NASA Technical Memorandum 4218

Abnormal Fault-Recovery Characteristics of the Fault-Tolerant Multiprocessor Uncovered Using a New Fault-Injection Methodology

Peter A. Padilla

MARCH 1991

NASA Technical Memorandum 4218

Abnormal Fault-Recovery Characteristics of the Fault-Tolerant Multiprocessor Uncovered Using a New Fault-Injection Methodology

Peter A. Padilla
Langley Research Center
Hampton, Virginia

NASA
National Aeronautics and Space Administration
Office of Management
Scientific and Technical Information Division
1991

Contents

Tables

Figures

Summary

A new design methodology for a fault-injection experiment was used to uncover and characterize problems with the Fault-Tolerant Multiprocessor (FTMP) error detection and faulty-unit isolation functions. The observed anomalies were discovered during hardware fault-injection experiments performed on the FTMP test-bed at the Avionics Integration Research Laboratory (AIRLAB) at the NASA Langley Research Center. During these experiments, the FTMP occasionally disabled a working unit instead of the faulted unit once every 500 fault injections, on the average. This behavior continued until the system crashed after 4000 to 6000 fault-injection tests.

The new fault-injection methodology involves new criteria for selecting the fault location (high fan-out signals) and new instrumentation to observe the behavior of the system in real time. This new fault-injection methodology can be applied to other fault-tolerant-system architectures to uncover design problems. Using this methodology, weaknesses in the FTMP design were found. The weaknesses increase the probability that any active fault can exercise a part of the fault-management software that handles Byzantine or "lying" faults.

Byzantine faults behave in such a way that the faulted unit points to a working unit as the source of errors. When errors generated by the fault occur in certain time intervals during the detection data acquisition, the FTMP fault-identification procedure incorrectly decides that only one line replaceable unit (LRU) detected the error (the so-called lone-accuser). This event triggers a software component, designed to handle Byzantine faults, that keeps track of the identity of the lone-accuser and disables it after a second event. In summary, the combined effects of the design weaknesses of the system result in a significant probability that a fault-handling error occurs in such a way that one fault causes two LRU's to be disabled, a good unit in addition to the faulty unit.

This paper gives some background material on the FTMP architecture, and then it proceeds to describe a typical fault-injection experiment and the observed anomalous behavior. Following these descriptions, the causes of the erroneous behavior are described. A simple model was constructed that predicts the general pattern of these events and their rates based on measurable system parameters. The paper ends with calculations of the rate at which an intermittent/transient fault or a permanent fault might induce the observed phenomena and with conclusions reached during this study.

Introduction

The Fault-Tolerant Multiprocessor (FTMP) is a multicomputer system designed according to fault-tolerant design principles to survive and continue operation after the occurrence of several nonsimultaneous faults. (See ref. 1.) The intended application of the system design is flight-critical applications where a high reliability is required to ensure the survival of the vehicle, crew, and passengers. The often-stated reliability goal is expressed as a probability of failure of less than or equal to 10^{-9} for a 10-hour mission. An engineering prototype of the FTMP was constructed in the late 1970's, and preliminary testing was conducted by the system designers. (See ref. 2.) Thereafter, the system was delivered to the NASA Langley Research Center where it has been subjected to more extensive experimentation. (See ref. 3.)

Early fault-injection experiments concentrated on obtaining recovery-time data used to construct a recovery distribution from which certain parameters are required inputs for reliability assessment. Also performed were experiments designed to measure or estimate fault latency and fault-free performance.

Early fault-injection experiments were straightforward: a pin was selected at random from a chip selected at random from a system board also selected at random. Other methods of selecting the fault-injection location, called sampling methods in reference 3, had been devised and used to design experiments. These methods of selecting a location for fault injection assured compliance with statistical theory and thus accuracy in parameter estimation. The data acquisition requirements for these experiments were modest and were satisfied with the existing data acquisition system. (See refs. 3 and 4.)

Selection of injection points at random over the whole system misses many weak points in the system that can be identified by careful examination and testing of the design. The common perception is that in order to explore the system for the existence of abnormal recovery behavior and single-point failures, exhaustive or "massive" fault-injection testing must be performed.

A new methodology for fault-injection experiments is proposed in this paper that enhances the probability of detecting abnormalities and inadequacies in a system. The results presented in this paper were obtained using the new methodology. These results support the contention that the new method enhances the probability of detecting inadequacies in fault-tolerant systems.

The new methodology is based on two principles:

1. Faults should be injected mainly in signals with high fan-outs (e.g., board enables and other control signals), thus maximizing the amount of damage (the number of errors introduced) to the system.
2. The information/data necessary to determine the system state should be observable at all times in real time.

These principles, although seemingly simple, require a completely new experimental environment. The data acquisition requirements imposed by the new methodology on the FTMP test environment overburdened the original data acquisition system. This system was sufficient for the early work but completely inadequate for the new methodology and necessitated the design of a new system with three orders of magnitude improvement in data acquisition rates. (See ref. 4.) The application of the new methodology to fault-injection experiments uncovered an abnormal fault-recovery behavior never observed before.

