0 0 0 0
  0 0 0 0 0 0 0          0 0 0 0 0 0 0
  0 0 r r d 0 0          0 0 r r d 0 0
  0 0 u 0 d 0 0          0 0 u 0 d 0 0
  0 0 u 0 d 0 0          0 0 u 0 d 0 0
  0 L 1 1 1 0 0          0 U 1 1 1 0 0
  0 0 0 0 0 0 0          0 0 0 0 0 0 0
      Generation 18          Generation 19
```

Fig. 5.2-2 (con'd). Spiral sequential test propagation.

```
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 r r d 0 0
0 0 u 0 d 0 0
0 U u 0 d 0 0
0 u 1 1 1 1 0
0 0 0 0 0 0 0
```

Generation 20

```
0 0 0 0 0 0 0
0 U 0 0 0 0 0
0 u r r d 0 0
0 u 0 0 d 0 0
0 u 0 0 d 0 0
0 u 1 1 1 0 0
0 0 0 0 0 0 0
```

Generation 21

```
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 U r r d 0 0
0 u 0 0 d 0 0
0 u 0 0 d 0 0
0 u 1 1 1 0 0
0 0 0 0 0 0 0
```

Generation 22

```
0 0 0 0 0 0 0
0 r R 0 0 0 0
0 u r r d 0 0
0 u 0 0 d 0 0
0 u 0 0 d 0 0
0 u 1 1 1 0 0
0 0 0 0 0 0 0
```

Generation 23

```
0 0 0 0 0 0 0
0 R 0 0 0 0 0
0 u r r d 0 0
0 u 0 0 d 0 0
0 u 0 0 d 0 0
0 u 1 1 1 0 0
0 0 0 0 0 0 0
```

Generation 24

```
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 r r d 0 0
0 U u 0 d 0 0
0 u 0 0 d 0 0
0 u 1 1 1 0 0
0 0 0 0 0 0 0
```

Generation 25

Fig. 5.2-2 (cont'd). Spiral sequential test propagation.

by the testing neighborhood index X_t of Table 5.2-2. This is due to the fact that it is necessary for the computational configuration to reconfigure by bridging over the continuous rings of the propagating test just as was the case of the concentric scheme. This propagation arrangement can further be described as having the attribute of unique multiplicity because only one diagnoser exists in the array at any given time.

As the pattern propagates, the following test configuration must be carried out during each generation. The diagnoser cells designated D, U, R, and L must test each of their neighboring cells that are not assigned a control state which is an element of the testing state alphabet A_t. Thus, any neighboring quiescent cells are tested by the diagnoser per generation. This testing configuration will diagnose all the ports and cells of the array for the given testing neighborhood index X_t of Table 5.2-2. Even if the neighborhood index X is made more complex to allow reconfiguration to bridge over the propagating test rings, this test configuration will ensure that the additional ports attached by the larger neighborhood index will also be tested.

5.2.3 Modified Spiral Test Propagation in E^2

Modifications can be made to the spiral test propagation so that continuous rings are not formed. This releases the requirement that additional complications be made to the neighborhood index X so that reconfiguration can take place in the computational configuration. The modification removes all the "stable" cells except the ones at the "corners" of the ring that were previously used as a "guidance rail"

for the propagating test.

Table 5.2-3 lists the specifications for a TA in E^2 that will allow propagation of this modified procedure through an array of quiescent cells. Figure 5.2-3 illustrates several generations of this modified scheme. The same test configuration requirements that apply to the spiral test propagation also apply to the modified version with the addition of the diagnoser cell designated S to the list of D, U, L, and R type cells. Because the propagating test consists of isolated cells, the computational configuration can reconfigure around the test using only the testing neighborhood index X_t. Furthermore, it is possible to assign the "corner footprint" state information as a sub-state of a cell taking an active part in a computational configuration so that reconfiguration around them is unnecessary.

5.2.4 Sequential Test Generators

Because the propagating tests in infinite or quasi-infinite arrays are one-pass tests, it is necessary to generate a seed configuration each time it is desired to initiate the propagation of a test. This can be done by placing a configuration in the center of an array that has a timing sequence of control states and can also initiate a test propagation. This "sequential test generator" thus cycles through a number of control states via the mapping σ so that a specific period of time can be measured. At the end of this control state sequence, the desired propagating test is initiated also via the mapping σ. The sequential test generator then goes back into the timing sequence of control states. In this way, sequential propagating tests can be

Table 5.2-3

TA Specification for Modified Spiral Test Propagation in E^2

$A_t = \{0,U,D,L,R,S,u,d,l,r,s\}$

$X_t = \{(-1,1),(0,1),(1,1),(-1,0),(0,0),(1,0),(-1,-1),(0,-1),(1,-1)\}$

x is the don't care state: $x \in A_t$

$c'(j) = \sigma[c(N(X_t,j))]$

σ:

0S0000000 → L
0000S0000 → s
0s00L0000 → l
00s00L000 → U
00s0Ul000 → u

00000s0ul → U
0000000Us → R
0000R000s → r
000R000s0 → R
000R00s00 → D

0000D0s00 → d
0D0s00100 → D
sD0100000 → D
0ul00L000 → L
ul000L000 → L

00000u0U0 → U
0000000Uu → U
0000000U0 → U
00r0000U0 → U
00000r0U0 → U

dD0000000 → D
0D0000000 → D
0D0000100 → D
0D0100000 → D
0D0d00000 → D

σ:

xxxxs0xxx → s
xxxxlxx0x → l
xxxxd0xxx → d
xxx0uxxxx → u
x0xxrxxxx → r

00r0000Uu → U
000R00r0d → R
0000000Ur → R
00u00L000 → U
1D0000000 → L

000R00d00 → D
00u0U0000 → u
0000D0d00 → d
0000R000r → r
1000L0000 → l

01000L000 → L
00100L000 → L
00000L000 → L
u0000L000 → L
0u000L000 → L

000R00r00 → R
000R00000 → R
000R0000d → R
000R000d0 → R
000R000r0 → R

all other possibilities map to 0

```
0 0 0 0 0      0 0 0 0 0      0 0 0 0 0
0 0 0 0 0      0 0 0 0 0      0 0 0 0 0
0 0 S 0 0      0 0 s 0 0      0 0 s 0 0
0 0 0 0 0      0 0 L 0 0      0 U 1 0 0
0 0 0 0 0      0 0 0 0 0      0 0 0 0 0
  Generation 0     Generation 1     Generation 2

0 0 0 0 0      0 0 0 0 0      0 0 0 0 0
0 0 0 0 0      0 R 0 0 0      0 r R 0 0
0 U s 0 0      0 0 s 0 0      0 0 s 0 0
0 u 1 0 0      0 u 1 0 0      0 u 1 0 0
0 0 0 0 0      0 0 0 0 0      0 0 0 0 0
  Generation 3     Generation 4     Generation 5

0 0 0 0 0      0 0 0 0 0      0 0 0 0 0
0 r 0 D 0      0 r 0 d 0      0 r 0 d 0
0 0 s 0 0      0 0 s D 0      0 0 0 0 0
0 u 1 0 0      0 u 1 0 0      0 u 1 D 0
0 0 0 0 0      0 0 0 0 0      0 0 0 0 0
  Generation 6     Generation 7     Generation 8

0 0 0 0 0      0 0 0 0 0      0 0 0 0 0
0 r 0 d 0      0 r 0 d 0      0 r 0 d 0
0 0 0 0 0      0 0 0 0 0      0 0 0 0 0
0 u 1 0 0      0 u 1 0 0      0 u 0 0 0
0 0 0 L 0      0 0 L 1 0      0 L 0 1 0
0 0 0 0 0      0 0 0 0 0      0 0 0 0 0
  Generation 9     Generation 10    Generation 11
```

Fig. 5.2-3. Modified spiral sequential test propagation.

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	r	0	d	0	0
0	0	0	0	0	0	0
0	0	u	0	0	0	0
0	U	0	0	1	0	0
0	0	0	0	0	0	0

Generation 12

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	r	0	d	0	0
0	U	0	0	0	0	0
0	0	0	0	0	0	0
0	u	0	0	1	0	0
0	0	0	0	0	0	0

Generation 14

0	0	0	0	0	0	0
0	R	0	0	0	0	0
0	0	r	0	d	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	u	0	0	1	0	0
0	0	0	0	0	0	0

Generation 16

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	r	0	d	0	0
0	0	0	0	0	0	0
0	U	u	0	0	0	0
0	u	0	0	1	0	0
0	0	0	0	0	0	0

Generation 13

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	U	r	0	d	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	u	0	0	1	0	0
0	0	0	0	0	0	0

Generation 15

0	0	0	0	0	0	0
0	r	R	0	0	0	0
0	0	r	0	d	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	u	0	0	1	0	0
0	0	0	0	0	0	0

Generation 17

Fig. 5.2-3 (cont'd). Modified spiral sequential test propagation.

