NASA TM-84482

NASA Technical Memorandum 84482

An Assessment of the Real-Time Application Capabilities of the SIFT Computer System

Ricky W. Butler

APRIL 1982

NASA
National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665

SUMMARY

The SIFT experimental computer system was designed to meet extremely high reliability requirements and to facilitate a formal proof of its correctness. These severe design constraints have impacted the user-interface in several ways. The system provides the user with static, nonpreemptive scheduling and requires that all tasks execute in less than 3 milliseconds. A tedious generation of vote and schedule tables is required of the user to coordinate the redundancy management capabilities of the SIFT system with his application workload. The characteristics of this user interface and its impact on application system design are assessed in this paper.

INTRODUCTION

The SIFT computer system developed by SRI International for NASA Langley Research Center is an experimental computer designed to support the flight controls of a relaxed static stability aircraft. Because the flight control functions are crucial in such aircraft, the reliability requirement for SIFT is a probability of failure not to exceed 10^{-9} for a 10-hour flight. The SIFT computer exploits an innovative approach to fault tolerance utilizing software controlled task replication and reconfiguration. The design of SIFT produced a new distributed clock synchronization algorithm and an interactive consistency algorithm of generic significance. Furthermore, the development of SIFT, using a formal proof methodology, has demonstrated the feasibility of such techniques on a nontrivial system-level problem.

The fault-tolerant characteristics of SIFT have been discussed in detail in the literature (refs. 1, 2, and 3); however, the application level capabilities have not been openly discussed. This paper will consider the SIFT system as it appears to the application user. Although the details of the user interface are not of concern, the capabilities which are inherently limited by the underlying redundancy management system are of much interest. Such limitations will be explored in this paper. Furthermore, this paper will describe the current state of the SIFT system which, after several major changes, is a system of less power and flexibility than the system described in the 1978 IEEE paper (ref. 1).

BASIC CHARACTERISTICS OF SIFT

The SIFT hardware consists of a set of Bendix BDX930 avionics computers fully interconnected by a serial, point-to-point broadcast network. Up to 8 processors may be included in the configuration. Each processor has 32K words of 16-bit memory and an instruction speed of approximately 1 million instructions per second. Two blocks of memory (1024 words each) are allocated for interprocessor communication. One of these blocks, called the "transaction file" is used to control the output information of a processor. The other block, the "datafile," is partitioned into 8 sections of 128 words each. One of these sections holds the output variables which are to be broadcast to the other processors. The seven remaining sections serve as "mailboxes" to receive information from the other processors in the system. The SIFT processors also contain a MIL STD 1553A bus interface for communications with the other aircraft systems.

The fundamental coordination of the SIFT system is accomplished through the use of a clock synchronization algorithm. Each processor contains its own real-time clock which must be synchronized with the other clocks in the system. Each clock in the system "corrects" itself relative to all other clocks in the system through use of a periodic broadcast technique. SRI has demonstrated that with a set of four or more processors, their synchronization algorithm will insure that the clocks will remain synchronized to within 50 microseconds, even in the presence of an arbitrary failure of one clock (ref. 2). Previously developed algorithms were found to be incapable of maintaining synchronization in the presence of a malicious lying clock (i.e. a clock that broadcasts different values to different clocks). The development of the SIFT synchronization algorithm is a significant accomplishment by the SRI team.

The basic mechanism of fault tolerance in SIFT is task replication and voting. Several replicates (i.e. identical copies of a task) are assigned to different processors in the SIFT configuration. Each replicate receives identical input and performs identical computations. The outputs of these replicates are "voted" to prevent propagation of hardware faults and to detect the failed processor. The vote discrepancies are noted by the executive system and later used to reconfigure the failed processor out of the working set of processors. Various degrees of reliability can be obtained for each task by using different amounts of task replication on different processors. The coordination of task replication and voting is accomplished through use of precalculated schedule and vote tables in each processor. Since task replicates on the various processors are controlled by schedule tables which are synchronized, all replicates receive the same data. Furthermore, through use of an "interactive consistency algorithm,"

developed by SRI, single source input data can be accommodated in the system (ref. 4). Without the algorithm, a failed processor, transferring single source data to the task replicates, could send different values to each replicate. This, in turn, could lead to the elimination of good processors from the system. The recognition and solution of this problem represents another significant achievement of the SRI team.