During some of the initial experiments using the new methodology, it was noted that line replaceable units (LRU's) not being faulted were regularly disabled by the fault-management software. Although the system classified the events as being caused by real permanent faults that occurred during the fault-injection experiment, no evidence of these faults could be found after the fault-injection experiment was completed. All the supposedly faulty units were reactivated either manually or by booting the system. Therefore, it was concluded that these events must somehow be caused by the faults injected during the experiments.

Experiments using the new data acquisition system were designed to characterize this behavior. Initially, the error-latch processing integrity was investigated. The data indicated that on certain occasions only one LRU reported a specific bus error. All the LRU's that singly reported an error on an active bus on two separate occasions were disabled. The authenticity of the reported error was investigated to determine if the behavior was caused by the injected fault affecting an LRU across fault-containment boundaries or by corruption of the error-latch information by software or noise. The new data acquisition system was instrumental in determining the authenticity of the observed error as it is described in a later section. Once the authenticity of the reported error was established, the mechanism was investigated by which the fault coerced a nonfaulted LRU to report an error not visible to others.

Once the cause of the behavior was discovered and characterized, an analytical model was developed using easily measurable parameters to estimate the probability that an abnormal recovery would occur during the process of recovering from a naturally occurring fault. Also, the model was used to estimate the probability of a general triple-modular redundant system suffering a recovery failure.

Background of Fault-Tolerant Multiprocessor

Figure 1 shows the physical configuration of the Langley Avionics Integration Research Laboratory (AIRLAB) FTMP test-bed. (For a detailed description see ref. 1.) There are 11 LRU's (12 is the maximum number of LRU's supported by the FTMP) that are interconnected by a redundant system bus. Each LRU contains a processor with cache, a system bus interface, and a system memory unit. (There are many other subsystems that are not relevant to the subject of this paper and are therefore not mentioned.)

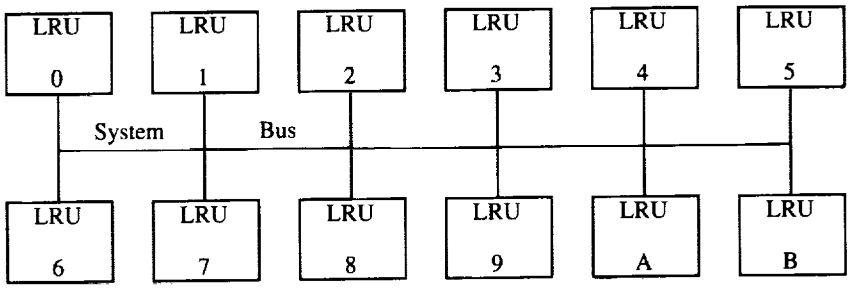


Figure 1. FTMP physical configuration. (LRU B is absent from the configuration.)

At system restart, the processors and the memories are organized in triads (see fig. 2) to provide the redundancy required to tolerate a single fault anywhere in the system. The triads are tightly synchronized; i.e., all the processor (memory) components of a processor (memory) triad execute the same instruction (operation) on the same clock cycle. Thus synchronized, three copies of all input/output (I/O) data from a triad are presented to the triple redundant system bus. The copies are then voted bit by bit to mask any errors that might occur because of a fault on any of the triad components.

The system bus is a composite of four different redundant serial buses. These buses are the Poll bus (P bus), the Transmit bus (T bus), the Receive bus (R bus), and the Clock bus (C bus). There are five of each P, T, R, and C bus types, of which three are used at any given time (a bus triad). One exception, the real-time clocks and the C bus are configured as quads (four units active) to implement the fault-tolerant clock. (See ref. 1 for details.)

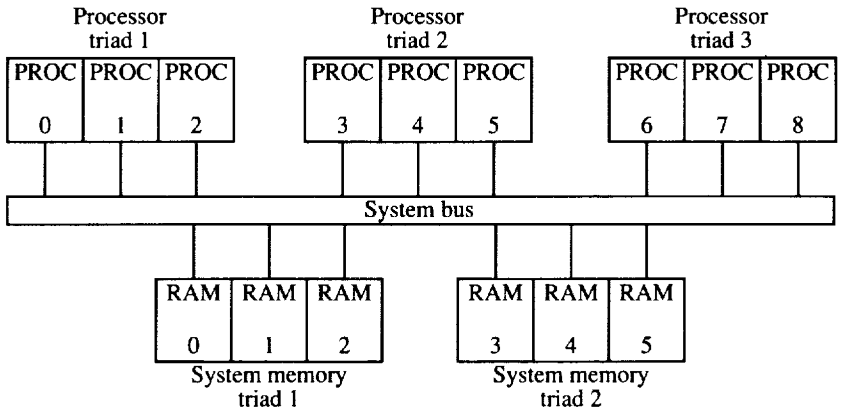


Figure 2. FTMP architecture.