```
0 0 0 0 0 0 0
0 r 0 R 0 0 0
0 0 0 0 d 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 u 0 0 1 0 0
0 0 0 0 0 0 0
```

Generation 18

```
0 0 0 0 0 0 0
0 r 0 0 0 D 0
0 0 0 0 d 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 u 0 0 1 0 0
0 0 0 0 0 0 0
```

Generation 20

```
0 0 0 0 0 0 0
0 r 0 0 0 d 0
0 0 0 0 0 0 0
0 0 0 0 0 D 0
0 0 0 0 0 0 0
0 u 0 0 1 0 0
0 0 0 0 0 0 0
```

Generation 22

```
0 0 0 0 0 0 0
0 r 0 0 R 0 0
0 0 0 0 d 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 u 0 0 1 0 0
0 0 0 0 0 0 0
```

Generation 19

```
0 0 0 0 0 0 0
0 r 0 0 0 d 0
0 0 0 0 d D 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 u 0 0 1 0 0
0 0 0 0 0 0 0
```

Generation 21

```
0 0 0 0 0 0 0
0 r 0 0 0 d 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 D 0
0 u 0 0 1 0 0
0 0 0 0 0 0 0
```

Generation 23

Fig. 5.2-3 (cont'd). Modified spiral sequential test propagation.

initiated on a regular basis.

5.3 Test Propagation in Modified Geometry Arrays

Because the boundaries of a finite array can be externally interconnected in a variety of ways, there can be a large number of possible test propagations designed for this use. A particular modified geometric array that offers good test propagation properties is the skew torus which is conceptualized in Figs. 5.3-1(a) and (b). The boundary interconnections for a 4 x 4 skew torus are shown in Fig. 5.3-2. Of course, the size of the torus can be increased provided that the style of boundary interconnections indicated in Fig. 5.3-2 is maintained. The reason this geometrical space is useful is because the boundary interconnections cause a once discontinuous two dimensional array to become a single surfaced continuous entity. Furthermore, the skew torus preserves its neighborhood index X. This is indicated in Fig. 5.3-3 which is a conformal mapping of the continuous skew torus of Fig. 5.3-2 onto an infinite two-dimensional array.

A single diagnoser test configuration can be propagated through this geometry simply by letting the new diagnoser be located in the position above (or below) the old diagnoser which has just tested that exact location. If the diagnoser also tests the cell to the right (or left) of its current position during each generation, all the ports of this array will also be tested. This propagating test need be initiated only once because it will continuously propagate within the torus and will repeatedly test all its cells and ports.

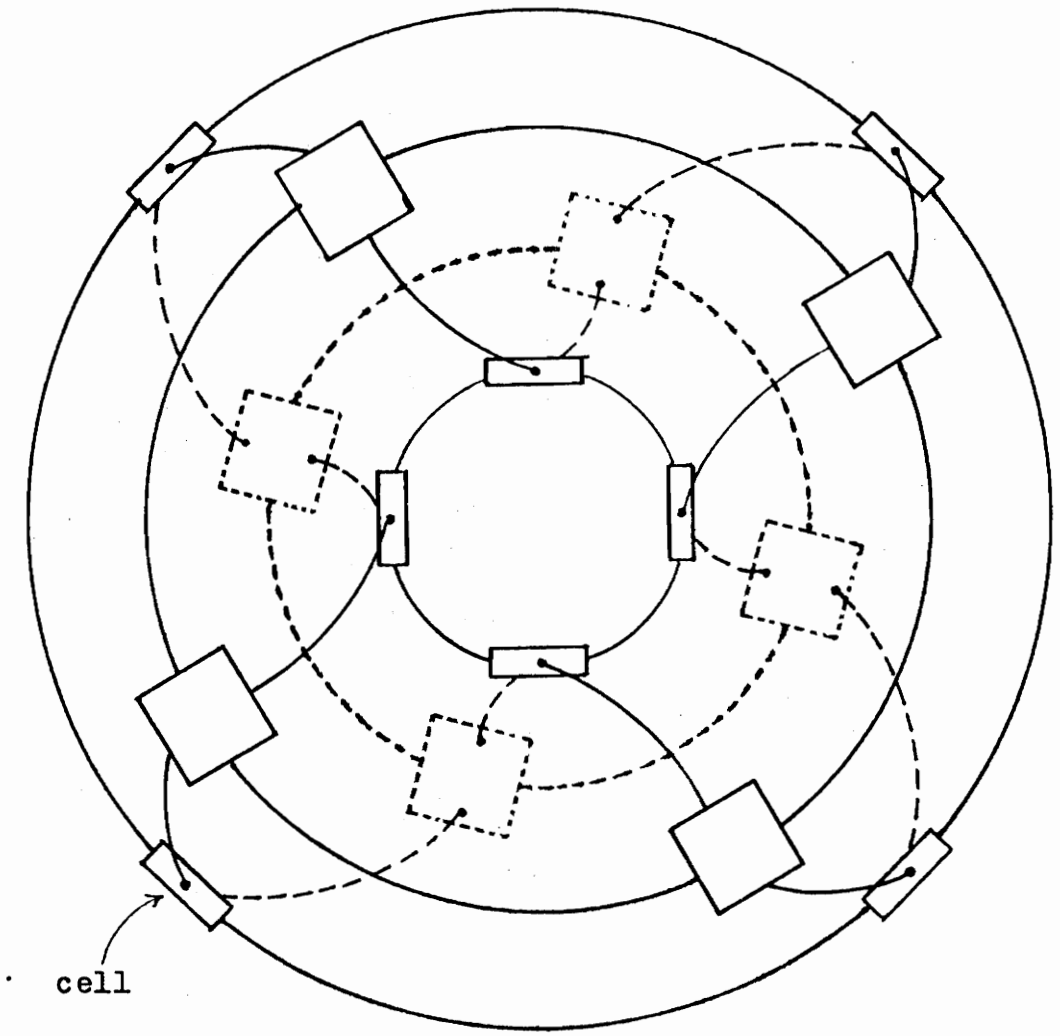


Figure 5.3-1 (a). Conceptualized representation of a 4 x 4 skew torus.

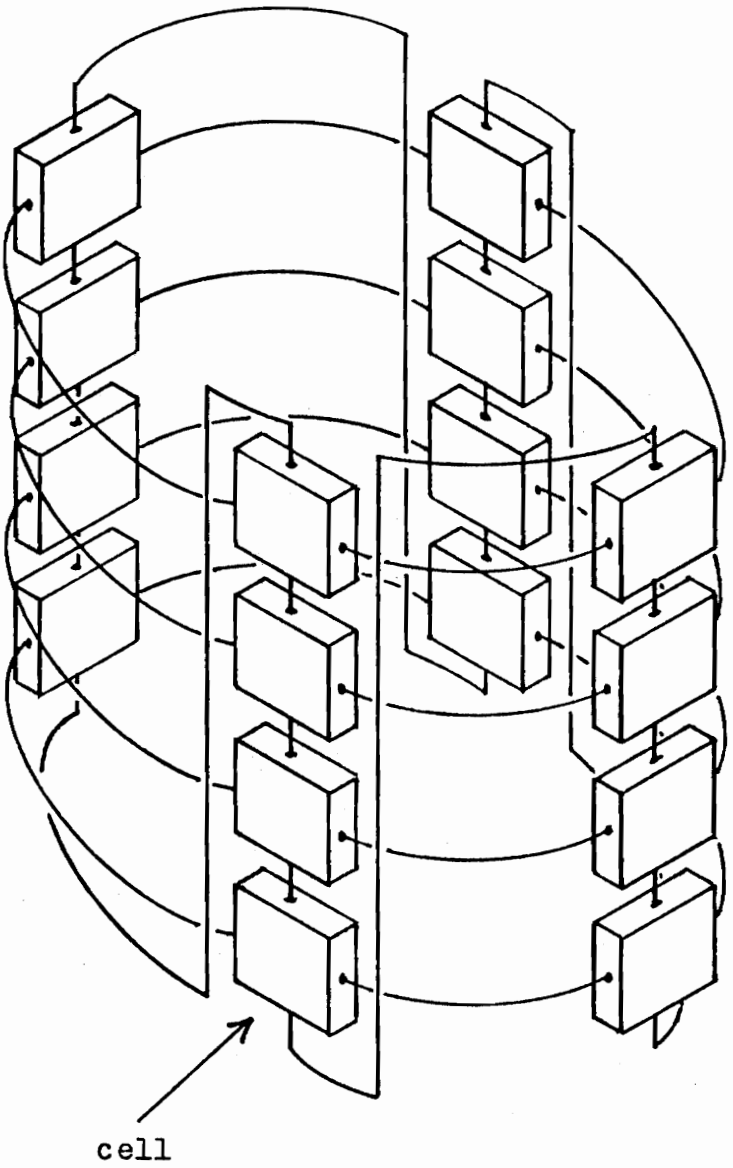


Figure 5.3-1 (b). Cylindrical conceptualization of a 4 x 4 skew torus.

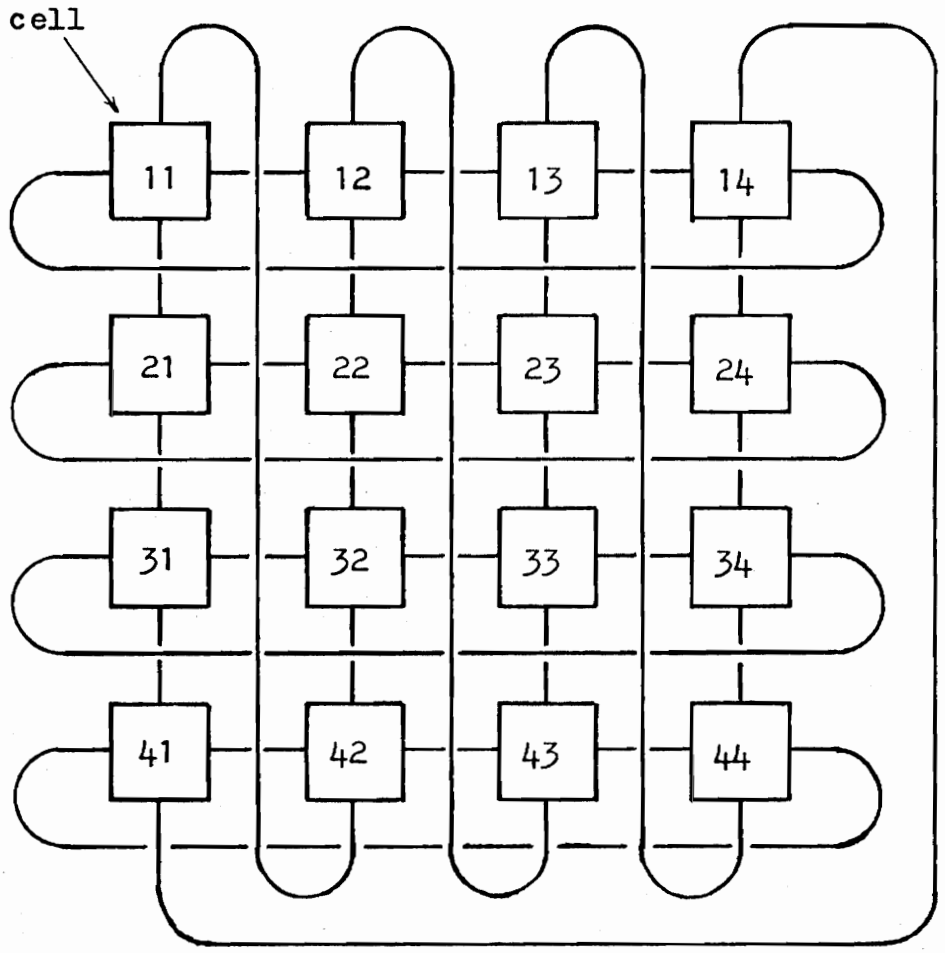


Figure 5.3-2. Boundary connections of a 4 x 4 skew torus.

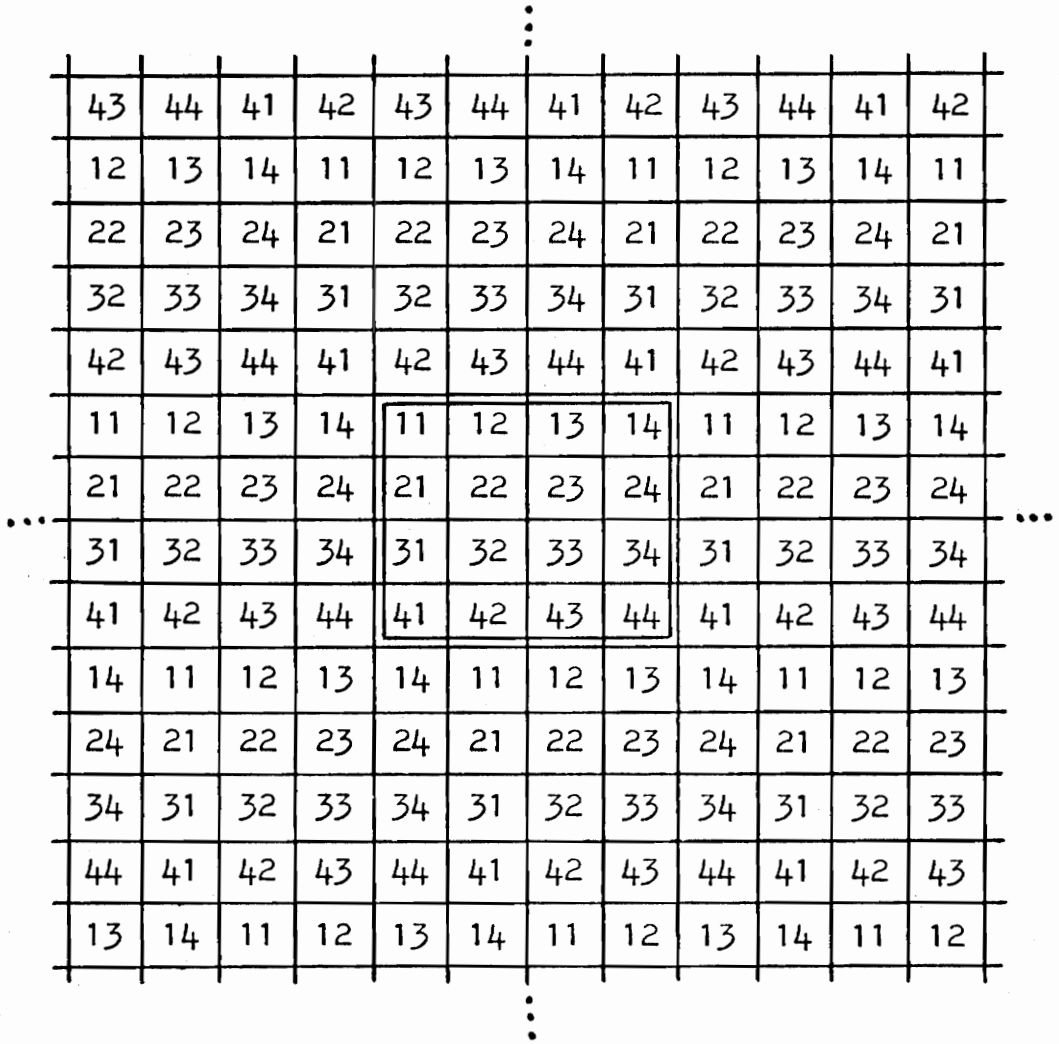


Figure 5.3-3. Conformal mapping of a 4 x 4 skew torus onto a two dimensional infinite array.

Because the skew torus maps onto an infinite two-dimensional array that preserves the neighborhood index, any theory applicable to a TA in E^2 will still apply to this geometry. Furthermore, because the propagating sequential test consists of a single diagnoser configuration, no special neighborhood index is required to allow the computational array to reconfigure around the test procedure.

5.4 Propagation Perturbation

A major problem with sequential testing is how a propagating test is to "get around" a nonreconfigurable cell such as a quarantined cell. Once the propagating test pattern reaches a quarantined cell (or cells) it must change its normal course in order to circumvent this quarantined cell which can never allow its state to be altered. The severity of this problem depends upon the type of sequential test implemented.

The spiral propagation procedure is best equipped to circumnavigate a quarantined cell or group of quarantined cells. When the propagating diagnoser encounters a quarantined group of cells, it simply treats it as if it were part of the guidance rail of stable control state cells which are left in the form of a ring. By following the edge of this group, it will eventually return to the guidance ring generated by test procedure and will return to its normal path of propagation. In essence, the propagating diagnoser follows a guidance rail that contains a "bump" on one of its edges which it simply follows around. This bump is actually the group of quarantined cells. Figure 5.4-1 illustrates a possible situation of this type.

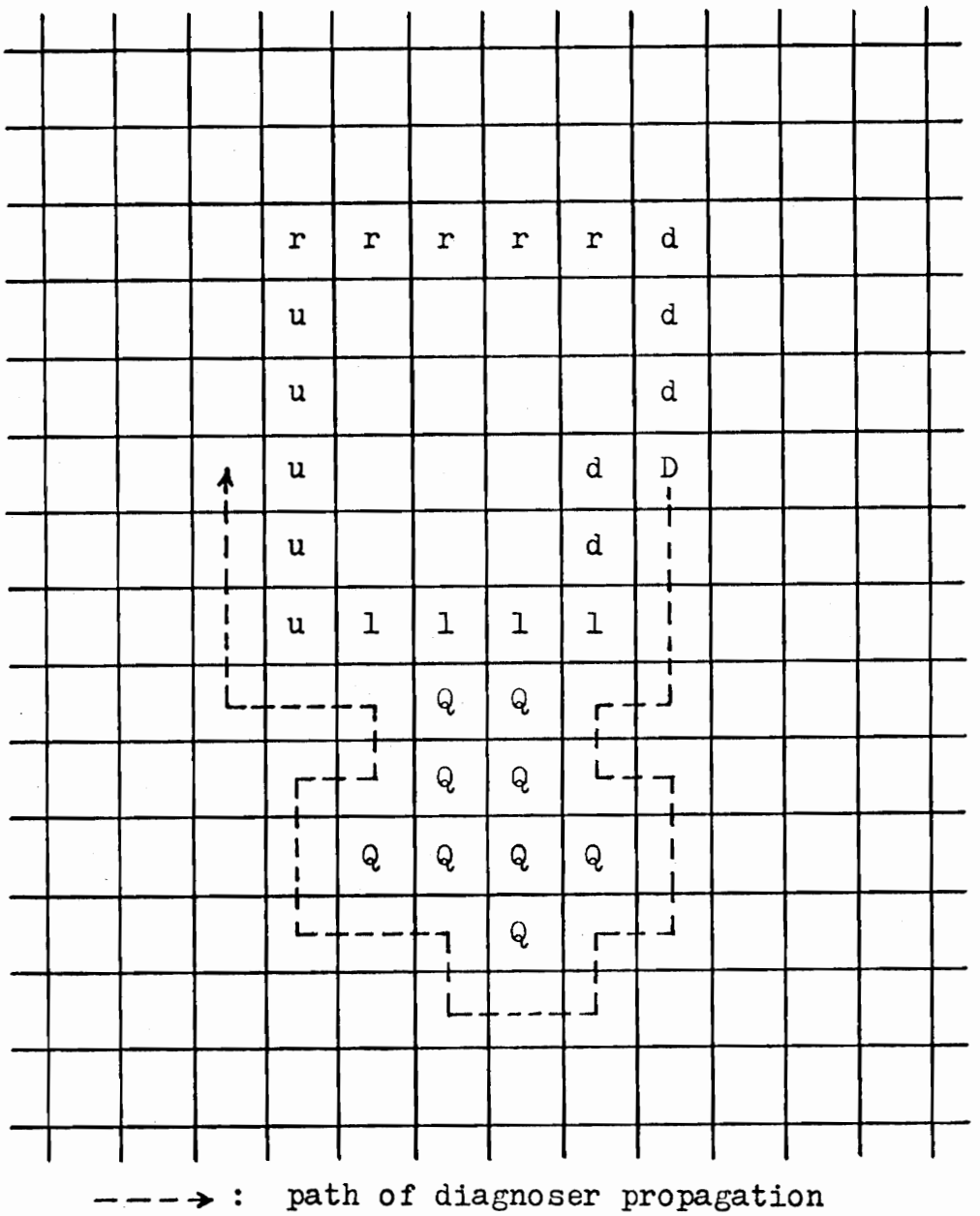


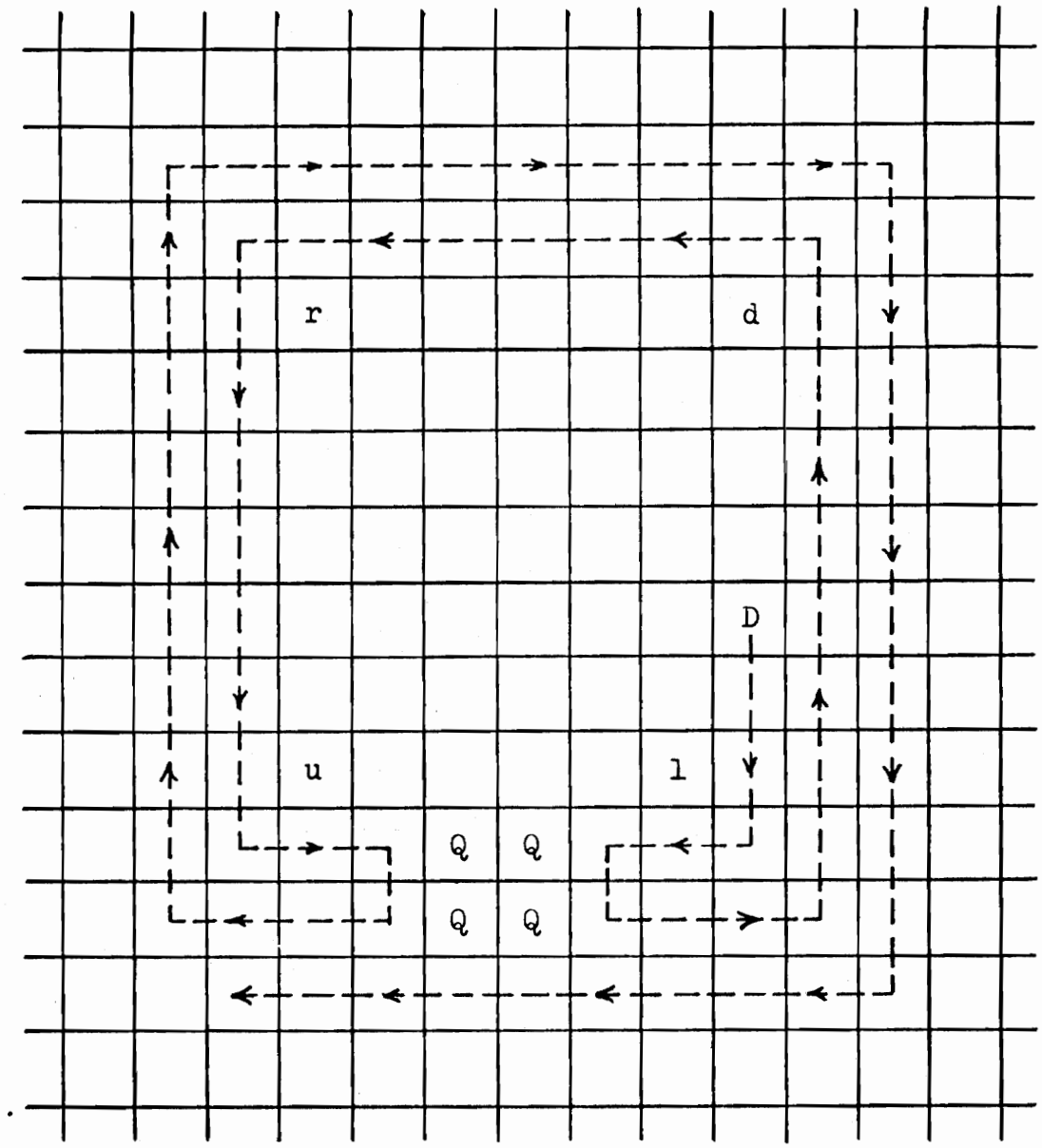
Figure 5.4-1. Illustration of a possible situation where the diagnoser of a spiral sequential test must propagate around a group of quarantined cells.

The modified spiral sequential test must be adjusted further so that when the propagating diagnoser encounters a blockage in the form of a quarantined cell, the direction of propagation is reversed. In this way, the blockage will not halt propagation and the now discontinuing spiral will continue to expand by bouncing back and forth off of the blockage until it has been circumvented. Figure 5.4-2 illustrates a possible situation involving blockage of the modified spiral.

Once the continuous ring of the concentric sequential test is broken by a blockage, it would be necessary to allow diagnosers to propagate along the edge of the blockage (as in the spiral test) until the ring was again restored. Once this is done, the ring will contain the blockage and the test can continue in its normal propagating fashion.

The technique of propagation direction reversal as suggested for the modified spiral sequential test can also be applied to the propagating diagnoser in the skew torus. When the diagnoser encounters a blockage in its path, it simply shifts to a neighboring row and reverses its direction of propagation.

Because it is possible to imagine configurations of quarantined cells that would cause problems to these suggested procedures, it may be necessary to add considerably more complex mappings to σ. Thus, by having a highly complex and possibly huge mapping σ, a sequential test can be guided around any finite blockage. Because of these added complications, this area of study represents one of the major problems involved with the propagation of sequential tests.



