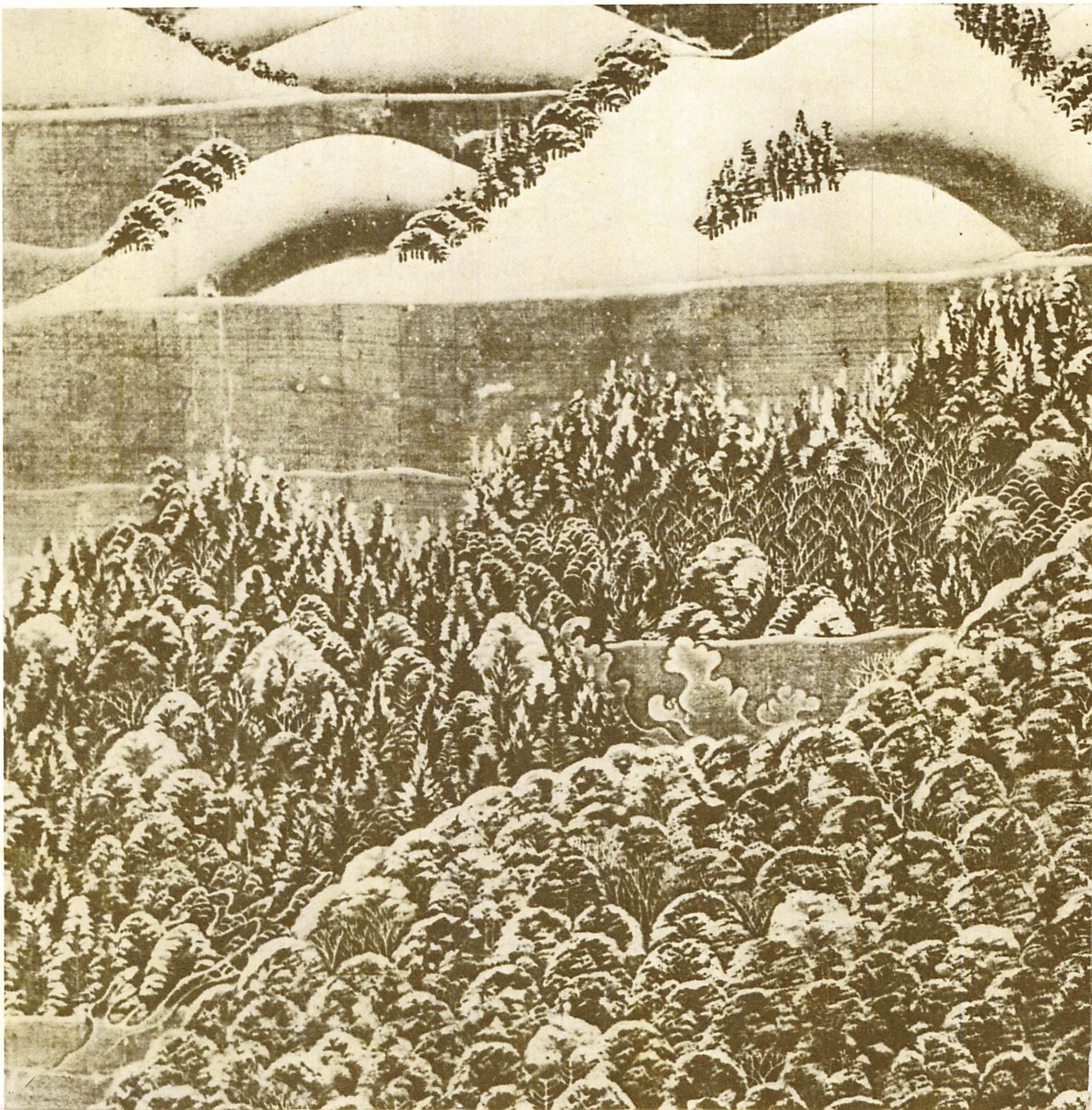
1985 INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS PROCEEDINGS

85CH2114-7 VOLUME 2 of 3



Proceedings of ISCAS 85

A Fault Diagnosis Algorithm Based On Intelligent Reasoning

Tao Yang and K. C. Smith

Department of Electrical Engineering
University of Toronto
Toronto, Canada

Abstract

In this paper, a system-level diagnosis algorithm for the FTMCS[14] is discussed. Unlike traditional diagnostic approaches, this algorithm applies techniques of artificial intelligence and expert systems. It collects naturally available information and status data as fault symptoms and reasons with these symptoms, using inference rules to discover the components which are responsible for the fault. Inference rules represent the knowledge of the FTMCS structure at a highly abstract level. The performance of the algorithm can be improved by modifying the inference rules.

