AN INTEGRATED APPROACH TO THE DESIGN OF FAULT TOLERANT COMPUTING SYSTEMS

L. Philip Caillouet, Jr.
Bruce D. Shriver, Sr.

DAIMI PB-32
July 1974

AN INTEGRATED APPROACH TO THE DESIGN OF FAULT TOLERANT COMPUTING SYSTEMS.

by

L. Philip Caillouet, Jr.

Bruce D. Shriver, Sr.

Computer Science Department

University of Southwestern Louisiana.

Summary

This paper offers an introduction to a research effort in fault tolerant computer architecture which has been organized at the University of Southwestern Louisiana (USL). It is intended as an overview of several topics which have been isolated for study, and as an indication of preliminary undertakings with regard to one particular topic. This first area of concentration involves the systematic design of fault tolerant computing systems via a multi-level approach. Efforts are being initiated also in the areas of diagnosis of microprogrammable processors via firmware, fault data management across levels of virtual machines, development of a methodology for realizing a firmware hardcore on a variety of hosts, and delineation of a minimal set of resources for the design of a practical host for a multi-level fault tolerant computing system. The research is being conducted under the auspices of Project Beta at USL (26).

* This research is supported in part by NATO Grant 755 and is being conducted at the University of Southwestern Louisiana and at the University of Aarhus.

Fault tolerant computing.

The design of an electronic computing system is necessarily constrained by the requirements of the particular class of problems it is dedicated to solve. Such constraints are imposed in the form of performance parameters (add times, memory access rates, storage addressability, etc.) which must be directly or analytically measurable. A range of acceptable values must be extablished for each parameter during the design process, which is dependent upon available technologies and the special requirements of solutions to the target problems.

A class of problems for which system reliability may be the most critical parameter of all is the class which includes onboard and external support for space flight, military command and control, signal transmission and analysis, air traffic control, assembly line and process control, and rapid transit applications. All of the above have in common the monitoring of irreproducible (or expensively reproducible) inputs, the real time determination of reaction strategies, and the initiation of irreversible (or expensively reversible) processes. Each exhibits an extraordinary concern for the success of its mission owing to the fact that human lives may be jeopardized by mission failure.

Few computer system designers would be foolhardy enough to suggest that they could predict reliability of any given logic gate (no matter the technology) within a specific lifetime. Statistical estimates are often made upon experience with thousands of components, but no degree of confidence in such an estimate can of itself preclude the possibility of a single, freak, and possibly very costly failure. Consequently the designer must arrange available hardware, firmware, and software components in a fashion as to minimize the failure probability of the system while justifying the cost-effectiveness of his schema. In this light, we may accept a qualitative definition of a fault tolerant computing system as a computing system ".... organized and structured such that it can perform its design specified functions even in the presence of hardware failure" (18). Some terms related to fault tolerant computing include:

Mission Time	-- an upper bound to the duration of the required reliable operation of a computer system.
MTBF	-- mean time between failure of a component of the system or of the system itself.
Fail-safe	-- a design requirement that the system remain in continuous and completely functional operation.
Fail-soft	-- a relaxation of the fail-safe design requirement which concedes some degradation of functioning, but retains the requirement for continuous operation.

Diagnosis	-- determination of the fact that a failure has occurred and of the location of the failure.
Recovery	-- reaction of the computing system to the report of a diagnosed failure.
Hardcore	-- "that portion of the system that must function correctly to initiate and perform fault testing procedures" (24).

Thus in quantitative terms, we may define a fault tolerant computing system as a computing system organized and structured around a hardcore which maximizes the MTBF of the system to such an extent as to ensure, within specific limits of confidence, the fail-safe (or fail-soft) integrity of the system for the duration of its mission.

Much effort has been expended in the development of reliability models for systems of identical components (7). Indeed, component redundancy seems to be the single most intriguing concept to those who have published in the area of fault tolerant computing (18, 25). Reliability modeling generally begins with consideration of a system of components which requires K out of N identical components to be functional in order that the system survive. Arrangement of the components in parallel and series subsystems leads to development of rules for the prediction of system reliability given the reliability of an individual component.