---→ : path of diagnoser propagation

Figure 5.4-2. A situation in which a modified spiral propagating test must bounce off of a blockage of quarantined cells.

5.5 Faults in Test Propagation

Random faults in an array can give rise to a cell which wrongly assumes the test propagating control state status. Thus, it is possible that random propagating patterns be initiated in addition to the desired propagation which is presently going on. Furthermore, a propagating test could be halted or directed off into infinity due to a fault occurring in a cell taking part in the test propagation. For these reasons, sequential testing does not appear to offer very good fault tolerant features.

However, even if a test goes astray, another test will eventually be initiated by the sequential test generator and this new propagating test will discover the problem which caused the previous test to fail. The truly vulnerable area of a sequential test is the sequential test generator. If a fault occurs in it, or if it is indirectly phagocytized, all future propagating tests could be nullified.

In the case of the skew torus, a sequential test generator could be incorporated into its array to increase the reliability of its sequential test. This generator could be used to initiate the propagating test at regular intervals in one direction and absorb any test that has propagated through the skew torus and has returned to the generator.

6.0 PARALLEL TEST SCHEDULING

6.1 Parallel Testing

Unlike the propagation of a testing pattern throughout the cellular array from cell-to-cell as in sequential testing, parallel test scheduling shows the property of oscillation between forms of simultaneous testing. For example, let the cells of a two-dimensional array be arranged in a checkerboard fashion (idential cells that can only be distinguished by the red and black colors they are assigned) for the purpose of parallel testing. It is then possible to allow all the red cells to test all the black cells simultaneously. When all the red cells have simultaneously completed this chore, it is then their turn to be tested because all cells of an array must be tested including cells that have the specific task of testing other cells. Now all the black cells can be used to test all the red cells simultaneously in the same manner as before. When this duty is complete, it will then again be the function of the red cells to test the black cells and so forth. Thus, the testing obligation oscillates back and forth between all the red cells and all the black cells.

6.2 Timing Considerations

Using a checkerboard testing approach leads to the following important question. If all the cells of one color are testing all the cells of the other color, which cells are doing useful computations? If it is assumed that the testing process requires all the functional time of this structure, then there can be no cells doing computation.

The solution to this problem is to time multiplex the system functions of testing and computation. Thus, the black cells are allowed a percentage of a red-black cycle to perform computations. At the end of this time period, the red cells are then allowed to test the black cells for the remaining (and hopefully smaller) percentage of the cycle. This process is then repeated with the roles of the red and black cells interchanged. Thus, a testing "cycle" is defined as the time period between the interchange of red and black roles.

6.3 System Preservation

Parallel testing leads to another problem concerning the preservation of the computing and control system structure. During the testing phase of a cycle, all cells of the array are either being tested or are doing the testing. It can be assumed that all of the state information is destroyed in the part of the cell being tested; therefore, it is necessary that this information be stored somewhere before the testing procedure is begun.

In sequential testing, this problem was circumvented by allowing the system to reconfigure in such a way that all the state information contained in the cell (or cells) to be tested was transferred to a quiescent cell (or cells) or to a cell vacated by another function which was being moved during this same reconfiguration. In this way, it was possible for the system integrity to be preserved in spite of the fact that the testing procedure was propagating right through the operational structure.

In a parallel testing format, however, there is no place for this state information to be reconfigured because all cells are involved in the simultaneous testing procedure. No external state information can be stored in either part of the cell performing the test because the computational part is performing the function M_t (the testing procedure), and the control part must retain its own state information in order to assign the M_t task to the computational part. Because the computational and control parts of both the cell being tested and the cell performing the test are identical, it is not possible to make room for state information storage in addition to the circuitry needed to implement the testing procedure. Therefore, it is necessary to alter the basic concept of the "fabric" so that the system structure can be preserved during the testing procedure.

6.4 Testing Hyperplane

To solve the problem of system preservation in a parallel testing environment, it is necessary to add a testing hyperplane beneath the control hyperplane of the previous fabric concept. Thus, each cell will now be composed of three parts: (1) computational, (2) control, and (3) testing. The testing part of the cell is capable of performing only the function M_t which had previously been assigned to the computational part. The control part now needs only to initiate the testing procedure by signaling the testing part to start. The control part is no longer needed to assign the function M_t to the testing part because it is capable of only that function. Thus, this frees the control part so that it can now be used to store the state information of