The SIFT operating system was designed so that a formal mathematical proof-of-correctness of the operating system could be made. This approach has had significant impact on the system structure and its capabilities. Because of the inherent complexity of the formal proof process, the system design has been kept extremely simple. The SIFT proof-of-correctness has demonstrated that formal techniques are applicable to system-level problems of much greater size and complexity than the problems found in most research papers on this subject. However, in many ways, this approach has resulted in a SIFT system of less power and flexibility than otherwise would have been possible. (See table 1.) In particular, a major simplification in the scheduling and voting strategies became necessary in order to formally prove the system correct. A priority-driven preemptive scheduler would have introduced unrestricted concurrency - the logically simultaneous execution of several tasks - into the system. The problem of proving a system of cooperating concurrent processes is still an open question, though progress has been made since the design of SIFT. Future generations of SIFT-like fault-tolerant systems may be able to incorporate the flexibilities and power of concurrency through priority-based schedulers as the formal verification technology advances. Nevertheless, the SIFT computer represents an important step in the evolution of formally verifiable fault-tolerant systems.

TABLE 1. - A COMPARISON OF SIFT TODAY WITH THE 1978 IEEE PAPER DESCRIPTION

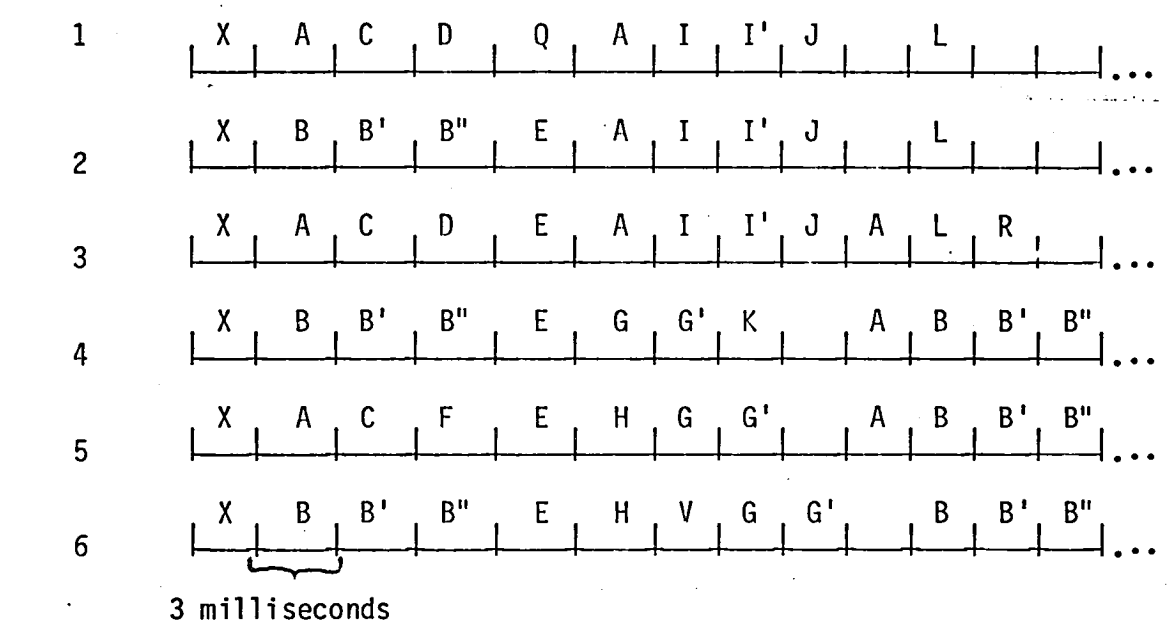
IEEE Paper	SIFT Today
Priority based periodic scheduling (preemptive)	Static preplanned scheduling (nonpreemptive)
Arbitrary task length	All tasks must fit in a subframe time slot
Dynamic allocation of tasks to processors	Static assignment of tasks to processors
Task replication is transparent to the application designer	The application designer must build schedule tables and statically assign task replicates to processors
Voting is transparent to the application designer through use of operating system routines to obtain interprocess data	The application designer must build a vote table which corresponds to the precalculated schedule table