1. Introduction

The past emphasis of fault tolerant and other computer reliability techniques has generally been on such specialized areas of application as spacecraft control [1,2,3] and telephone switching[4]. As a result of a great deal of work, various sophisticated reliability techniques have been developed for such major systems[5],[6]. On the other hand, there are a very large number of less sophisticated applications in which microprocessors are used to control particular devices in industry and manufacturing[7]-[11]. Moreover, this area of low-cost microprocessor-based controllers is rapidly expanding as developments in VLSI appear. However, it is apparent that it is an area to which too little attention has been paid in the application of reliability techniques.

Several trends strongly suggest that a time for change is upon us - that specific reliability techniques should be developed and implemented for low-cost systems:

A. Developments in VLSI technology, especially of microprocessors and memories, have provided new and less costly ways to construct reliable and fault tolerant computers and computer-based systems.

B. As the use of VLSI technology expands, even highly critical applications adopt microprocessors in their controlling elements. For these applications, current microprocessor-based controllers are not sufficiently reliable [12],[13].

Motivated by these trends, we have designed a Fault-Tolerant MicroComputer System (FTMCS) which is intended for industrial and manufacturing control. The design of the FTMCS was guided by the following objectives:

1. *Permanent and transient fault survival:*
2. *On-line repair:*
3. *Degraded operation:*
4. *Modular design:*
5. *Commercial VLSI components:*
6. *Dynamic response-time / reliability trade-off:*
7. *Self-diagnosis:*

In this paper, we will introduce a fault diagnosis algorithm for the FTMCS. This algorithm applies AI techniques-such as deductive reasoning (from facts to conclusions) and hypothesis reasoning (from assumptions to expected facts). In order to provide the context for its description, we will briefly introduce the FTMCS system.

2. System Architecture

The FTMCS was created to satisfy the requirements underlying the objectives presented in the last section. The system is configured with three identical computing modules connected by communication channels. The structure is shown in Figure 1 (For detail, see [14]).

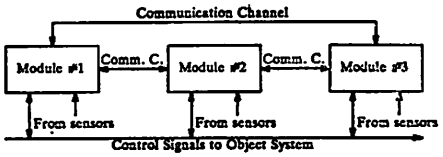


Figure 1. System Architecture

Each of the identical computing modules executes the same program (the control program). To begin, each accepts feedback signals from the object system (the signals can be sent by either a single sensor group or triplicated sensor groups). Then each calculates the corresponding control signals independantly, and, after this, each module broadcasts its computed result to the other modules for majority-voting. By this means, any single error caused by a single hardware failure can be detected and corrected. To recover from transient failures, the execution of the control program is modularized by task and the modules are task-synchronized. That is, after the completion of each task, the modules are required to exchange important information in order to mask errors caused by transients. The details of this process are discussed in [14].

If a module or one of its components is detected to be faulty, that module is isolated from the others. The system, thus, is degraded to a dual module one. In this degraded mode, two modules execute the same program as usual. They also exchange computing results and system information in order to detect permanent and transient failures. Many other features are software-implemented and are discussed in [14].

The module structure is shown in Figure 2.

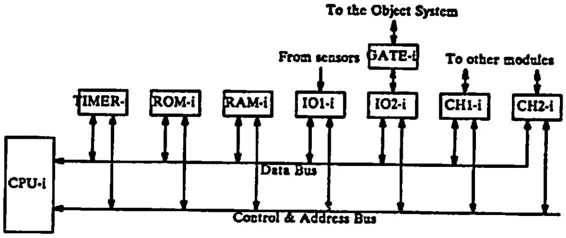


Figure 2. Module Structure

3. The Fault Diagnosis Algorithm

3.1. Background

The traditional model used conventionally in system-level diagnosis algorithms is that introduced by Preparata, Metze, and Chien[15]. In this model, a set of computing modules, or units, is assembled such that each unit can test a subset of the other units. It is assumed that at most a bounded number of units is permanently faulty and that these faulty units can claim either that fault-free units are faulty or that faulty units are fault-free. Based on this model, interesting results given by Hakimi and Amin[16] show that there exist necessary and sufficient conditions on the testing assignments of the units such that the set of the faulty units in the system can be uniquely identified on the basis of any possible collection of test results, assuming that the number of faulty units does not exceed a given bound, denoted as t. Such systems are said to be t-diagnosable. Since 1974, several algorithms based on this model have been developed by researchers in this field[17]-[21].