the control part of the cell under test. Furthermore, the computational part has been relieved of its testing duties and can now be used to store the computational state information of this same cell under test.

There are a number of advantages to this modified fabric. In fact, it is possible to pair each red cell to a black cell and group them together to form a more complex single cell. This structure lends itself to a parallel testing procedure that offers numerous advantages in its operation. This new cell architecture is best described by the term "semistructure".

6.5 Semistructure

To best explain the construction of the semistructure, it is necessary to introduce additional notation. Each cell is composed of two semicells: (1) a red semicell, and (2) a black semicell. These two semicells are identical and can be identified only by the difference in their attribute of color. Red semicells are connected to neighboring red semicells by way of ports, and this arrangement is known as the red semiarray. Black semicells are connected to neighboring black semicells with the identical pattern of ports as used in the red semiarray, and the black arrangement is known as the black semiarray. Within each single cell, the red semicell and the black semicell are interconnected via a semicell link. Each semicell is composed of three parts: (1) a computational subcell, (2) a control subcell, and (3) a testing subcell. Each type of subcell communicates with others of its type within its own semiarray by way of ports in the appropriate hyperplane as described in previous chapters. Within

each semicell, the three subcells intercommunicate via subcell links internal to this semicell. These concepts are illustrated in Figs. 6.5-1 and 6.5-2.

It is worth reemphasizing that the only difference between semiarrays is the attribute of color. The structure of the individual subcells, the individual hyperplane port interconnection pattern, and the internal subcell links is identical in every respect in both red and black semiarrays. In other words, the red and black semiarrays are exact duplicates of each other in all respects. Any system that could be implemented within the red semiarray can be identically implemented within the black semiarray.

6.6 The Parallel Testing Procedure

The parallel testing procedure to be described was investigated because of the multitude of advantages that can be derived from its use. This procedure will operate only in an array of cells which internally contain the semistructure previously mentioned. For the sake of simplicity and comprehension, the steps of the procedure are listed in only the most general of terms. Furthermore, a set of restrictive assumptions are made so that the essence of the procedure can be initially described without the obscuring effects of optional complications. Table 6.6-1 states the assumptions, and Table 6.6-2 tabulates the parallel testing procedure.

A complete testing phase is defined as the time period of the parallel testing procedure during which a testing subcell conducts the diagnostic procedure on its semicell counterpart. Thus, steps 1

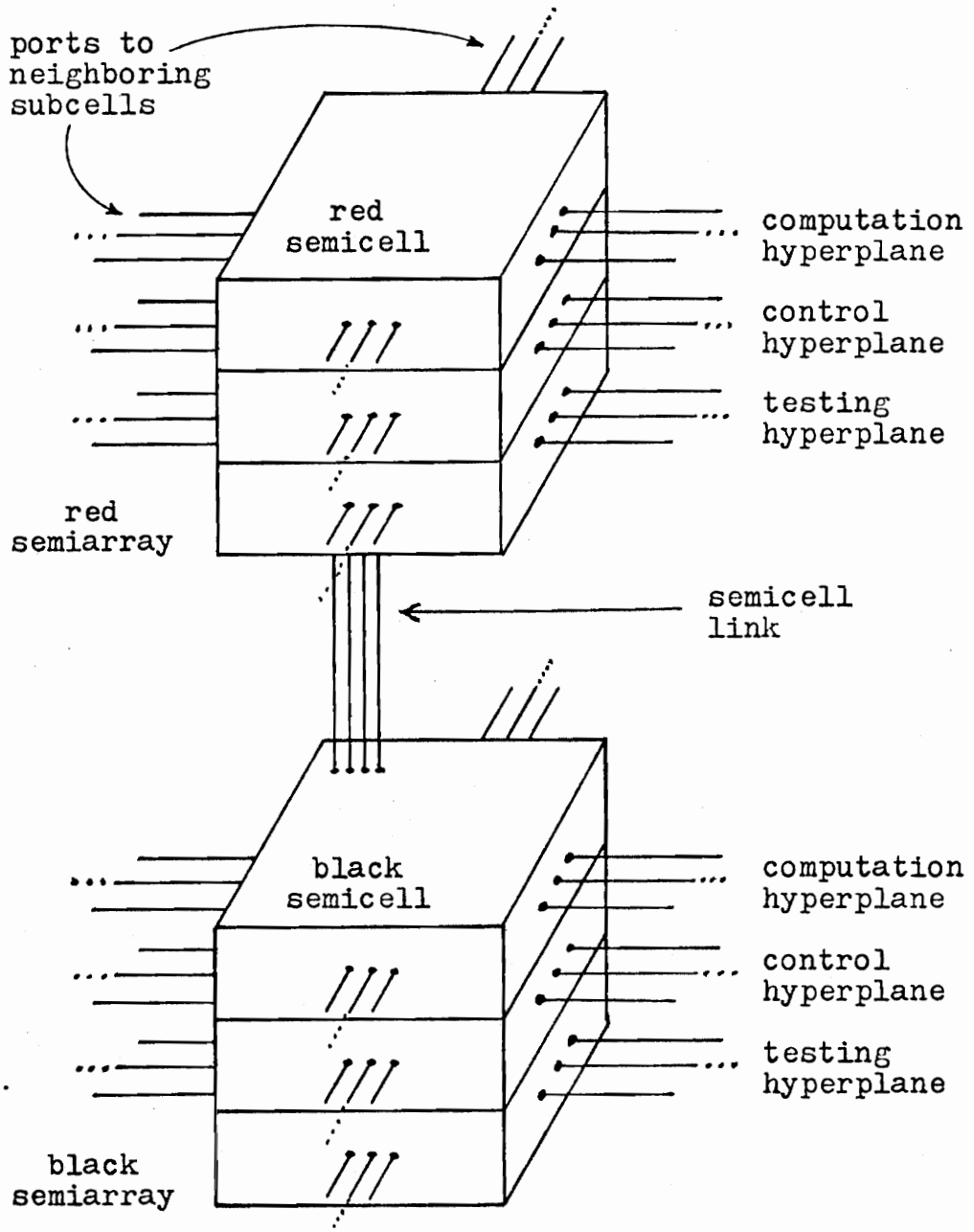


Figure 6.5-1. Structure of a single cell showing the semicell link between each semicell of the red semiarray and the black semiarray.

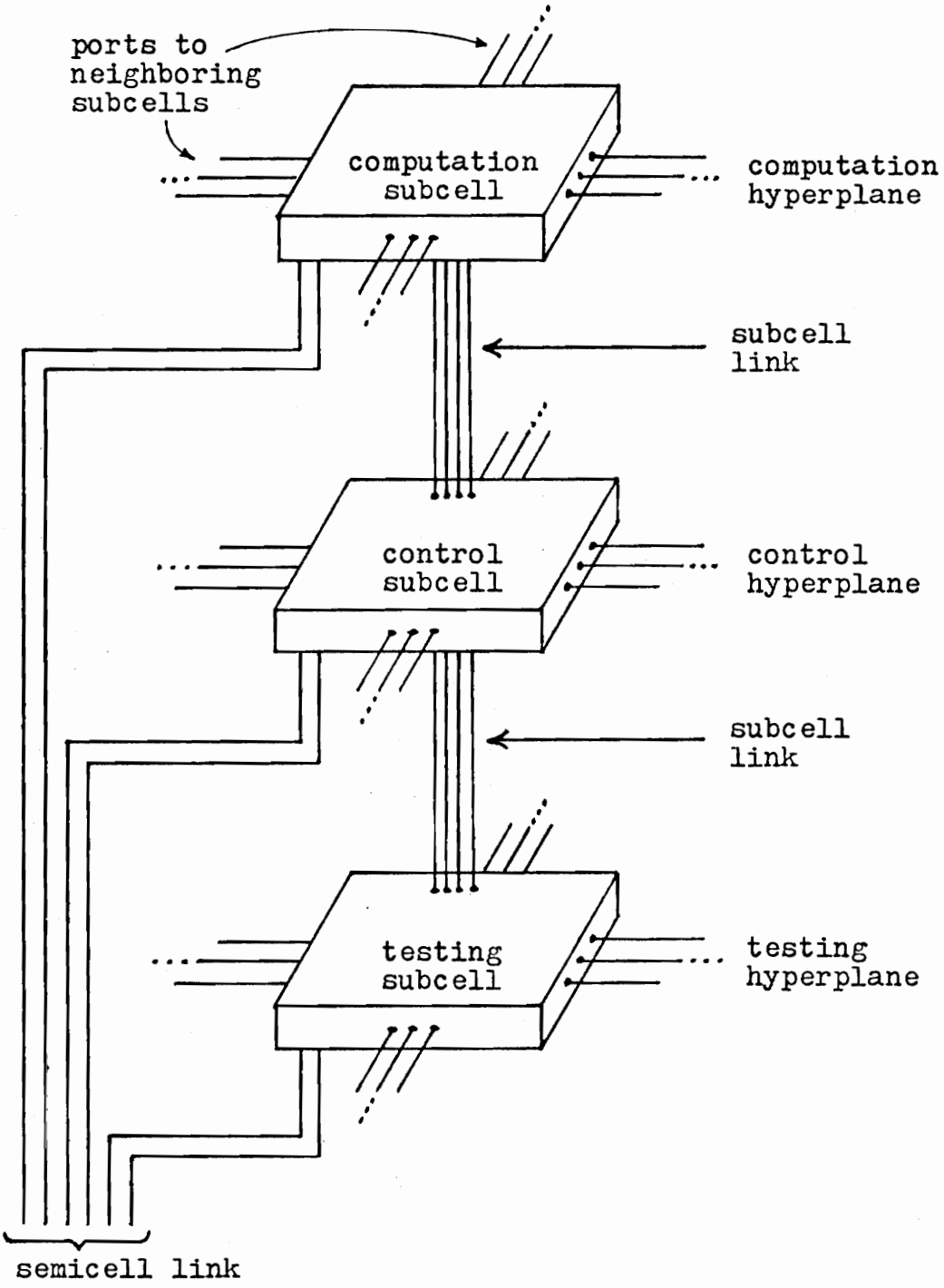


Figure 6.5-2. Subcell links between the three subcells within each semicell of a given semiarray.

through 15 of the Parallel Testing Procedure of Table 6.6-2 comprise a complete testing phase. Step 16 of this same procedure also comprises a complete testing phase. These two complete testing phases, that is, steps 1 through 16, make up a testing cycle.

Table 6.6-1

Assumptions of the Parallel Testing Procedure

1) This synchronous procedure is performed simultaneously in all red semicells of the red semiarray; and is also performed simultaneously in all black semicells of the black semiarray.

2) Each computational subcell, control subcell, and testing subcell is identical not only to its counterpart in all other subcells, but also to each other in the same semicell.

3) Each subcell is capable of breaking and reinstating all intercommunication links (ports) from its neighbors.

4) Each subcell is capable of breaking and reinstating all subcell links to other subcells.

5) All of the state information contained in the computation subcell of one semicell can be transmitted to the computation subcell of the other semicell via the semicell link.

6) All of the state information contained in the control subcell of one semicell can be transmitted to the control subcell of the other semicell via the semicell link.

7) The test performed by the testing subcell can detect any permanent single or multiple fault within each subcell of the alternate semicell during the subcell testing phase.

8) The test performed by the testing subcell is structured so that a faulty subcell does not appear fault free due to the failure of a communication link to be broken.

9) The tests performed by the testing subcell via the semicell links can diagnose any permanent single or multiple fault within the semicell link during some phase of the testing cycle.

10) Faults do not occur in both the red semicell and the black semicell of the same cell during a single testing cycle.

Table 6.6-2

The Parallel Testing Procedure

1) The black testing subcell signals the red testing subcell, via the semicell link, that a complete testing phase has just been completed. Thus, the testing procedure is now initiated within the red testing subcell.

2) The initiation of the testing procedure in the red testing subcell is reinforced by signals coming from similarly initialized red testing subcell neighbors. Moreover, if the initiation signal from the black testing subcell does not arrive, an initiation signal from a majority of red testing subcell neighbors is sufficient to initiate the testing procedure in the red testing subcell.

3) The red testing subcell signals via the semicell link that the black computation and control subcells are to halt their functions.