Models exist (18, 7) for systems using voted outputs as well, in the form of n-modular-redundant (nmr) systems, where the outputs of n components are compared and the majority ((n+1)/2) output is chosen as that of the system. Triple-redundancy is a popular special case of nmr for n=3. Detection of disagreement among the voters in nmr systems may or may not call into play some reconfiguration mechanism to eliminate the minority components and replace them with others from a pool of spares. Such "dynamic" redundant techniques (18, 25, 3, 6) are known as <u>standby sparing</u>, and contrast with "static" techniques (18, 25) which derive the majority output while retaining the faulty component, thus <u>masking</u> the fault. Fault masking can be effective against transient failures, such as may be caused by local flutuations. These failures are generally irreproducible and may not affect continued reliability if a voting scheme is being employed, whereas logic failures suggest that the offending component need be removed from active status for repair. Hybrid redundant techniques (18) apply both static and dynamic philosophies to realize nmr with sparing.

In the context of fault tolerant computing systems, committent of the system hardcore completely to <u>hardware</u> often limits the utility of the system with regard to the solution of problems not in the original class, e.g. variation in the degree or extent of diagnosis required by a given application, or in the mode of recovery available to the system. Such limitations may be felt in attempts to recover from multiple adjacent logic failures where the hardware has been designed to detect only single failures; in this case, there may be certain critical applications for which it may not be possible to use the given computing system. There is, however, the tendency today to design fault tolerant computing systems from bottom upward (18, 25, 3), a strategy which is known to restrict even the primary applicability of a system by "casting in concrete" basic features of the system. Further, witness the difficulty experienced by systems programmers in attempting to devise diagnostic routines and strategies for a computing system _after_ its design had been finalized (10). Design overhead and post-design complications may be considerable when rethinking and retooling for each system in a given production series is required.

Current research efforts.

The work which is being initiated at USL is concerned with the following areas:

(1) development of a systematic approach to the design of fault tolerant computing systems,

(2) comparative analysis and refinement of techniques for diagnosis of dynamically microprogrammable processors with fail-safe or fail-soft requirements,

(3) realization of firmware diagnostic routines initiatable by virtual machine emulators (in real time) on a given host,

(4) delineation of architectural constraints for microprogrammable fault tolerant computers.

Heretofore, research efforts have been reported in hardware implementations of fault tolerant techniques and software operating system implementations (18, 25, 14, 17, 29, 11). Both approaches restrict the flexibility of interpretation of the host architecture. The former imposes restrictions at fabrication time and the latter at system generation (sysgen) time. Microprogrammed implementations of both diagnosis and recovery strategies offer delayed binding of the incorporation of fault tolerant capabilities into a computing system, thus adding flexibility to the control of host hardware, while functioning with relative independence of higher-level software. It should be obvious that this approach should offer an order of magnitude enhancement in operating speed beyond that provided by software schemes -- a valuable bonus indeed for real time systems.

Manufacturer and user experience with servicibility and maintenance aspects of system design provides a wealth of insight into the specific problem areas of the diagnosis of a functioning system. Examples of this experience are found in the literature (11). We will now expand upon the research which is currently underway.

Development of a systematic approach to the design of fault tolerant computing systems.

Professor Wilkes' 1951 article (30) had as its expressed purpose the establishment of a systematic approach for the design of the control organ of a digital computer. The technique of microprogramming has since found widespread applicability not only in the design of new commercial systems, but also in the realization of virtual machines -- emulated machines -- on microprogrammable processors committed to architectural research. Firmware implementations of fault tolerant designs have not been reported in the literature, however, with the exception of microprogrammed diagnostic routines in certain systems (11, 5, 13, 16) and isolated instances of firmware beckup of hardware functional units (5).