In recent years, there has been an increasing interest in applying techniques of artificial intelligence and expert systems to fault diagnosis for digital computing systems[31]. Several authors have proposed and developed new diagnostic methods which employ these techniques[24]-[30]. These methods are designed to implement diagnostic tools for electronic devices and other digital systems. Most of these diagnostic systems perform as troubleshooting consultants. It is generally assumed that the diagnostic systems themselvies are fault-free. The role of the troubleshooter is to initiate tests on the object system(the system to be diagnosed), then, reason with the test results according to rules incorporating knowledge of the structure and function of the object system to reach conclusions about the misbehaviour of the system. Often, the conclusions available are too general to point directly at the source of the fault, therefore, strategies must be taken to select further tests. These tests, intended to obtain further and sufficient information about the fault, may be done either with the help of human interaction or the knowledge stored in the system.

In this section, we will introduce a new diagnostic method which borrows the idea of troubleshooter reasoning described above. However, the method presented here is basically different from those intelligent diagnostic tools in several aspects: First, it is a self-diagnosis algorithm, not a diagnostic tool for some other system. Second, it is a system-level diagnosis algorithm while most of the troubeshooters operate at the circuit level. Third, the diagnosis is based on fault symptoms collected from naturally occurring system operational information and status, not from the results of special tests.

The diagnosis system under discussion collects system operational information and status which normally includes fault symptoms. It then reasons with these fault symptoms, using inference rules, to discover the cause of the fault or to identify the components which are responsible for the fault. The inference rules provide the means for the system to analyze the symptoms for the purpose of detecting their root cause. In general, inference rules are dependent on the actual system architecture. They represent the system structure at a highly abstract level. Thus, the performance of the algorithm depends on how well the inference rules represent the system.

One of the advantages of the approach used is that the algorithm can be improved by modifying only the inference rules and the organization of information about the system. Another advantage is that the algorithm does not require explicit testing.[1]

The third feature of the algorithm is that it does not utilize fault dictionaries and has no knowledge of fault models or fault mechanisms.

3.2. The System Model As Viewed By The Algorithm

In this subsection, we will modify the representation of the system structure for the convenience of the diagnosis algorithm. First, we would like to define some terms.

Definition 1: A *component* is defined to be a VLSI chip, denoted by a string of characters (its name). Referring back to Figure 2., CPU-i, TIMER-i, IO1-i, CH1-i, etc. are examples of components.

Definition 2: A *mechanism* is defined as a set of components, denoted by a string of characters (its name). Logically, a mechanism is a functional part of the system. It is also the smallest unit the diagnosis algorithm can distinguish. For example, a communication channel CM12 (see Figure 3) is a mechanism consisting of components CH1-1 in module #1 and CH2-2 in module #2. The mechanism OUT in Figure 3 is another example.

Definition 3: An *event* is defined as a boolean variable which has a value of either "true" or "false". An event is denoted by a string of characters (its name). Events are merely symbols used to describe system fault symptoms, denote inference rules and represent diagnostic conclusions. If event A is true, we say that A has happened, or conclusion A is valid, etc.. Events can also be considered subject to boolean operators such as "AND", "OR", and "NOT". Any proper boolean expression in which events are variables is also an event. Events can be classified into two kinds: *atomic events* and *non-atomic events*. The definitions of both will be given shortly in section 3.3.

Figure 3 shows the correspondingly modified representation of the system structure. Each box represents the mechanism named in it.

Mechanism #1 consists of components CPU-1, TIMER-1, ROM-1, and RAM-1 (Figure 2). Mechanism IN1 contains IO1-1. Mechanism OUT1 consists of components GATE-1 and IO2-1. Mechanism CM12 (also called CM21) contains components CH1-1 and Ch2-2. Other mechanisms in Figure 3 have similar organizations as those described above.