4) The red testing subcell signals the black computation subcell to shift all of its state information into the red computation subcell over the semicell link, and it signals the black control subcell to shift all of its state information into the red control subcell over the semicell link. This signaling is done via the semicell link and the black subcell links.

5) The red testing subcell signals, via the red subcell links, that the red computation and control subcells are to resume the functions transferred in the form of state information from their black counterparts.

6) The red testing subcell signals all three black subcells, via the semicell link and the red subcell links, to break all intercommunication links (ports) with their neighbors.

7) The red testing subcell signals all three black subcells, via the semicell link and the red subcell links, to break all subcell links between each other.

8) The red testing subcell applies a diagnostic test to all three black subcells simultaneously via the semicell link and the red subcell links.

Table 6.6-2 (Continued)

The Parallel Testing Procedure

9) If a fault (or faults) is discovered, the red testing subcell signals this discovery to the red control subcell so that reconfiguration can be initiated; moreover, the continuation of the testing procedure is unnecessary. If no fault is found, the procedure continues.

10) The red testing subcell signals all three black subcells, via the semicell link and the red subcell links, to reinstate all of their subcell links.

11) Now that the individual black subcells have been found internally fault free, they are assigned by the red testing subcell, via the semicell link and the red subcell links, the function of applying a diagnostic test to their own subcell links.

12) If a fault (or faults) is discovered, the red testing subcell is signaled, via the semicell link and the red subcell links, by the black subcell that discovered the fault. The red testing subcell signals this discovery to the red control subcell so that reconfiguration can be initiated; moreover, the continuation of the testing procedure is unnecessary. If no fault is found, the procedure continues.

13) The red testing subcell signals all three black subcells, via the semicell link and red subcell links, to reinstate all of their intercommunication links (ports) to their neighbors.

14) Now that the black semicell has been found internally fault free, it is assigned by the red testing subcell, via the semicell link and the red subcell links, the function of applying a diagnostic test to the ports between the black semicell and its neighbors in the black semiarray.

15) If a fault (or faults) is discovered, the black semicell signals this to the red testing subcell via the semicell link. The red testing subcell signals this discovery to the red control subcell so that reconfiguration can be initiated; moreover, the continuation of the testing procedure is unnecessary. If no fault is found, the procedure continues.

Table 6.6-2 (Continued)

The Parallel Testing Procedure

16) Repeat steps 1 through 15 with the words "red" and "black" interchanged.

17) Repeat steps 1 through 17.

6.7 Advantages of the Parallel Testing Procedure

The Parallel Testing Procedure listed in Table 6.6-2 offers many advantages over sequential testing schemes. One of the primary advantages of this test is the fact that almost no computation time is lost due to the testing procedure. In fact, the only computation time lost is the time needed to shift the state information in the computation and control subcells of one semicell into the respective subcells of the other semicell during the beginning of each complete test phase. At all other times when the testing procedure is being performed, the computation and control subcells and hyperplane ports of the semiarray conducting the test are busily continuing the computational tasks of the system. Thus, there is practically no computational "down-time" due to this testing procedure.

The testing procedure is essentially transparent to the operational perspective of the overall system array. Provided no faults have occurred in the array, the effective control state of each cell remains unchanged. Thus, global equilibrium of a final configuration is maintained. When a fault is found, reconfiguration is triggered in the control subcell of the semicell conducting the test. This reconfiguration now becomes a function of the control hyperplane and proceeds independently from the parallel testing procedure that is still continuously taking place.

Because the testing procedure is independent of the control functions of the array, no complex mappings in σ are needed to allow testing to proceed. No constraints are made on the neighborhood index by

the testing procedure because there is no need for the computational configuration to reconfigure around a propagating testing pattern. Because a testing pattern does not propagate, it is not necessary to devote complex mappings and control states to the testing procedure so that it can propagate and move around quarantined cells without failing to test all other testable cells within the array.

A fault may occur in the testing subcell which is conducting the testing procedure such that it fails to initiate the testing procedure in its counterpart testing subcell when the complete testing phase is over. Even so, the neighbors of this counterpart will initiate the testing procedure in it. The counterpart testing subcell will then proceed to discover the fault in its opposite subcell during its testing phase.

Although this procedure will operate in any array geometry, it is well suited for use in an infinite or quasi-infinite array of arbitrary dimensionality. Therefore, no special array geometries need be constructed for the proper operation of this testing procedure.

6.8 Quarantined Cells

Steps 9, 12 and 15 of the Parallel Testing Procedure all initiate reconfiguration in the presence of a faulty cell. If the fault was discovered to be not in the control subcell, the portion of the semicell link used to shift control state information, nor in any of the control hyperplane ports to its neighbors, it is necessary only to quarantine this one faulty cell. Thus, the control subcells of both semicells must assume the quarantine state output and maintain this state