The P bus is used by a processor triad to request control of the T and R buses. The T bus is used by a processor triad to transmit read/write commands to system memory triads. The R bus is used by memory triads to respond to processor triad read commands transmitted through the T bus triad. (See ref. 1.) In a typical example, when a processor triad needs to input or output a data value to a memory triad, it will first poll for control of the system bus through the P bus. Each element of the processor triad polls in synchronism with the other elements on one of the active P buses. (As mentioned above, there are three active elements at any time.) The polling protocol is called the Lanning protocol and is described in reference 1. After obtaining control of the system bus, the processor triad issues read or write commands (three copies) through the active T buses. If the command is a read command, then the responses (three copies) from the addressed memory triad appear in the active R buses after a short delay. There is a second type of read operation, called a simplex read, in which there is only one response transmitted through one of the R buses. The simplex read is required to read nonredundant data sources, e.g., an error latch.

The three active buses of each type are compared, any single error is masked, and the detection is flagged by setting a bit in an error latch. Each LRU contains four error latches, one per bus type (i.e., P, T, R, and C error latches). Each error latch contains five bits, one for each available bus of the specified type; e.g., the P latch has five bits, one for each of the five P buses (P1, P2, P3, P4, and P5) available to the system. Setting the bit corresponding to the P1 bus, for example, signifies that an error was detected in that bus during a previous transaction.

The P bus is monitored continuously by all the available LRU's; therefore, if a processor fails to correctly follow the polling protocol, then every LRU will flag an error in the P bus used by the misbehaving processor. Error handling is different in the T and R buses. Although all the T and R bus operations are voted by all the LRU's, only the LRU's that initiated the transactions (by having their processor units issue a read or a write command) will log error detection in the T or R error latches. This is required so that when a simplex read transaction is performed, the other LRU's not involved in the transaction (which had not been signaled that a simplex transaction was requested) do not mistake the simplex data for bus errors.

On the test-bed system one processor (processor A) in FTMP is used as a "master" processor during system restart. This master processor is used to load system memory with the application software and to issue initial configuration commands, e.g., which processors are going to be in triads 1, 2, and 3. After system restart, the master processor is used for data acquisition and software debugging purposes so that it is never part of a triad or used as a spare. The system memory unit of LRU A is used as a spare for a system memory triad.

Error-Latch Processing and Fault-Injection Experiments

Pin-level fault-injection experiments have been performed for the past several years on the FTMP test-bed. (See refs. 2 and 3.) In these experiments, hardware fault-injection systems use different methods (see refs. 2 and 5) to induce a temporary malfunction in one or more components of the target system. The errors resulting from the induced fault propagate through the circuits, thus producing erroneous behavior at the system bus level. Manifestation of the errors at this level occurs by several "fault-to-system bus error" paths operating singly or simultaneously. For example, a fault can corrupt output data on cache or disable the system bus interface, or both; in such a case, the external behavior of the faulted unit will differ from the "normal," as determined by voting the data presented to the system bus by the two nonfaulted triad partners.

Error-Latch Processing

Errors at the system bus level are flagged by setting the error latches. The latches are read into system memory and processed every 320 msec by the fault-management software. (See ref. 6 and table 1.) Error latches are not redundant; each LRU contains one of every kind. Therefore, error-latch values are read with a "simplex read operation"; i.e., there is only one copy transmitted through one of the active R buses. During a read operation the 5 error-latch bits are transmitted as the least significant bits of a 16-bit word, and the 11 most significant bits are all transmitted as 1's. Therefore, an error latch with no error signaled will appear as $FFE0_{16}$.

The data shown in table 1, acquired with the high-speed data acquisition system (ref. 4), are a typical example of what occurs during a fault injection. The data indicate that all the LRU's detected a bus error in the P1 bus. (LRU 3 was enabled on this bus.) However, the data also show that LRU's 4 and 5 detected a bus error in the T1 bus. (LRU 3 was enabled to transmit in that bus.) The faulted behavior of LRU 3 (for the same fault used to gather the error-latch data) was studied with the high-speed data acquisition system in order to deduce the sequence of events described below. The observed behavior of LRU 3 under the specific fault was the complete absence of bus transactions; i.e., the fault completely disabled LRU 3 transmissions on the system bus. Therefore, it follows from the error-latch data in table 1 that the processor triad composed of LRU's 3, 4, and 5 issued a read/write request through the T bus after performing a polling sequence to acquire control of the bus. The polling sequence was not executed by LRU 3, which led to the detection of a bus error in the P1 bus. Even with LRU 3 out, the two surviving partners of the triad gained control of the system bus and performed a read/write transaction on the T bus. Again, on this transaction LRU 3 was out, which led to the detection of a bus error on T1. This error should have been detected by LRU's 3, 4, and 5 (T and R bus errors are detected only by the triad that generates the transaction where the error occurs), but LRU 3 error-latch values were discarded by the system because the system does not utilize the latch values from units suspected of being faulty.