Indeed diagnostic processing has historically been considered inherently "different" from target process execution. Examples of this situation may be drawn from everyday experience with most any third generation computer: diagnostic programs ("diagnostics") are run typically in a privileged mode by a system engineer, and are run most often only during maintenance seesions. Certain systems (5, 16) employ diagnostic microprograms ("microdiagnostics") in concert with diagnostics, but the scheduling of these is also privileged. Examples of failure diagnosis and timely system recovery are not frequently found in the marketplace as firmware implementations; exceptions to this statement are noted in the most recent systems, particularly as regards the argumentation of channel/device-controller autonomy as exemplified in the IBM Systems 370(11).

The research outlined here is concerned with the institution of a systematic design schema for multi-level computing systems with requirements for high reliability and high availability. Our approach will attempt to extend that of (20, 28, 4) to multi-level systems.

By "multi-level" here is meant that there exists a collection of virtual (emulated) machines related in a tree structure. The root node of the structure corresponds to the physical host hardware of the computing system as in Figure 1 and Figure 2. The order or rank of a node in the structure indicates the <u>level</u> of a particular virtual machine; an <u>index</u> may be associated with each virtual machine on a given level to facilitate unique identification of the node it occupies. The pair [level, index] specifies a virtual machine in a multi-level system. Acronyms can be introduced into the [level, index] pair and are to be equated on a one-to-one basis with the <u>index</u> portion of the pair. This identification schema is similar to that found in (12). The <u>level</u> information is retained explicitly in Figure 1 and Figure 2, which are examples of multi-level systems.

A virtual machine identified by the pair [o, i], where i is an arbitrary integer index, is a physical machine, specifically one of the host computers available to the system. In cases where i is identically zero for level zero, only one host is in the system; this is the situation depicted in Figure 1. Figure 2, on the other hand, suggests that communication among processors may be a function of special purpose virtual machines.

Let us consider the triplet of virtual machines as depicted in Figure 3. The three machines include a physical machine, an intermediate-level virtual machine, and a virtual machine of the highest level in the system. Communication between extrema machines is mapped through the facilities of the intermediate machine, by definition. Realization of the level-one (intermediate) virtual machine is accomplished in the native language of the host; the level-two machine is realized in the software of the level-one machine.

A fault tolerant computing system designed in a multi-level fashion requires the communication of information of a special nature between pairs of virtual machines. For example, since physical hardware failures at the host level may induce failures at higher levels, a mapping of the operational status of host functional units upward is essential. Likewise, reconfiguration initiated by virtual machines in order to bypass faulty physical hardware will require downward communication, perhaps in the form of component enable/disable instructions based on conditional recovery strategies. Any adjacent pair of virtual machines must maintain this mutual mapping of status in a commonly accessible store. These mapping functions are intended to be extensions of those found in (12).

Diagnosis of hardware failure must of course precede the initiation of any recovery strategy determination. Diagnosis of failures in a multi-level computing system is, as expected, a multi-level problem. Solutions to the individual problems are diverse, ranging from hardware check circuits and flag bits to algorithmic processes. The tradeoffs to be considered in the selection of one technique over another at the lowest level of the system will be between added hardware complexity on one hand and process generality and execution time on the other. So long as hardware costs continue to fall, more complex hardware may well be consistently the more attractive extreme. However, penalities of inflexibility of the system host may be payable in the future. Furthermore, whereas the diagnosis of higher levels of the system must be accomplished via algorithmic diagnostics, an investment in the systematic development of generalized diagnostic procedures for application at every level should result in significant benefits in reduced hardware design overhead and other areas.

Initiation of recovery strategies can be made the responsibility of the level-one virtual machine. Access to hardware functional units is direct at this level, facilitating the resumption of previous processor states with the greatest ease of implementation. Switching of standby componentry may also be effected at this level should that strategy be preferred. The basis for determination of recovery strategies may well be supplied in tabular form as part of an environment specified by some higher-level machine in the system. Intimate knowledge of the host should be required of any virtual machine other than the level-one machine however.