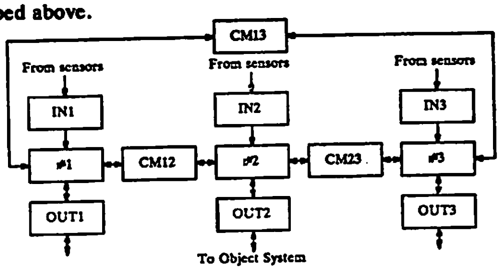


Figure 3. The System Model for Diagnosis

3.3. Description of the Algorithm

Before we describe the algorithm, we would like to state our assumptions:

Assumption 1: the algorithm will diagnose only permanent faults (since transient faults are treated in some other way [14]);

Assumption 2: more than one mechanism can be faulty at any time;

Assumption 3: no two different faulty mechanisms produce the same fault symptom;

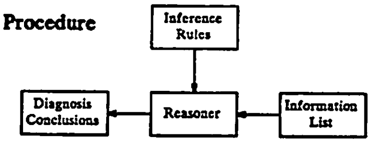
Assumption 4: no fault-free mechanism becomes faulty during diagnosis;

Assumption 5: a faulty mechanism manifests its fault by presenting incorrect output.

As shown in Figure 4, the diagnosis algorithm consists basically of three components: an information list, inference rules, and a reasoner. The information list contains the system fault symptoms. Inference rules are IF THEN statements which can be used to reason deductively with the system symptoms. The reasoner is the heart of the algorithm. It choses how to reason, which symptoms should be applied to which inference rules and what conclusion should be made, etc.. The objective of the algorithm is to identify

[1] Though the FTMCS uses extra software, the role of this software is to ensure correct operation, not to diagnose the source of the difficulty. We use its outputs, however, as data for the diagnosis process.

Figure 4. Diagnosis Procedure



the mechanism or provide a list of probable mechanisms which are responsible for the system fault symptoms depicted in the information list. Following is a detailed description of the algorithm.

3.3.1. The Information List

The information list is a set of basic events or atomic events in the system. The values of these events serve to fully describe the fault symptoms of the overall system. These events are basic or atomic (and will be refered to in this paper as atomic events) because their values do not logically depend on other events and are derived directly from the system operational information and status. These events must be both readily accessible and representative of fault symptoms.

As mentioned earlier in section 2, there are basically three fundamental voting processes in the system. The first is the vote taken of input signals from the sensors in order to make sure that every module has received the same inputs. This vote is also used to check the IN mechanism of each module. Since three values (v1, v2, and v3 from different modules) are voted, there are five possible voting results at any particular voting in each particular module. Because this voting process is held independantly in each of three modules, for each event, there will be three voting results, one coming from each of three modules. The voting results will be denoted as Imn, where m and n are decimal digits. The m specifies in which module the voting is held, while n denotes the result of the voting as either 0, 1, 2, 3, or 4, where:

n-values	Voting-results	n-values	Voting-results
n=0	v1=v2=v3	n=3	v1=v2, v3≠v1
n=1	v2=v3, v1≠v2	n=4	v1≠v2≠v3≠v1
n=2	v1=v3, v2≠v1		

For example, I10 means that the input vote held in module #1 indicates that v1=v2=v3. Another example is I23 which means that the input vote in module #2 judges that v3≠v1=v2.

The second fundamental voting process in the system is result voting. The values to be voted are the final computed results. In correspondence to the notation for input voting, result voting is denoted by Rmn. Here the m and n have the same meaning as in Imn. Similarly, the third process is output voting denoted as Omn.

After examining the conditions that must be met, we choose the voting results as atomic events in the same notation as used for voting results. Therefore, we identify three groups of atomic events.

The first group corresponds to input voting: I10, I20, I30; I11, I21, I31; I12, I22, I32; I13, I23, I33; I14, I24, I34;

The second group corresponds to result voting: R10, R20, R30; R11, R21, R31; R12, R22, R32; R13, R23, R33; R14, R24, R34;

The third group corresponds to output voting: O10, O20, O30; O11, O21, O31; O12, O22, O32; O13, O23, O33; O14, O24, O34;

Since these are events, each could be either "true" or "false". If, for example, I22 is true, this means that the input voting process of module #2 says that its input signals are different from those of modules #1 and #3 and those of #1 and #3 are the same.