Table 1. Bus Errors Detected by LRU

	Raw error-latch data				Interpreted data
					Buses where error
LRU	P.EL[a]	R.EL[b]	T.EL[c]	C.EL[d]	was detected
0	FFE1[e]	FFE0	FFE0	FFE0	P1
1	FFE1	FFE0	FFE0	FFE0	P1
2	FFE1	FFE0	FFE0	FFE0	P1
3	FFE1	FFE0	FFE0	FFE0	P1
4	FFE1	FFE0	FFE1	FFE0	P1, T1
5	FFE1	FFE0	FFE1	FFE0	P1, T1
6	FFE1	FFE0	FFE0	FFE0	P1
7	FFE1	FFE0	FFE0	FFE0	P1
8	FFE1	FFE0	FFE0	FFE0	P1
[f]9					
A	FFE1	FFE0	FFE0	FFE0	P1
[f]B					

[a]P bus error latch.
[b]R bus error latch.
[c]T bus error latch.
[d]C bus error latch.
[e]All error-latch values in hex.
[f]Not in present configuration.

As a result of the simplex nature of the error-latch data, several tests are performed on the error data in system memory before it is considered by the fault-management software. These tests are designed to filter out corrupted data. First, any "zeroed" bit found in the 11 most significant bits of each value indicates that the value was corrupted during the read operation. Second, if a value showed errors in an inactive/spare bus, that value is considered corrupted. Third and last, if the five least significant bits contained more than one bit set to 1 (errors on multiple buses), the value is also assumed corrupted. The R bus from which the corrupted error-latch values were read is then marked suspect and any other values read through it are discarded.

A latch value that passes all three tests is considered good. All the good values are then checked for consistency; e.g., all the latches indicating an error in an active P bus must agree on the identity of the accused P bus (say P1), whereas all the latches indicating an error on the T bus must agree on the identity of the accused T bus (say T3). If there is consistency among values and one of the accused buses is an R bus, this bus is marked as suspect and all the error-latch values read through it are discarded. If they are not consistent, those data are excluded from further consideration.

The identity of all the buses with errors, as indicated by all the latch values that survived the consistency check, is passed to the fault-management software for isolation of the faulty component(s) (which could be the bus or any processor or memory unit enabled on the suspect bus).

A special case occurs when only one good error latch indicates an error on an active bus. This situation is a special case of a Byzantine fault, and it is called the lying-fault syndrome in reference 6. In such a case the identity of the LRU from which the error latch was read is stored in system memory. If the situation repeats with the same LRU, the LRU is disabled. (Both the processor and the memory unit of that LRU are disabled.)

Fault-Injection Experiments

A typical fault-injection experiment involves the selection of the board, chip, and pin on the FTMP LRU 3 where faults of a particular fault type (e.g., stuck at one) would be inserted at random times. The number of faults inserted can be varied and it normally ranged from 500 to 5000 faults. The fault detection, fault isolation, and reconfiguration times, plus the identity of the disabled unit, were acquired for every fault and were stored in computer disk files.

The instrumentation required by the new fault-injection methodology allows investigators to capture the behavior of the FTMP in real time. Two locations in LRU 3 were selected according to the new methodology criteria: (1) a control signal in the central processing unit (CPU) data path, and (2) a control signal in the cache controller. Fault injections in these locations have the desired effect in that abnormal recovery behavior was observed every 500 to 1000 faults.

Anomalous Behavior: Observations and Preliminary Data

As mentioned above, during the analysis of the fault-injection data it was noted that the LRU's not being faulted were regularly disabled by the fault-management software. Although the system classified the events as being caused by real permanent faults that occurred during the fault-injection experiment, no evidence of these faults could be found after the fault-injection experiment was completed. All the supposedly faulty units were reactivated either manually or by booting the system. Therefore, it was concluded that these events must be caused somehow by the faults injected during the experiments.

For experiments that lasted 4000 faults or longer, 4 nonfaulted LRU's were usually disabled before the system crashed. Obviously, whatever was causing the LRU's to fail did not disappear after the first LRU was disabled. Some discernible patterns in the data were observed. For example, in all the experiments where LRU 9 was not in the configuration (because of a real fault), LRU A was disabled first. In approximately 50 percent of the experiments, the second LRU to be disabled was LRU 5 followed by LRU 8. In the other 50 percent, LRU 8 was second followed by LRU 5. The fourth disabled LRU was LRU 7.

With LRU 9 present (after being repaired), usually five LRU's are disabled before the system crashes. The first LRU to be disabled alternated randomly between LRU's 5 and A. The second LRU to be disabled would be either LRU 5 or LRU A, depending on which was first. The next casualty would then be LRU 9, and thereafter LRU's 8 and 7 would follow.

To investigate the causes behind these failures, the high-speed data acquisition system was used to observe the error-latch values that were read, stored in system memory, and processed by the error-latch processing routines. By monitoring this activity during a fault-injection experiment, it is possible to observe the system behavior and detect any processing error or data corruption.

The acquired data indicated that there were occasions when only one LRU reported a specific bus error. All the LRU's that singly reported an error on an active bus on two separate occasions, triggering the lying-fault syndrome component of the fault-management software, were disabled. Why is one specific LRU the only one detecting an error? The answer to this question is of fundamental importance. It implies that weaknesses exist in the robustness of this fault-tolerant architecture and of fault-tolerant systems in general. This architecture was designed with the principles of fault-tolerant-systems design theory in mind. These principles state the number of fault-containment regions and how they are to communicate with each other in order to tolerate n faults, including Byzantine faults. For more details on fault-tolerant-systems theory and prototypes, see reference 7. The fact that single faults can cause the system to misbehave implies that the fault-tolerant design principles are incomplete and cannot by themselves produce a reliable or robust architecture.