THE APPLICATION DESIGNER INTERFACE TO SIFT

The SIFT operating system provides an interface to the applications designer which is significantly different from traditional real-time systems. The most striking feature of the SIFT system from the user's perspective is the requirement that all tasks must execute in less than 3 milliseconds. The reason for this restriction is that the voting algorithm in SIFT is based on a static schedule table and a vote table which consist of 3 millisecond time slots for the tasks. The schedule table determines the set of tasks and their order of execution on a SIFT processor. Since aircraft guidance and control functions consist of periodic sensor sampling, transfer function computations and actuator

command generation, this schedule table is executed repetitively. Functions with higher iteration rate requirements can appear more than once in the schedule table. Functions with lower rate requirements can use a flag to cause alternate skipping of a particular task. The schedule table corresponds to one major frame which consists of 30 subframes. Each task is statically assigned to one or more subframes and if a function requires more than 3 milliseconds to execute, it must be subdivided into 2 or more tasks. (See figure 1. - Function G, for example, has been subdivided into tasks G and G' and has been replicated on processors 4, 5 and 6). The creation of the static schedule table is the responsibility of the application designer.

FIGURE 1. - STATIC, NONPREEMPTIVE SCHEDULE TABLE CONCEPT



The tasks must be ordered according to their functional dependencies and structure the schedule table accordingly. However this is only a small part of the initialization which must be done. SIFT is designed to support task replication - identical tasks executing on different computers in the SIFT network - and is capable of reconfiguration. The applications designer must allocate the task replicates to the various processors to meet various reliability requirements. Some tasks may require 5 replicates, whereas others require only 1 or 3.

As long as there are no precedence constraints on the execution of the tasks (i.e., a particular ordering of tasks is not required), this is not an intractable problem. However, if precedence constraints exist, then the resulting "scheduling" problem can be very complex. Even ignoring the need to synchronize the task replicates, the complexity of finding an optimal schedule is NP-complete when 3 or more processors are to be scheduled (ref. 5). Furthermore, it is inevitable that precedence constraints will be present since the partition of a function (longer than 3 ms) into tasks generates a sequence of tasks which must be executed sequentially. (Although optimal schedules may not be necessary, this illustrates that generating such schedules is not trivial - especially good ones.) The reconfiguration capability of SIFT also increases the workload of the application designer. One table must be generated for every processor for every configuration (i.e., as processors fail and are removed we have a 6-processor configuration, then a 5-processor configuration, then a 4-processor configuration, etc.). If there are n processors in the system

initially, the user must generate n + (n-1) + (n-2) +...+ 2 tables or(n^2 + n-2)/2 tables. For the present SIFT configuration of 6 processors, 20 different schedule tables must be generated.

The SIFT processor achieves its fault tolerance from the voting of the task replicates. Whenever a task produces output data it must be voted. This is not transparent to the application designer. The application designer must enter the index of the output data in a "vote" table which must be generated for each system configuration. Hence, the designer must generate another n-1 vote tables for an n-processor SIFT. The total number of vote and schedule tables which must be created by the application designer on an n-processor SIFT is thus (n^2 + 3n - 4)/2 or 25 for a 6-processor system.

It should be noted that the proof of correctness of the SIFT system does not cover the application domain. The proof demonstrates that SIFT correctly masks and reconfigures single faults out of the system and will schedule tasks according to the user generated tables. The ability of the system to mask and isolate faults, however, is dependent upon the inclusion of the necessary intertask variables in the vote table. If an error is made in the schedule tables or vote tables, system failure is possible. Therefore, a reliable application system of SIFT requires a systematic and reliable approach to the generation of schedule and vote tables. Clearly, such a systematic approach must also use formal proof-of-correctness techniques since the entries of the vote table and schedule tables depend on the interaction of the application tasks (e.g., the presence or absence of entries in the vote table depends on what information must be sent from one task to another and this is determined by the formally specified functions these tasks must perform).

ANALYSIS OF SIFT CAPABILITIES

The primary constraint on the flexibility of the SIFT system is the nonpreemptive, unit-execution time scheduling philosophy. The advantages of problem decomposition and modularity have been widely discussed in the literature (refs. 6, 7, and 8), yet an artificial execution time constraint can only serve to complicate the system structure. To enhance program simplicity and reliability, the partition must be made on the basis of functional utility, not because of an arbitrary sizing constraint. This unit-execution constraint will cause the introduction of inefficient module interfaces and a consequent increase in intertask communication variables. This compounds another allocation problem in SIFT--the mapping of intertask variables to the 128 data file locations. If more than 128 intertask variables are needed, they must be "multiplexed" in time by the application user. This time-multiplexed allocation of intertask variables, however, would have to be carefully coordinated with the processor's task schedule.