information. If the nature of the fault is such that these areas of cell structure that could prevent both control subcells from assuming the quarantined control state contain the fault, it will be necessary to phagocytize the faulty cell. Thus, reconfiguration, so that all the neighbors of the faulty cell assume the quarantined control state before the completion of the complete testing phase, makes it unnecessary to attempt assigning any particular control state to either control subcell within the faulty cell.

6.9 Modification of the Parallel Testing Procedure

The purpose of the Parallel Testing Procedure is to allow one semicell to cause the other semicell's subcells, subcell links, ports to neighbors, and the semicell link to be tested. Any modification of the steps of Table 6.6-2 that allows this purpose to be fulfilled is acceptable. Furthermore, the purpose of breaking intercommunication links (ports) and subcell links is that signals from one subcell under test will not effect the outcome of another subcell under test. Thus, it is necessary to break only such links that will interfere with the testing procedure of the individual subcells. For example, it is possible that step 8 of the Parallel Testing Procedure be modified so that the subcell diagnostic can be applied individually to the subcells of the semicell under test as opposed to the suggested simultaneous testing of all three subcells.

The single assumption that places the greatest restrictions upon the use of this procedure is that all subcells are identical. It is possible to allow the subcells to contain different structure provided

the testing subcell is still capable of individually testing all three types of subcells. In fact, it is possible to modify the testing procedure so that the testing subcell first tests the least complex of the three different subcells. Once it has been found fault free, it can be assigned the function of helping the testing subcell to diagnose the next more complex of the subcells to be tested. This newly tested subcell can now also be pressed into the service of aiding in the testing of the most complex of the three subcells. Under this modification, it is not even necessary that the testing subcell be the most complex of the three possible subcells because by the time it is needed to test a subcell more complex than itself, it has the enlisted help of two other subcells that as a whole is more complex than this most convoluted of single subcells.

Of course it is possible to design a subcell that is much more complex than the sum of the testing abilities of the testing subcell plus those of the two lesser subcells already found fault free. In this case, it is necessary to make modifications in the Parallel Testing Procedure that would allow neighboring semicells to become part of the testing procedure of a single subcell. Another possibility is to partition this highly complex cell to be tested into smaller submachines that can then be tested by the coalition of the testing subcell and the already tested subcells of semicell under test.

7.0 FAULT RESISTANT TECHNIQUES

7.1 Critical Faults

A fundamental problem exists in the design of any fault tolerant computational system. If this system involves a finite structure, even though it may be implemented within an infinite fabric, there is always a finite probability of a multiple fault occurring that will cause the system to fail catastrophically. For example, there is a finite possibility that every nonquiescent cell of any array configuration will acquire the quarantined state simultaneously due to a set of individual but simultaneous faults. This would cause total failure of that computing configuration independent of the testing procedures that were implemented within this array.

There also exists a set of critical faults that are much more likely to occur and are still capable of producing potentially incurable injury to the system. In a sequential test scheme on a quasi-infinite array, a simultaneous fault in all of the cells which constitute the sequential test generator could cause the cessation of any further propagating testing patterns being generated. This would invalidate the self-repairing properties of a sequential test scheme. For the parallel testing approach, it is possible that simultaneous faults occur in both testing subcells of a faulty cell in such a way that a reconfiguration signal is never generated. Thus, this faulty cell might continue to appear fault-free to its neighbors and cause critical errors to be accepted by the computational system. If is possible, however, to help reduce the criticality of this type of failure by combining

testing schemes, specifying fault tolerant architecture in a chosen configuration, and by using fault detecting software within the finished system.

7.2 Combined Testing Schemes

It is possible to combine both parallel and sequential testing formats to gain the advantages of both types. With this combination, the parallel testing approach can best bear the burden of detecting all faults that do not occur within both semicells of a single cell during a single testing cycle. A sequential test generator can be implemented on this parallel testing array as part of the initial configuration. Its task would then be to occasionally initiate a sequential test to propagate throughout this quasi-infinite array and search for cells containing faults in both of their semicells which had incapacitated the parallel testing procedure within this cell. This combination would detect all types of faults except for the specific case where both semicells of all cells of the sequential test generator have simultaneously occurring faults. Even this particular case can be avoided by the use of redundant sequential test generators.

7.3 Redundant Architecture

Any of a number of classic fault tolerant architectures can be implemented in the form of configurations on the self-repairing array. The classical approach usually involves a number of redundant subsystems with outputs convergent upon some type of majority decision element from which the final output is derived. In this way, the same