First, the authenticity of the reported error must be considered; i.e., the reported error could be real (which means that the injected fault somehow prevented other LRU's from seeing it), or the error could be fictitious (which means that the injected fault somehow affected the error-latch values of another LRU across fault-containment regions). Corruption of the error-latch values could occur by several paths: (1) by erasing the latches, (2) by corrupting system memory, or (3) by noise coupling in the system bus. During fault injections in the FTMP, the corruption events must occur twice per LRU disabled (to trigger the lying-fault syndrome software). Therefore, any corruption of the error-latch values must happen periodically. Consequently, if error-latch data corruption is the cause of the behavior, an example should be readily caught.

The data acquisition system monitored the buses when the fault-management software read the error latches. The data thus obtained come directly out of the latches and into the system bus. Subsequently, the locations where these values are stored in system memory are monitored. The data from both sources can be compared to determine if any value has changed. In order to reduce the data that will be stored and analyzed, the acquisition system was programmed to sift through the acquired error-latch data and to store only those sets where bus errors are reported. These data clearly indicated that the error was authentic; i.e., latches were not being cleared by faulty software and values were not corrupted during storage in system memory or during transit in the system bus. The reported errors occurred on P and T buses; in particular, the P and T buses where the faulted LRU was enabled. (The system configuration data that include what LRU is enabled on each bus were also acquired in real time.) Lacking any confirmation on repetitive error-latch data corruption by faulty software or noise, the next step is to assume that the reported error is real.

The question of a fault in one LRU affecting another, although highly unlikely in this architecture, was investigated. The only way for signals from one LRU to reach another LRU is through the system bus. No other signal path exists except for the 28-V power supply that was checked for noise and/or glitches during nonfaulted and faulted operation, and nothing was found. Therefore, for a fault in one LRU to affect another, the fault-generated errors must propagate out from the faulted LRU through the system bus and into another LRU, defeating the voters that are always enabled. To defeat the voters the original error must affect at least one other bus by cross talk so that at least two of the three copies presented to the voters are in error. If the buses were that sensitive to cross talk, the effects would show during normal or nonfaulted operation and would have been easily detected. The fact that no such event was observed during the course of the experiments implies another cause for the observed behavior.

Causes of Anomalous Behavior

During the analysis of the acquired error-latch data, a pattern emerged that suggested an explanation of the anomalous behavior. The first hint was observed in the data sets containing the data present on the bus as the fault-management software read the error latches and stored the values in system memory. The observed interaction between the system architecture, the fault-management software, and the system bus is the clue to the problem. The fault-management software component that reads the error latches is structured as shown in figure 3.

A routine written in the Algol Extended for Design (AED) high-level language calls an assembler subroutine 12 times, once per LRU, to read the error latches of each LRU one by one into local cache. Then, the four error-latch values per LRU are sent to an array in system memory.

Reading the error latches seems like a simple task, except when the multiprocessing nature of the architecture and the operation of the system bus controller are taken into account. First, the system bus controller (SBC) is a direct memory access (DMA) device. To operate, the SBC must be supplied with the following: a destination address, a source address, a word count (number of words to transfer), and the type of transaction (normal read or write, or a simplex read). A DMA operation is very inefficient at transferring single words or multiple words at random noncontiguous locations. For a single-word transfer, 10 instructions must be executed to set the SBC to read the word to local cache and then to transmit the word to system memory. For multiple noncontiguous words, each word must be treated as a single-word transfer; i.e., the SBC must be set for every word transferred.

READ ERROR-LATCH PROCEDURE[a]

```
FOR I = 0 TO 12
CALL READEL(LRU.ID = I);
EXIT;
```

READEL PROCEDURE[b]

```
SET SBC[c] TO READ P.EL OF LRU I INTO CACHE
    READ P.EL
SET SBC TO READ R.EL OF LRU I INTO CACHE
    READ R.EL
SET SBC TO READ T.EL OF LRU I INTO CACHE
    READ T.EL
SET SBC TO READ C.EL OF LRU I INTO CACHE
    READ C.EL
WRITE P.EL, R.EL, T.EL, AND C.EL FROM CACHE
    INTO ARRAY IN SYSTEM MEMORY
EXIT
```

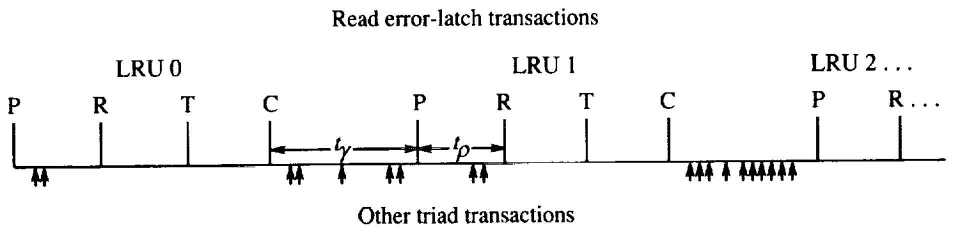
[a]The routine is originally coded in the AED language.
[b]The routine is originally coded in CAPS-6 assembler.
[c]System bus controller.