Now, we would argue that these events are readily available since the system will always hold votings for inputs, results, and outputs. To implement the diagnosis algorithm, it is only necessary that this information be automatically recorded by the system. Finally, it is apparent that any occurrence of a faulty mechanism will be noted in these events. Therefore, these events meet our requirements for atomic events.

3.3.2. Inference Rules

Inference rules are simply a collection of IF THEN statements. They have the following general form:

IF e1 THEN e2

where e1 is boolean expression of events, and e2 is boolean assignment expression based on events.

Before we describe the inference rules, we would like to extend the idea to include other events. We claim that the names of the mechanisms in section 3.2 can be used here as notations for these events. We name this group of events non-atomic events since the values of these events are determined by those of atomic events. If the value of a non-atomic event is true, then the corresponding mechanism is functioning properly, otherwise, it is faulty. Therefore, for example, #1="true" means mechanism #1 is fault free.

The inference rules are divided into three groups: first order rules, second order rules, and third order rules. In first order rules, the e1 expression contains only atomic events. The e1 expression of the second order rule can contain both atomic and non-atomic events. However, the e1 expression of third order rules consists of only non-atomic events. For result voting, there are three first order rules, three second order rules, and two third oder rules.

In the following, i, j, and k are integers between 0 and 4, where i≠j≠k≠i.

```
The first order rules:
  FOR1   IF Ri0 AND Rj0 THEN CMij=true AND #1=true AND #2=true AND #3=true
  FOR2   IF Ri0 AND Rjk THEN CMij=true AND #i=true AND #j=true
  FOR3   IF Rik AND Rjk THEN CMij=true AND #i=true AND #j=true
The second order rules:
  SOR1   IF Rij AND #i THEN CMij=false OR #j=false
  SOR2   IF Rij AND #j THEN CMij=false OR #i=false
  SOR3   IF Ri4 AND #i THEN (CMij=false AND Cmik=false) OR (CMij=false AND
         #k=false) OR (CMik=false AND #j=false) OR (#j=false AND #k=false)
The third order rules:
  TOR1   IF (NOT CMij) AND #j THEN Rji OR Rj4
  TOR2   IF (NOT #i) AND #j THEN Rji OR Rj4
```

The reader may note that we have described the inference rules in terms only of the second group (i.e. the R group) of atomic events. In fact, the inference rules corresponding to other groups of atomic events are similar to the rules described above. For simplicity, we will not list them here.

3.3.3. The Reasoner

The reasoner is the core of the algorithm. It applies the inference rules to the atomic events, and generates some non-atomic events which can be regarded as conclusions. The reasoning procedure can be divided into four steps:

First step: apply all possible first order rules to the available atomic events, and generate corresponding non-atomic events if any;

Second step: apply all possible second order rules to the available atomic and non-atomic events if any. If there is no non-atomic event at this time, assumptions can be made, and the second order rules can be applied to the assumptions. This step is intended to identify the suspects;

Third step: If it is possible, apply third order rules to discriminate some suspects. In this step, assumptions will be made about some non-atomic events;

Fourth step: Examine the non-atomic events obtained in the last three steps and analyze the relationships amongst these non-atomic events. Then generate the final conclusion.

The first step is actually reasoning from facts to conclusions. In this step, some conclusions can be derived directly from the facts, that is, from the atomic events. However, in this step, only some of the fault-free mechanisms are determined, and no information about the faulty mechanism(s) is generated.

In the second step, suspects for the faulty mechanism are determined according to atomic events and non-atomic events. However, there are cases where for a given information list, no non-atomic events can be generated in the first step. In this case, the reasoner has to reason from hypothesis to possible facts. It first makes an assumption, then checks the assumption with the atomic events and inference rules to see if the assumption is in conflict with the atomic events. If so, the assumption is wrong, otherwise, the assumption is correct.

The third step is intended to discriminate amongst the suspects. Usually, among the suspects generated in the second step, there are mechanisms which are not faulty. These mechanisms can be eliminated from the suspect list by hypothesis-based reasoning.

The fourth step is required to organize the suspects into a list. It will sort the suspects in terms of probability of being faulty, and suggest to the user which mechanism is the most probably faulty one.