output that is presented by a majority of the redundant subsystems is chosen as the final output because it is assumed to be error free. Outputs that differed from this majority decision output are assumed to contain errors due to a fault in their respective subsystems. The flaw in the concept of this system being fault tolerant is that a single fault in the nonredundant majority decision element can invalidate all advantages of having any system redundancy. However, when implemented in a self-repairing array of the type developed in previous chapters, this classical architecture is free of its previously fundamental flaw. Any fault occurring in the nonredundant majority decision element will simply be quarantined, and the configuration will reconfigure so as to restore the functional operation of this nonredundant majority decision element.

The advantage of having redundant subsystems is that important computational state information will be duplicated. As was previously pointed out in Chapter 4, it is possible to loose all of the current computational state information of a cell in which a fault occurs. If this information is critical to a particular calculation taking place, errors will result. This problem can be largely defeated by this duplication of computational state information.

For example, an architecture is chosen that allows triplication of a computer configuration that is to perform a set of calculations and of the program memory associated with each computer subsystem. If all three of these subsystems perform the same calculation algorithm, computational errors due to computational state information loss following the occurrence of a fault will appear in the subsystem containing

the fault. However, the other two subsystems will have finished the calculation without error. The output results of all three subsystems are then forwarded to a nonredundant majority decision element that would then pass only the error free result. The number of duplicate subsystems are dictated as a function of the probability of faults occurring in more than one subsystem per calculation algorithm. Thus, the more often the occurrence of faults, the more the duplication of subsystems will be necessary to ensure reliable calculations.

It is also possible to increase the reliability of sequential testing schemes by providing redundancy of sequential test generators. For example, a sequential test generator configuration could be triplicated, and these three generators could then be grouped near the computational configuration. If the timing sequences are selected so that the tests are generated in a staggered sequence, all three generators can individually initiate propagating sequential tests that do not interfere with each other. Figure 7.3-1 gives the conceptualization of the combination of both examples in this section. With reference to Fig. 7.3-1, the "top" sequential test generator is the first to initiate a propagating sequential test (as shown) that essentially expands in a concentric fashion. Once this concentric group of cells involved in the test propagation has grown so that all of the nonquiescent configurations have been tested and are therefore included within the concentric ring, the "middle" sequential test generator will initiate another propagating sequential test. This staggered test initiation is then continued by the "bottom" sequential test generator, then back to the "top"

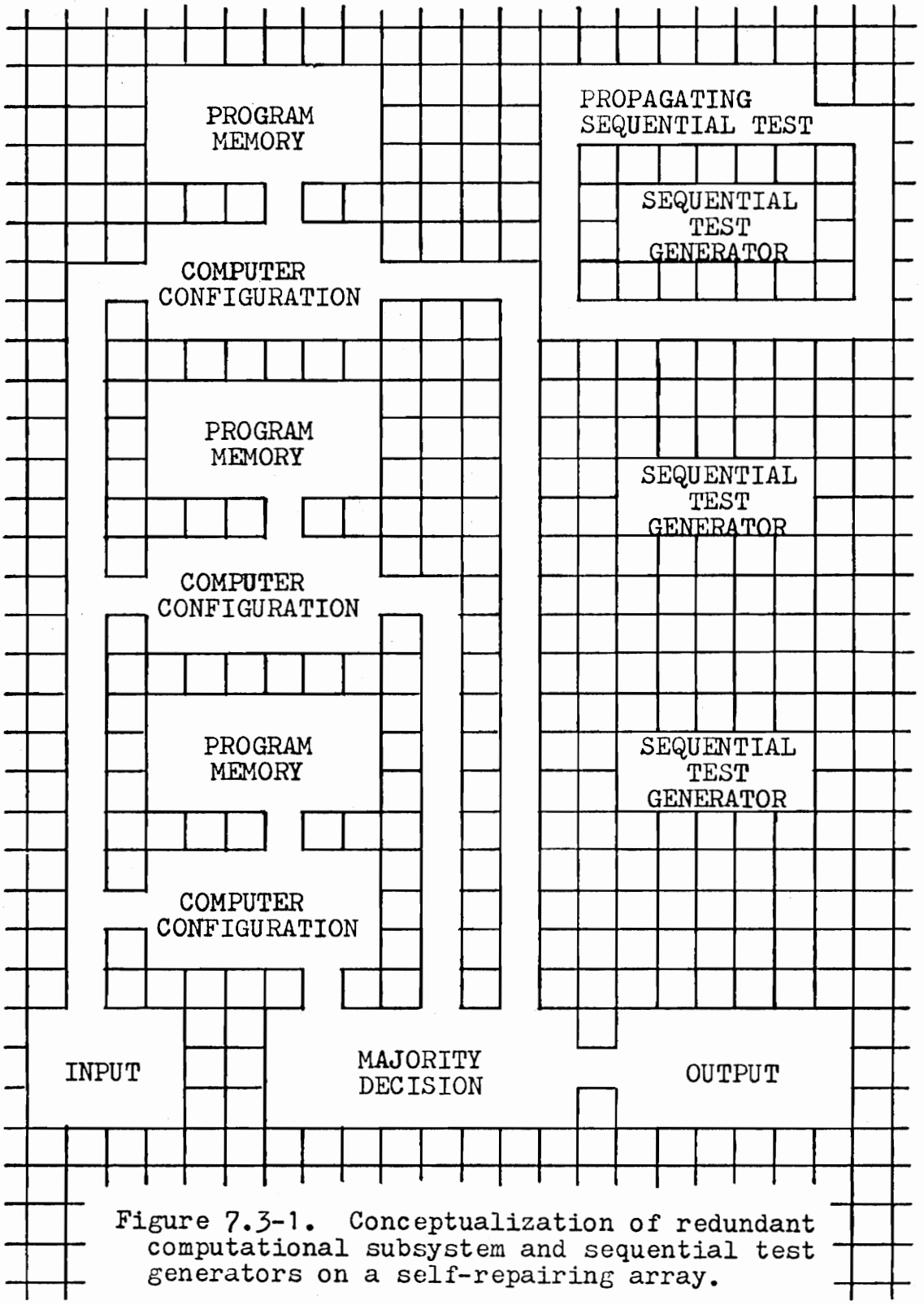


Figure 7.3-1. Conceptualization of redundant computational subsystem and sequential test generators on a self-repairing array.

generator and so forth. In this way, if the rate of concentric expansion is relatively constant between two propagating testing patterns, they will never collide so as to interfere with each other.

If a classical architecture has been chosen that incorporates a nonredundant majority decision element, computational state information could be lost due to a fault in this element. This does not cause problems because the nonredundant majority decision element need not retain any computational state information during the performance of its function. Thus, if a fault were to occur in the nonredundant majority decision element, an error at its output would disappear as soon as reconfiguration had occurred because of the quasi-combinatorial logic property of this function.

7.4 Redundant Software

The system of redundant architecture described in the previous section can be modified so that redundant computer configurations are unnecessary. Redundant software, or the repetition of blocks of calculation algorithms, can be used in a single computer configuration as long as duplicate storage is provided for intermediate results. In this way, computational state information is accessed from one area of storage and is processed by the calculation algorithm and then is replaced in storage by the result of that algorithm. This same computational state information is then accessed from a different area of cuplicate storage and is processed and replaced as above. This is repeated for as many areas of duplicate storage that are provided. When this is done, a nonredundant majority decision element that need not

retain any computational state information can simultaneously examine the duplicate results and produce a single output. Thus, if an error occurs due to a fault in either a redundant storage area or in the computer configuration during a single run of the calculation algorithm, only the faulty storage area or the result produced by the run with the error due to a fault will contain any possible errors. The other duplicate results will be error free due to the self-repaired status of the computer following reconfiguration around the fault before the next calculation algorithm is repeated.

8.0 CONCLUSIONS

8.1 Conclusions

As a result of this research, the author has satisfied himself that it is possible to conduct test procedures in an array which exists in a fault tolerant environment. The following section contains recommendations as to what procedures should be used based on the opinions of the author.