An example of a possible diagnosis-recovery sequence is given as follows:

Presume that a hardware failure has occurred, rendering a bank of local memory unreliable. Presume further that this bank of local memory was dedicated to the registers of a particular level-two virtual machine. Al level two, the unavailability of these registers is noted via an intermediate mapping by the level-one machine. At the time that the level-two machine was realized on the level-one machine, specification of a strategy for recovery from a failure in the registers was made. This strategy may have taken the form of a coded entry in a failure contingency table, an entry recommending "rollback to restart point and retry", or "reconfigure to apply available resources to replace registers", or even something on the order of "abort conditionally". The strategy specification may itself be conditional based upon the history of failures within the virtual machine at level two.

To review the foregoing discussion of multi-level fault tolerant computing systems: the philosophy of the system's organization is the distribution of diagnostic and recovery responsibilities across all levels of the system. Special purpose diagnostic hardware may not be a requirement of the host architecture, although certain features which are cost-effective and do not impact the generality of the design may be incorporated. Tha advantages of a <u>microprogrammable</u> host are considerable in view of the resolution capability of microdiagnostics (15, 19), and the minimal time penalities for level-one diagnostic process execution and for level-one reconfiguration control. The resolution capability of microdiagnostics is not limited by interface with an intermediate control structure, as are conventional software diagnostics. Figure 4 suggests the relationship of the several partitions of the diagnosis and recovery responsibility; shaded areas indicate that some portion of a given virtual machine is devoted to diagnosis, recovery, and status mapping.

<u>Comparative analysis and refinement of techniques for diagnosis of dynamically microprogrammable processors with fail-safe or fail-soft requirements.</u>

Diagnosis of faults in dynamically microprogrammable processors in a fail-safe or fail-soft restricted environment may require access to and analysis of a considerable quantity of historical data regarding the status of on-line and off-line subprocessor units and components. Data management in such an environment is a critical function, both from the standpoint of the integrity of stored data and on the issue of retrieval time, especially when recovery strategies include either program rollback or possible system reconfiguration. Correlation of fault detection reports is an area which may require particularly high search rates, as regards the isolation of transient failures among diverse processor subsystems.

Recognizing fault data management as a significant problem area, care must be taken in the selection of diagnostic methods for use with microprogrammable processors. Checkout techniques which require storage of large numbers of test patterns may compound the data management problem by further restricting the amount of available memory, a commodity limited in simpler systems by addressability limitations. Concurrent testing of in-service functional units is an alternative to exhaustive checkout of an otherwise idle unit. The presence of exhaustive diagnostic microcode in a processor control store creates a non-trivial problem in itself, by drastically reducing the amount of memory available for emulators and therefore dictating additional overhead necessary to "page in" diagnostics from some secondary storage, be it main store or floppy disc (11).

One tentative solution to this problem is the liberal application of error-detecting and error-correcting codes (9, 22, 8, 1, 2, 21) to both control and data words in the processor, a solution which has applied in several contemporary systems (3, 11). Addition of check bytes, interspersal of error codes, and appendage of redundant encodings to words in the processor and its various memories is an approach which may minimize the number of storage locations required for diagnosis-associated firmware. These techniques invariably require an expansion of processors, memory, registers, and bus widths. Byte-serial implementations (3) are also feasible, of course.

An alternative approach for the application of error codes is the use of an auxiliary memory/functional unit in tandem with control store and/or main store. This "Snooper Store" (27) could be considerably wide than control store, and would be charged with storage of error codes for the protection of data words stored in <u>associated</u> locations in the companion store. The simplest association of addresses in the pair of stores is the <u>identity association</u>, for which a fetch of codes at a particular Snooper Store location would be initiated simultaneously with a fetch of the control store at the identical address. The Snooper Store must be writable as well, not only to accomodate modifications to the data words in the store it protects, but also that coding schemes may be user-altered as need be during execution. Residual control of page or bank addressing may provide an option for multi-emulator utilization of the Snooper Store just as such an addressing scheme may be applied for control store or main store. The flexibility of interpretation of error coding strategies may be enhanced by use of such an auxiliary memory. Realization of the Snooper Store as an associative memory may add to this flexibility by incorporating distributed logic for the analysis of coded words without added time or firmware. Further research is required in this area and is currently underway.