3.4. Examples

The reader may find it easier to understand the capability of the algorithm by working through examples.

Example 1 Assume that the information list contains atomic events (R10, R23, R32). Diagnosis:

```
FOR2  IF R10 AND R23 THEN CM12=true AND #1=true AND #2=true
FOR2  IF R10 AND R32 THEN CM13=true AND #1=true AND #3=true
SOR1  IF R23 AND #2 THEN CM23=false OR #3=false
SOR1  IF R32 AND #3 THEN CM23=false OR #2=false
```

List of Suspects: CM23, #2, #3

Since #2 and #3 are true according to FOR2, they should be removed from the suspect list. Therefore, the mechanism which is responsible for the fault is CM23.

Example 2: The information list is (R12, R21, R31). Diagnosis:

```
FOR3  IF R21 AND R31 THEN CM23=true AND #2=true AND #3=true
SOR1  IF R21 AND #2 THEN CM12=false OR #1=false
SOR1  IF R31 AND #3 THEN CM13=false OR #1=false
```

The list of suspects: #1, (#1 and CM12), (#1, and CM13), (CM12, and CM13), (#1, CM12, and CM13).

Assume #1 is functioning. Because of SOR1 both CM12 and CM13 must be faulty. Therefore, according to TOR1:

```
TOR1  IF (NOT CM13) AND #1 THEN R13 OR R14
```

Since the atomic events R13 and R14 are false, then the assumption that #1 is functioning is incorrect, that is, #1 is faulty. So, diagnosis conclusion is that mechanism #1 is faulty.

4. Future Extensions

In this paper, we have introduced a diagnosis algorithm for the FTMCS. However, while this algorithm has been designed and described specifically for the FTMCS, the concept which included in it can be applied to other systems even those which are much more complex. To facilitate the application of this idea in other areas, future research should:

(1) explore a general and systematic method and a corresponding language by which the user can define both atomic and non-atomic events;

(2) develop a language by which the user can define relationships amongst the events;

(3) provide a compiler which can translate the relationships amongst the events into inference rules;

(4) explore an existing programming language in which it is most suitable to implement the entire diagnostic system.

5. Conclusions

In this paper, we have introduced a system-level fault diagnosis algorithm for a low-cost controller, the FTMCS system. The algorithm is different from traditional methods in that it applies techniques of AI and expert systems. It has several advantages over the traditional methods.

While, this algorithm has been developed specifically for the FTMCS system described, we believe its basic approach to be applicable to other systems --even systems which are much more complex than the FTMCS.

6. Acknowledgements

The authors gratefully acknowledge the support of the Canada's Natural Sciences and Engineering Research Council through NSERC grant A1753.

7. References:

[1] George Gilly, "The Fault-Tolerant Spaceborne Computer," Annual Rocky Mountain Guidance and Control Conference, Feb. 1979, Keystone, Colorado.

[2] A. L. Hopkins, T. B. Smith, and J. H. Lala, "FTMP-- A Highly Reliable Fault-Tolerant Multiprocessor for Aircraft," Proceedings of the IEEE, Vol. 66, No. 10, October 1978, pp. 1221-1239.

[3] A. E. Cooper and W. T. Chow, "Development of On-board Space Computer Systems," IBM Journal of Research and Development, Vol.20, No. 1, Jan. 1976, pp. 5-19.

[4] W. N. Toy, "Fault-Tolerant Design of Local ESS processors," Proceedings of the IEEE, Vol.66, No.10, Oct. 1978, pp. 1126-1145.

[5] A. Avizienis, "Fault-Tolerant Computing-- Progress, Problems, and Prospects, " Proceedings of IFIP Congress 77, Toronto, pp. 405-420, Aug. 1977.

[6] S. Y. H. Su and R. J. Spillman, "An Overview of Fault Tolerant Digital System Architecture," AFIPS Conference Proceedings, Vol. 46, pp. 19-26, 1977.

[7] T. A. Nondahl and M. S. Hicken, "A Microprocessor-based Gate Pulse Generator for Phase-Controlled Rectifiers," IEEE 1983 IECON, pp. 314-317.

[8] M. Kunugi, et al, "Application of Microprocessor to Automatic Reclosing of Electric Power Systems," IEEE 1983 IECON, pp. 219-224.