8.2 Recommendations

It is recommended that cells which offer single port testability be used in the array due to the variety of test configurations that can be used to diagnose them. Furthermore, single diagnoser test configurations should be used whenever possible because of the high efficiency of implementation when compared to other testing configurations. In the realm of sequential testing, the modified spiral approach appears to offer the best compromise among all of the problems which plague the sequential form of testing.

In the question of sequential versus parallel testing, it is better to avoid sequential testing altogether. Because parallel testing is transparent to the system TA, and because it seems not to suffer from any critical complications, this choice is vastly superior over the muddled implementation and associated problems of a sequential test. It is possible, however, to combine both parallel and sequential testing schemes in an effort to afford advantages of both types. Also, various degrees of redundant "macrostructure" can be incorporated into

the desired computer configuration in an effort to improve overall system reliability.

8.3 Future Research

The next logical step in this line of research is to try to implement some of these abstract testing concepts in an array of cells with some physical internal structure. The use of Universal Modular Trees (UMT's) [9,10] should be investigated for this purpose.

It has not escaped the author's notice that tessellation automata theory coupled with some testing procedure might be used as a general model for the existence and functioning of any multicelled biological organism. The similarities between a TA array and a living organism, for example, a human being, are many. As a case in point, consider how the human animal is grown from a single initial configuration "seed" cell. The mapping σ that allows this development to occur is stored chemically in the form of DNA. In fact, every cell of the body carries the same genetic information in its own DNA which is complete enough to develop an exact whole duplicate of that individual. The fact that cloning can eventually produce such an exact mature copy of a living organism offers proof of how uniform the rules of this living TA are implemented. Once the growing configuration reaches a final configuration (maturity), the only perturbations which upset global stability (with the exception of the testing and desired propagation functions) are caused by faulty cells. White blood cells (leucocytes) can be considered a form of sequential test which propagates throughout the system and phagocytizes any faulty cell it detects. The energy source

of the system is highly distributed and can withstand intermittent operation. The control state information is communicated between neighboring cells chemically via enzymes and other processes.

Comparisons of this type can be made on almost an indefinite number of biological topics. There is ample room in this area of research to produce highly significant results pertaining to the modeling of general biological systems through these abstract technical means.

BIBLIOGRAPHY

1. R. A. Thompson, S. M. Walters, and F. G. Gray, "Stability in a Class of Tessellation Automata," Southeastern Symposium on System Theory, April 1977.

2. J. Von Neumann, Theory of Self-Reproducing Automata, (edited and completed by A. W. Burks), University of Illinois Press, 1966.

3. F. C. Hennie, "Fault Detecting Experiments for Sequential Circuits," Proc. Fifth Annual Symposium on Switching Circuit Theory and Logical Design, pp. 95-110, Princeton, N. J., November 1964.

4. J. H. Holland, Essays in Cellular Automata, (edited by A. W. Burks), University of Illinois Press, 1970.

5. S. F. Unger, "Pattern Recognition Using Two-Dimensional Bilateral Iterative Combinational Switching Circuits," Proceedings of Symposium on Mathematical Theory of Automata, 1962.

6. H. Yamada and S. Amoroso, "Tessellation Automata," Information and Control, Vol. 14, pp. 299-317, 1969.

7. H. Yamada and S. Amoroso, "A Completeness Problem for Pattern Generation in Tessellation Automata," J. Comput. Sys. Sci., Vol. 4, pp. 137-176, 1970.

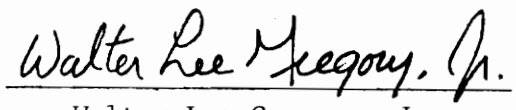
8. H. Yamada and S. Amoroso, "Structural and Behavioral Equivalences of Tessellation Automata," Information and Control, Vol. 18, pp. 1-31, 1971.

9. F. G. Gray and B. A. Prasad, "Fault Diagnosis in Uniform Modular Realizations of Sequential Machines," International Symposium on Fault Tolerant Computing, Palo Alto, California, June 20-22, 1973.

10. F. G. Gray and B. A. Prasad, "Research Initiation: A. Theoretical Investigation of Diagnosable Logic Systems," Final Report, December 1973, NSF Grant GJ-32718, NTIS Accession No. PB 230444/AS.

VITA

Walter Lee Gregory, Jr. was born September 5, 1953, in Newport News, Virginia. He attended Virginia Polytechnic Institute and State University and received a BSEE degree in June, 1976. During the next year, he worked on self-repairing automata funded by an NSF grant. He expects to receive his MSEE degree in May 1977. He has had previous engineering work experience with RCA Corporation and will be employed, after graduation, by Hewlett-Packard Company in Loveland, Colorado.



Walter Lee Gregory, Jr.

TEST SCHEDULING AND CONFIGURATION IN A SELF-REPAIRING COMPUTER

By

Walter Lee Gregory, Jr.

(ABSTRACT)

The structure of cellular arrays and Tessellation Automata are introduced as a "fabric" for use in the implementation of self-testing and self-repairing computational configurations. Testing schemes are suggested for use in the "self-testing" operation of this computer system. Sequentially propagating tests are examined for both finite and infinite geometries of cellular arrays. A static parallel testing procedure is also suggested which offers these advantages: (1) the parallel testing procedure is transparent to the computer configuration, (2) very little computational down-time" results from testing, and (3) the parallel testing procedure does not initiate reconfiguration of the computational configuration.