Figure 3. Structure of error-latch procedure.

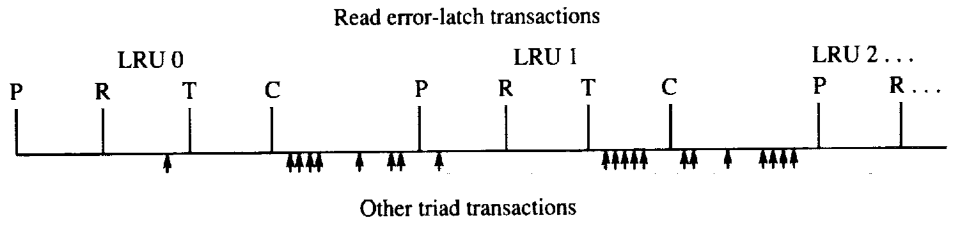
The SBC can be set to release or maintain control of the system bus between noncontiguous word transactions. If the bus is released after every word transfer, the SBC must execute the polling sequence to acquire control of the bus before transferring the next word. If the bus is not released, the poll sequence can be avoided and word transfers speeded up. However, there is a trade-off that must be investigated: if the number of noncontiguous words to be transferred is larger than a certain threshold, it is better to release the bus after each transaction. The limit in the length of a transaction while holding the bus is apparent when the multiprocessing character of the FTMP is taken into account; i.e., there are two more triads trying to execute other programs and system bus transactions. If one triad holds the bus too long, the other triads will have to wait to finish their tasks, thus increasing the probability of missing a deadline. Holding the bus could result in an increased probability of missing interrupts, decreased system throughput (two triads would not be able to communicate for that period), undersampling of sensors, and falling under the minimum permissible rate for updating actuators with dire consequences for the stability of the controlled system. Measurements obtained with the data acquisition system indicate that to read 48 error latches without releasing the bus would take approximately 3.2 msec.

Experimental Data

Releasing the bus after every transaction avoids the aforementioned problems but it increases the time required to read the error latches. The time increase has two components (ref. 4): (1) a poll must be executed for every error latch read (i.e., 48 latches $\times 15$ μsec/poll $\approx$ 0.7 msec), and (2) the triad must wait when the bus is busy (48 latches $\times$ 10 μsec/wait $\approx$ 0.5 msec). The expected time to read all the error latches becomes, then, $3.2 + 0.7 + 0.5 = 4.4$ msec. Measurements indicate that depending on the I/O traffic on the bus, the time to read the error latches (releasing the bus every time) varies from 4 to 6 msec. In figure 4, two examples of the bus activity during error-latch read transactions are shown.



(a) Example 1.



(b) Example 2.

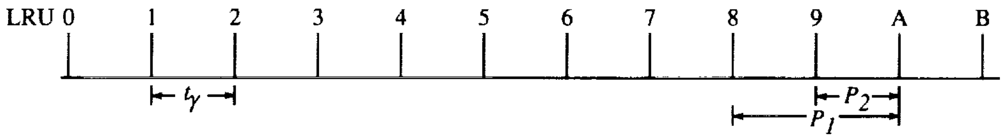
Figure 4. Two typical examples of system bus activity during error-latch read transactions.

The time between reading the C bus error latch of LRU n and reading the P bus error latch of LRU $n+1$ (this interval is denoted by t_γ) varies depending on the number of transactions by other triads (5 and 7 in the examples in fig. 4) from 277 to over 500 μsec. The time between reading error latches of the same LRU (this interval is denoted by t_ρ), e.g., reading P and R bus error latches of LRU 1, varies with the I/O traffic from 67 to 250 μsec.

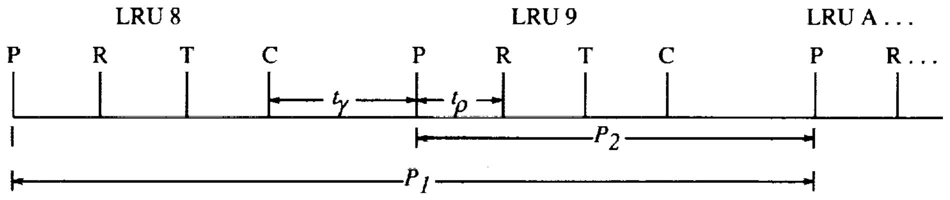
Approximately 45 percent of the triad transactions captured with the data acquisition system during the error-latch read transaction contained errors. The external manifestation of the fault was an inability of LRU 3 to transmit data on the P and T buses. The average error rate in both P and T buses as a consequence of the fault injected in LRU 3 was approximately one error every 210 μsec (4762 errors/sec). The error rate was approximately the same on both buses because every word transaction (about 99 percent of all transactions) involves a poll sequence. Therefore, when the triad with LRU 3 as a member starts a transaction, a bus error will be generated on both the P and T buses.