[9] R. J. Evans, "METCOM--A Large Microcomputer Network for Fire Station Monitoring, I. E. Aust. Conference On Microprocessors, Melbourne, Aust. Nov. 1979, pp. 81-84.

[10] K. Rajaraman, et al, "A Microcomputer-based Elevator Control System," IEEE 1982 IECON, pp. 10-15.

[11] D. E. Orin, et al, "Dynamic Computer Control of A Robot Leg," IEEE 1983 IECON, pp. 256-262.

[12] A. M. Usas, "Fault Detection and Diagnosis in Digital Computer Input/Output Systems," Ph.D. Thesis, Stanford University, 1978.

[13] H. N. Peterson, "Are Fault Tolerant Control Sytems Affordable? " IEEE 1983 IECON, pp. 249-255.

[14] T. Yang and K. C. Smith, "A Proposed Fault-Tolerant Microcomputer system," Submitted for Publication to the Canadian Electrical Engineering Journal, Sept. 1984.

[15] F. P. Preparata, G. Metze, and R. T. Chien, "On the connection assignment problem of diagnosable systems," IEEE Trans. Electron Comput., vol. EC-16, pp. 848-854, Dec. 1967.

[16] S. L. Hakimi and A. T. Amin, "Characterization of the connection assignment of diagnosable systems," IEEE Trans. Comput., Vol. C-23, pp 86-88, Jan. 1974.

[17] T. Kameda, et al, "A diagnosing algorithm for networks." Informat. Contr., Vol. 29, pp. 141-148, 1975.

[18] A. M. Corluhan and S. L. Hakimi, "On an algorithm for identifying faults in a t-diagnosable system," in Proc. 1976 Conf. on Informat. Sci. Syst., The Johns Hopkins Univ., Baltimore, MD, pp. 370-375. Apl. 1976.

[19] G. G. L. Meyer and G. M. Masson, "An efficient fault diagnosis algorithm for symmetric multiple processor architectures," IEEE Trans. Comput., Vol. C-27, pp. 1059-1063. Nov. 1978.

[20] S. Mallela, "On diagnosable systems with simple algorithms," in Proc. 1980 Conf. on Informat. Sci. Syst., Princeton Univ., Princeton, NJ. Mar. 1980, pp.545-549.

[21] G. G. L. Meyer, "A fault diagnosis algorithm for asymmetric modular architectures," IEEE Trans. Comput., Vol. C-30, pp.81-83, Jan. 1981.

[22] A. T. Dahbura and G. M. Masson, " An O(n) fault identification algorithm for diagnosable systems," IEEE Trans. Comput., Vol. C-33, No. 6, June 1984. pp. 486-492.

[23] S. H. Hosseini and S. M. Reddy, "A diagnosis algorithm for distributed computing systems with dynamic failure and repair," IEEE Trans. Comput., Vol. C-33, No. 3, Mar. 1984.

[24] H. Shubin and John W. Ulrich, "IDT: an intelligent diagnostic tool," AAAI*-1982, pp. 290-295.

[25] R. Davis, et al, "Diagnosis based on description of structure and function," AAAI*-1982, pp.137-142.

[26] M. R. Genesereth, "Diagnosis Using Hierarchical Design Models," AAAI-1982, pp. 278-283.

[27] R. Davis, "Diagnosis via causal reasoning: paths of interaction and the locality principle," AAAI-1983, pp.88-94.

[28] N. Yamada and H. Motoda, "A diagnosis method of dynamic system using the knowledge on system description," IJCAI*-1983, pp.225-229.

[29] W. Hamscher, "Using structural and functional information in diagnostic design," AAAI-1983, pp.152-156.

[30] W. Hamscher and R. Davis, "Diagnosing circuits with state: an inherently underconstrained problem," AAAI-1984, pp. 142-147.

[31] R. A. Maxion and D. E. Morgan, "The application of artificial intelligence techniques to reliable and fault tolerant computing," FTCS-14, June,1984 pp.285-290.

Notes:
AAAI-- American Association for Artificial Intelligence;
FTCS-- International Conference on Fault Tolerant
Computing Systems
IECON-- Conference on Industrial Electronics
IJCAI-- International Joint Conference on Artificial
Intelligence

完