There are indeed other classes of hardware failures which may not manifest themselves as data word errors. Data transformations in processor functional units and the transport of data between registers of the processor are two important classes for consideration. Codes may again be critical to the concurrent checkout of functional units of the hardware (22, 8, 1, 2, 21). Validation of bus transport is imperative as well, both from the standpoint of data integrity and as regards accurate selection of source and destination for the transfer. These topics have been treated in detail (3) for a particular bussing structure, but are the subject of ongoing study over a variety of data transport structures.

Realization of firmware diagnostic routines initiatable by virtual machine emulators (in real time) on a given host.

At the time of this writing, microdiagnostics for the MATHILDA (27) machine are under development. At attempt is being made to realize a level-one partition of diagnosis and recovery algorithms for a proposed fault tolerant multi-level system in the microcode of MATHILDA -- completely in firmware without radical assumptions regarding the capabilities of error detection by the existing hardware. This "firmcore" partition will have responsibility for detection and location of hardware failures, and for communication of failure reports to successive levels of the system .

Algorithms devised for the diagnosis of the physical hardware at level one will likely be applicable to the problem of diagnosing "virtual hardware" at higher levels. Experience in the development of diagnostics and microdiagnostics (24, 19, 23) has dictated the starting point of this effort. There will be more to report at a later date.

Delineation of architectural constraints for microprogrammed fault tolerant computers.

This research area consists of a synthesis of information gathered and assessed in the development of each of the foregoing topics, in an effort to identify the minimum set of functional units (memories, arithmetic-logic units, coding schemes, etc.) necessary to the structure of a microprogrammed fault tolerant computing system. The goal of this aspect of the overall study is thus to determine the most effective design for the host level of a multi-level fault tolerant computing system.

Speculation of the composition of the host at this date maybe somewhat presumptuous. It is possible, however, that diagnostic data paths may be transporting data in parallel with main machine data paths in order that execution of both diagnostic processes and target processes be expedited when considering machines of a given cost/performance ratio. By the same token, a horizontal control structure may be required for such an organization, perhaps with micro-operations provided specifically for checkout and testing of componentry otherwise idle during a given machine cycle. The mapping of hardware status through various levels of multi-level system may best be facilitated via a multiple-key associative memory. Such alternatives as are mentioned here may or may not figure in the final design, pending further research.

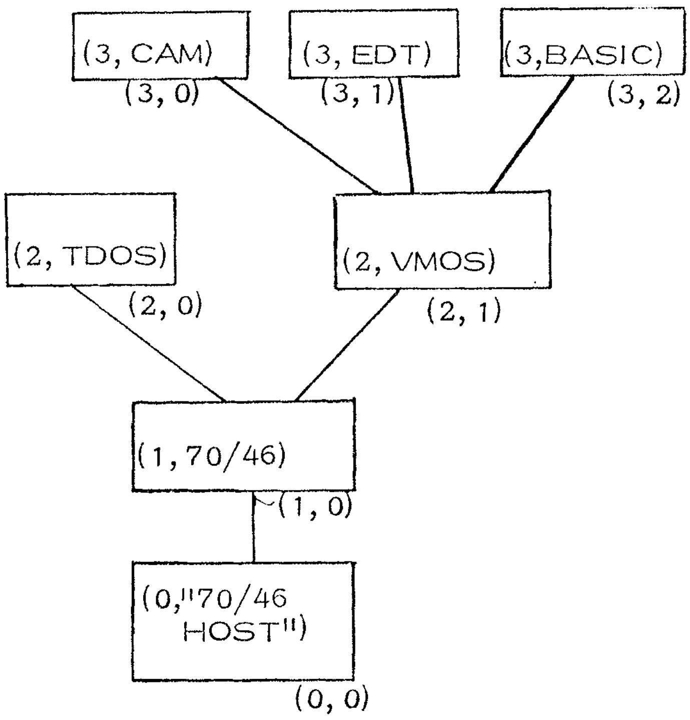

Figure 1. A Multi-level Computing System, the Univac 70/46 System.