Subsystem Interaction Leading to Anomalous Behavior

The interaction of the error-latch read transactions, the time that the errors occur in the P and T buses, and the particular details of the system bus provide the framework on which the solution to the subsystem anomalous behavior can be found. In figure 5(a) the entire operation of reading the 12 LRU error latches is portrayed in a time-line diagram, and an expanded diagram of the events for LRU's 8, 9, and A is shown (fig. 5(b)). This figure will be used to explain the anomalous behavior of LRU A during fault injections.



(a) P, R, T, and C error-latch read transactions. (LRU B is absent.)



(b) Expanded diagrams. (P_n denotes interval where a transaction generates the first P bus error.)

Figure 5. Time diagram of error-latch read transactions of LRU A.

During experiments, faults are injected at random times during the software processing cycle. Therefore, the first system bus error generated by the faulty unit can occur anywhere during the cycle. Triads do not operate in synchronism with other triads; therefore, to one triad the I/O transactions of another triad will appear to occur at random times. Thus, from the point of view of other triads, subsequent errors generated by the faulty unit will appear to occur at random. The length of the slowest cycle in FTMP, in which the error latches are read and processed, is 320 msec. Therefore, for different fault-injection events, the first and subsequent errors generated by a fault occur at random times during this cycle.

In figure 5(b) two transaction time windows, P_1 and P_2, are shown. Both P_1 and P_2 represent a time interval where a transaction that generates the first P bus error in the cycle could occur. The P_1 window occurs when LRU 9 is not in the configuration. Under these conditions, a P bus error generated anywhere after the P bus error latch of LRU 8 is read, but before the P bus error latch of LRU A is read, would be signaled only by LRU A. Although the error is detected by all the LRU's, the software read the error-latch values of LRU's 0 through 8 before the error occurred. So when the P bus error latch of LRU A is read, it is the only one that appears to the software as having detected an error. The P_2 window occurs when LRU 9 is in the configuration.

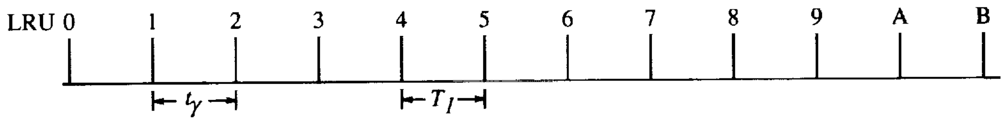
Using the average values for t_γ (304 μsec) and t_ρ (71 μsec), it is possible to estimate the probabilities that the first bus error in the cycle occurs in the intervals P_1 and P_2. The total time interval where a transaction can occur is 320 msec (1 cycle). Assuming that an error can occur at a random time within the cycle, then for P_1 the event probability P_e is the length of P_1 (i.e., $3t_\rho + t_\gamma + 3t_\rho + t_\gamma \approx 1.034$ msec) divided by the total time in a cycle (320 msec). Therefore, the estimated probability is $P_e \approx 1/309$. For P_2 the same analysis gives a probability $P_e = (3t_\rho + t_\gamma)/320$ msec $\approx 1/618$. The P_1 window is double the size of P_2; therefore when LRU 9 is absent, LRU A has double the chance of lying.

Using the estimated probabilities it is possible to estimate the average number of fault injections before an LRU is disabled by the lying-fault syndrome. The probability of an LRU lying per 320-msec cycle (T) is P_e. Two lying events are required for an LRU to be disabled. Therefore, the average number of cycles required before two events occur is given by $2/P_e$. The average fault is maintained for a period (F_t) of approximately 372.4 msec. Thus, the number of faults before an LRU is disabled is given by how many F_t's can fit into $(2/P_e)T$ (the average time to wait for two events). Thus, the average number of faults is given by

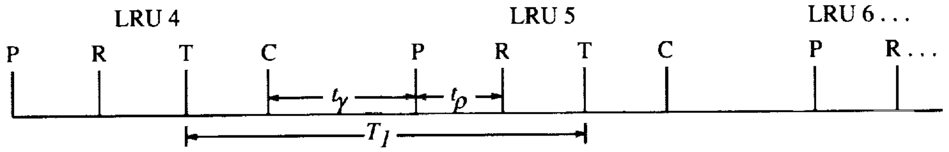
$$N_F = \frac{2T}{P_e F_t} \tag{1}$$

Without LRU 9, $P_e = 1/309$ for LRU A; therefore the expected number of faults injected before the LRU is disabled is $N_F \approx 534$. With LRU 9, $P_e = 1/618$ for LRU A; therefore $N_F \approx 1067$.

After LRU A is disabled, the same analysis applies for the active LRU whose error latches are read last by the fault-management software; e.g., if LRU A is the first LRU to be disabled and LRU 9 is not in the system, then this same analysis applies to LRU 8 (see fig. 5(a)) which results in a probability P_e of 1/618 ($N_F \approx 1067$) of LRU 8 being the only LRU signaling a P bus error. A similar analysis applies to LRU 5. Figure 6 shows the situation for this LRU. (Remember that T bus errors can be logged only by the members of the processor triad that initiated the bus transaction.)