The nonpreemptive characteristic of the SIFT scheduler limits the performance and flexibility of the system. A preemptive scheduler would offer several advantages over a nonpreemptive scheduler:

1) rapid response to asynchronous events (cf. in a nonpreemptive system a task must be continually dispatched which "polls" an event repetitively) (ref. 9);
2) more effective CPU utilization (i.e. a preemptive schedule can always be constructed which is shorter or equal to a nonpreemptive schedule when the number of processors $\geq$ 2) (ref.10);
3) accommodation of transient system overloads of crucial functions by <u>not</u> dispatching low-priority tasks (ref. 11);
4) support of arbitrary iteration rates for the tasks (cf. in a static table nonpreemptive schedule, all iteration rates must be multiples

of a base rate; hence, tasks sometimes must be scheduled to run at a higher rate than the function requires) (refs. 11 and 12); and

5) support of periodic task sets with up to 100 percent processor utilization while still guaranteeing that all real-time deadlines will be met on a single processor (ref. 12).

Another consequence of the nonpreemptive scheduler of SIFT is that support for high-level languages with flexible concurrent processing capabilities such as Ada or Concurrent Pascal is not possible. The flexibilities of the preemptive approach listed above are incorporated into these languages. The unit execution restriction on task length further aggravates this problem. Furthermore, task replication should be transparent at the application language level. But since asynchronous communication is possible in these languages and SIFT's replicates are controlled by static vote tables, the mapping to SIFT is impossible.

A LOOK TOWARDS THE NEXT GENERATION AIRCRAFT CONTROL SYSTEMS

The traditional approach to the development of an aircraft electronics system has been to partition the system functions and to design each independently (e.g., into flight controls, navigation, guidance, etc.). Often these functions are assigned to different computers. These functions are implemented as a set of tasks scheduled on the computers by a nonpreemptive cyclic executive with a predetermined execution sequence, where each cycle is initiated by a clock interrupt and every task is run to completion. In such a federated system approach, the SIFT computer could

support the flight critical functions such as stability augmentation and flutter-mode suppression in a relaxed static stability aircraft.

The new techniques of modern control theory, research for advanced aerodynamic concepts, and functional integration research are revealing significant benefits to be obtained from an integrated system approach (refs. 13). Through the integration of traditionally separate functions, higher aircraft performance is promised, but this will come at the cost of increased logical complexity and an increased computational workload. Capabilities exceeding the nonpreemptive cyclic executive approach will be needed to cope with the high degree of functional interaction and to provide rapid response to asynchronous events such as pilot commands and failed sensors or actuators. Also, the nonpreemptive, cyclic executive will be undesirable since it impacts the application system design in several adverse ways:

1) The static nature of the task schedule causes a proliferation of discrete (Boolean) variables in order that the tasks can be responsive to changes in flight phase and other asynchronous events. The availability of dynamic scheduling enables the designer to build tasks tailored to a single flight phase which are only dispatched when needed.
2) The addition of new low priority tasks necessitates a redesign of the static schedule table (ref. 9).
3) Task execution length variability results in inefficient utilization of the CPU since the schedules must be built for worst-case execution times. The proliferation of discrete variables as described above further compounds this problem (ref. 11).

Because of the inefficiencies and inflexibility of the task management approach presented above, the next generation aircraft systems will need the capabilities of a priority-driven preemptive approach (ref. 14). Furthermore, the software engineering methodologies are already extolling the virtues of high-level languages with concurrent processing capabilities (e.g., Ada) (ref. 15). These languages provide additional safety through detection at compile time of many time-dependent errors which are traditionally extremely hard to discover through testing alone. It appears to be desirable that the fault-handling systems of future fault-tolerant architectures be compatible with these high-level languages. It is important, therefore, that the future generations of SIFT-like architectures have capabilities beyond those of SIFT today. If this flexibility is unobtainable then future aircraft system designers are going to be limited in many ways.

CONCLUSIONS

The development of the SIFT computer system represents a major accomplishment in fault-tolerant systems technology. However, the high reliability requirements and the use of formal proof techniques have restricted the capabilities of the system in several ways and have placed severe demands on the application designer using the system. In particular, the requirements that the application designer must partition all functions into 3 millisecond tasks and manually delineate task replication, scheduling, voting, and reconfiguration are especially severe. Perhaps future generations of SIFT will be able to bring the redundancy management functions totally under the control of the operating system and present a

user-friendly interface to the applications designer. This undoubtedly will only come through major improvements in the software techniques used to accommodate hardware failures. If such improvements are unattainable, the users of SIFT-like computers will face new obstacles in the implementations of their application systems.