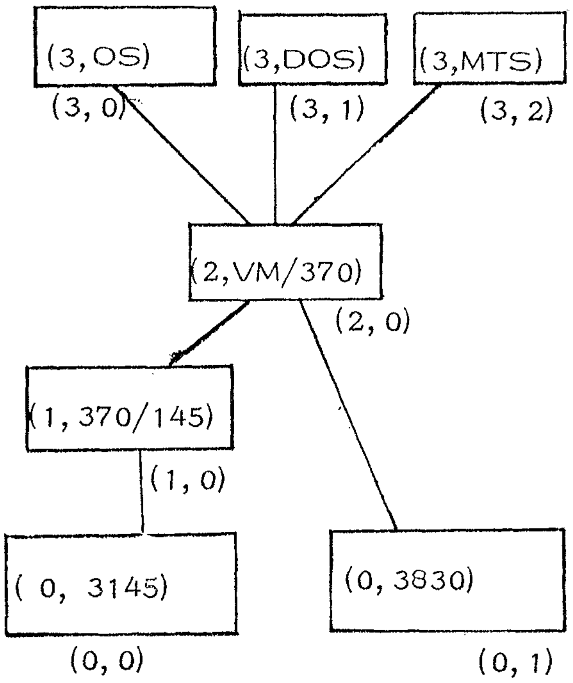

Figure 2. Dual-host Multi-Level Computing System, a System 370 subset.

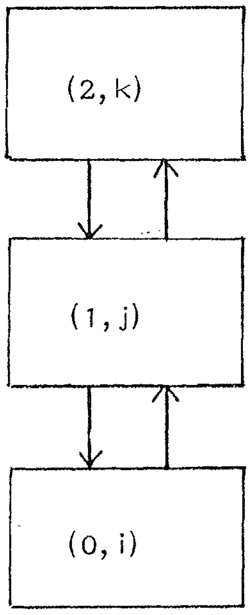

Figure 3. Basic Triplet of a Multi-level Computing System.

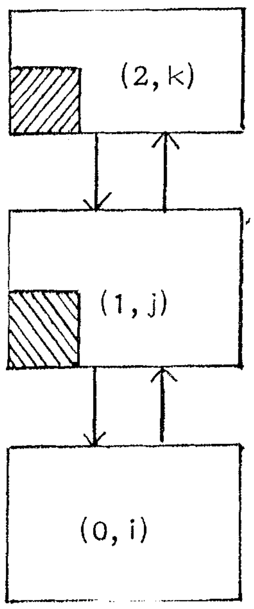

Figure 4. Basic Triplet of a Multi-level Fault Tolerant Computing System, Depicting Distributed Diagnosis and Recovery Compon ɜnts.

REFERENCES.

1. Avizienis, A., "Arithmetic Error Codes: Cost and Effectiveness Studies for Application in Digital System Design", IEEE-TC 20, 11, 1322-1331, Nov. 1971.

2. Avizienis, A., "Arithmetic Algorithms for Error-coded Operands", IEEE-TC 226, 567-572, June 1973.

3. Avizienis, A., et al, "The STAR (Self-Testing and Repairing) Computer: AN Investigation of the Theory and Practice of Fault-tolerant Computer Design", IEEE-TC 20, 11, 1312-1321, Nov. 1971.

4. Avizienis, A.,"The Methodology of Fault-tolerant Computing", First USA-JAPAN Computer Conference, 1972. Session 13-2-1. 6-14.