REFERENCES

1. Wensley, J. H.; Lamport, L.; Goldberg, J.; Green, M.; Levitt, K. N.; Melliar-Smith, P. M.; Shostak, R. E.; and Weinstock, C. B.: SIFT: Design and Analysis of a Fault-Tolerant Computer for Aircraft Control. Proceedings of the IEEE, Vol. 66, No. 10, Oct. 1978, pp. 1240-1255.

2. Goldberg, Jack: SIFT: A Provable Fault-Tolerant Computer for Aircraft Flight Control. Proceedings IFIP Congress 80, 1980, pp. 151-156.

3. Weinstock, Charles B.: SIFT: System Design and Implementation. The 10th International Symposium of Fault-Tolerant Computing, Oct. 1980, pp. 75-77.

4. Pease, M.; Shostak, R.; and Lamport, L.: Reaching Agreement in the Presence of Faults. Journal of the ACM, Vol. 27, No. 2, April 1980, pp. 228-234.

5. Lenstra, J. K.; and Rinnooykan, A. H. G.: Complexity of Scheduling Under Precedence Constraints. Operations Research, Vol. 26, No. 1, 1978, pp. 22-35.

6. Parnas, D. L.: On the Criteria for Decomposing Systems into Modules. Communications of the ACM, Vol. 15, No. 12, Dec. 1972, pp. 1053-1058.

7. Liskov, B. H.: A Design Methodology for Reliable Software Systems. AFIPS Fall Joint Computer Conference, 1972, pp. 191-199.

8. KLOS, Larry C.: An Interface Management Approach to Software Development. NAECON, 1978, pp. 741-747.

9. Post, David L.: Executive Architecture for Digital Avionics Systems. NAECON, 1978, pp. 714-724.

10. Coffman, Edward G.; and Denning, Peter J.: Operating Systems Theory. Prentice-Hall, Inc., 1973, pp. 115-116.

11. Maclaren, Lee: Evolving Toward Ada in Real-Time Systems. Sigplan Notices, Vol. 15, No. 11, Nov. 1980, pp. 146-155.

12. Liu, C. L.; and Layland, James C.: Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment, Journal of the ACM, Vol. 20, No. 1, Jan. 1973, pp. 46-61.

13. Deyst, John J., Jr.; and Hopkins, Albert L., Jr.: Highly Survivable Integrated Avionics. Astronautics and Aeronautics, Sept. 1978, pp. 30-41.

14. Bate, Roger R.: Distributed Microprocessors in Avionic Systems. 2nd Conference on Computers in Aerospace, Oct. 1979, pp. 252-257.

15. Wirth, Niklaus: Toward a Discipline of Real-Time Programming. Communications of the ACM, Vol. 20, No. 8, Aug. 1977, pp. 577-583.

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA TM-84482		

4. Title and Subtitle	5. Report Date
An Assessment of the Real-Time Application Capabilities of the SIFT Computer System	April 1982
	6. Performing Organization Code
7. Author(s)	8. Performing Organization Report No.
Ricky W. Butler	505-34-43-06
9. Performing Organization Name and Address	10. Work Unit No.
National Aeronautics and Space Administration Langley Research Center Hampton, Virginia 23665	11. Contract or Grant No.
12. Sponsoring Agency Name and Address	13. Type of Report and Period Covered
National Aeronautics and Space Administration Washington, DC 20546	Technical Memorandum
	14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

This paper discusses the real-time capabilities of the SIFT computer system, a highly reliable multicomputer architecture developed to support the flight controls of a relaxed static stability aircraft. The SIFT computer system was designed to meet extremely high reliability requirements and to facilitate a formal proof of its correctness. Although SIFT represents a significant achievement in fault-tolerant system research it presents an unusual and restrictive interface to its users. The characteristics of the user interface and its impact on application system design are assessed.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Flight Computers Fault-Tolerance Operating Systems Scheduling	Unclassified - unlimited Subject Category 61

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	16	A02

For sale by the National Technical Information Service, Springfield, Virginia 22161