5. Bartow, N., McGuire, R., "System/360 Model 85 Microdiagnostics", AFIPS FJCC 1970 36, 191-197.

6. Borgerson, B. R., "Dynamic Confirmation of System Integrity", AFIPS FJCC 1972 41, 89-96.

7. Bouricius, W. G., et al, "Reliability Modeling for Fault-tolerant Computers", IEEE-TC 20, 11, 1306-1311, Nov. 1971.

8. Brown, D. T., "Error Detecting and Correcting Binary Codes for Arithmetic Operations", IRE-TC, Sept. 1960.

9. Chien, R. T., "Memory Error Control:Beyond Parity", IEEE Spectrum, 18-23, July 1973.

10. Corman, D., Burns, J. G., "A Comparison of Digital Simulation and Physical Fault Insertion in Diagnostic Test Development", Digest of Papers of the 1972 International Symposium on Fault-tolerant Computing, Newton, Mass., June 19-21, 1972, 42-46.

11. Ellenby, J., "Microprogramming in I/O ", Infotech State of the Art Lectures, Microprogramming and Systems Architecture, April, 1973, London, England.

12. Goldberg, R. P., "Virtual Machine Architecture", Honeywell Computer Journal, 7, No. 4, 251-260 (1973).

13. Guffin, R. M., "Microdiagnostics for the Standard Computer MLP-900 Processor", IEEE-TC 20, 7, 803-808, July 1971.

14. Hodges, K. J. H., "A Fault-tolerant Multiprocessor Design for Real-Time Control", Computer Design 12, 12, 75-81, Dec. 1973.

15. Husson, S. S., Microprogramming Principles and Practices, Prentice-Hall, Inc. Englewood Cliffs, N. J., 1970.

16. Johnson, A. M., "The Microdiagnostics for the IBM System 360 Model 30", IEEE-TC 20, 7, 798-803, July 1971.

17. Leaman, R. J., Lloyd, M.H., Repton, C.S., "The Development and Testing of a Processor Self-test Program", Computer Journal 308-314, 1973.

18. Mathur, F. P., "Trends in Fault-tolerant Computer Architecture", International Workshop on Computer Architecture, Grenoble, France, June 26-28, 1973.

19. Parhami, B., "Diagnostic and Microdiagnostic Techniques for Digital Systems", (unpublished lecture notes) University of California at Los Angeles, March 1974.

20. Parke, N. G., IV, Barr, P.C., The SERF Fault-tolerant Computer. IEEE Catalog no. 73cho772-4C. Fault-tolerant Computing 1973. 27-31.

21. Peterson, W. W. Rabin, M. O., "On Codes for Checking Logical Operations", IBM Journal, 163-168, April 1959.

22. Peterson, D. T., "On Checking an Adder", IBM Journal, 166-168, April 1958.

23. Rachlin, J.K., A Proposed Scheme of Microdiagnostics for a Microprogrammed Computer, Departmental Report 36, 72-MU, Department of Computer Science, State University of New York at Buffalo, Aug. 1972.

24. Ramamoorty, C. V., Chang, L.C., "System Modeling and Testing Procedures for Microdiagnostics", IEEE-TC 21, 11, 1169-1183, Nov. 1972.

25. Short, R. A. "The Attainment of Reliable Digital Systems through the Use of Redundancy -- A Survey", Computer Group News, 3, 2-17, March 1966.

26. Shriver, B.D., Lewis, T. , and Anderson, W. eds. "Project Beta: Emulation and Evaluation of Virtual Systems", Computer Science Department Report, University of Southwestern Louisiana, Lafayette, 1974. ⁎

27. Shriver, B.D., A Description of the MATHLIDA system, DAIMI PB-13, University of Aarhus, Denmark, April 1973.

28. Stiffler, J. J., The SERF Fault-tolerant Computer. IEEE Catalog No. 73cho772-4C. Fault-tolerant Computing 1973. 23-26.

29. Wensley, J. H., "SIFT--Software Implemented Fault Tolerance", AFIPS FJCC 1972 41, 243-253.

30. Wilkes, M.V., "The Best Way to Design an Automatic Calculating Machine", Manchester University Computer Inaugural Conference, 1951, 16.