(a) P, R, T, and C error-latch read transactions. (LRU B is absent.)



(b) Expanded diagram. (T_1 denotes interval where a transaction generates the first T bus error.)

Figure 6. Time diagram of error-latch read transactions of LRU 5.

During these experiments the processor triads were initially organized as follows:

(1) Triad 1 = LRU's 0, 1, and 2

(2) Triad 2 = LRU's 3, 4, and 5

(3) Triad 3 = LRU's 6, 7, and 8

The situation for LRU 5 is similar to that for LRU A. It is the last LRU in the error-latch read sequence that is capable of detecting a T bus error generated by its triad partner LRU 3. Therefore, if the first error-generating event for a cycle (T_1) occurs after the T bus latch of LRU 4 is read but before the T bus latch of LRU 5 is read, LRU 5 will appear to the software as the only LRU signaling a T bus error. The probability of this event can be estimated as before (i.e., $P_e = (3t_\rho + t_\gamma)/320$ msec $\approx 1/618$), and results seem to be the same as the P_2 event probability for LRU A. Here the similarities between LRU's 5 and A stop. When the triad containing the faulty unit reads the error latches (one-third of the time), the probability of a lying event is given by the probability of the first error occurring in the interval T_1. However, the only transaction this triad does on that interval is read the error latches from LRU 5. Unfortunately, when triads read simplex data they disable error detection and voting; therefore during this cycle no lying event will be generated. If any of the other two triads is reading the error latches (two-thirds of the time), the probability of a lying event is given by the probability that the triad with the faulted unit makes a transaction in T_1. Therefore, the total probability of a lying event for LRU 5 is $1/3(0) + 2/3(1/618) \approx 1/927$, which implies that $N_F = 1600$.

To account for the variability of the real process, the sample standard deviations of F_t (σ_F = 120.3 msec), t_γ (σ_γ = 53.5 μsec), and t_ρ (σ_ρ = 39.4 μsec) are used to estimate a first-order approximation to the variabilities of P_e and N_F. Equation (3) defines the standard deviation of P_e (eq. (2)) assuming that the random processes with standard deviations σ_ρ and σ_γ are independent. It should be noted that if

$$P_e = at_\rho + bt_\gamma$$

then

$$\sigma^2_{P_e} = a^2\sigma^2_\rho + b^2\sigma^2_\gamma$$

where a and b are arbitrary constants. The resulting equations are

$$P_e = \frac{L(3t_\rho + t_\gamma)}{T} \tag{2}$$

$$\sigma_{P_e} = \frac{L\sqrt{9\sigma^2_\rho + \sigma^2_\gamma}}{T} \tag{3}$$

$$N_F = \frac{2T}{F_tP_e + 2m\sigma_{P_e}F_t + 2n\sigma_FP_e + 4mn\sigma_F\sigma_{P_e}} \tag{4}$$

where

$$L = \begin{cases} 1 & \text{(for LRU A with LRU 9 present)} \\ 2 & \text{(for LRU A with LRU 9 absent)} \\ 2/3 & \text{(for LRU 5)} \end{cases}$$

and both m and n are ± 1. Equation (4) is obtained by expanding the expression

$$N_F = \frac{2T}{(F_t \pm 2\sigma_F)(P_e \pm 2\sigma_{P_e})} \tag{5}$$

Substituting the parameter values in the equations above makes it possible to estimate the 2σ bounds of N_F (table 2).

From table 2, the lower and upper 2σ bounds of N_F are

$$400 \le N_F \le 6000 \quad \text{(with LRU 9)}$$

$$200 \le N_F \le 3000 \quad \text{(without LRU 9)}$$

$$600 \le N_F \le 9000 \quad \text{(for LRU 5)}$$

The upper bounds obtained indicate a wide distribution and the possibility that the real distribution of these values might not be the normal distribution. Because of the relatively small data sample taken, the real distribution cannot be determined. Therefore, in order to correct the data for a skewed distribution, the upper bounds are taken as 2000 with LRU 9, 1000 without LRU 9, and 3000 for LRU 5. Assuming that the signs of the standard deviations are random and independent from the other parameters, these upper bounds will not be exceeded 75 percent of the time.

Table 2. Variability of N_F

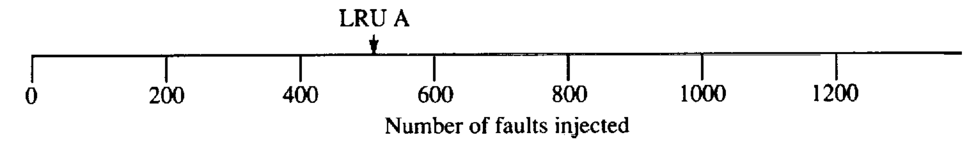
Signs	2σ bounds of N_F for—		
m, n	$P_e = 1/309$	$P_e = 1/618$	$P_e = 1/927$
−, −	3017	6034	9051
−, +	649	1297	1946
+, −	1000	2001	3002
+, +	215	430	645

Model Predictions

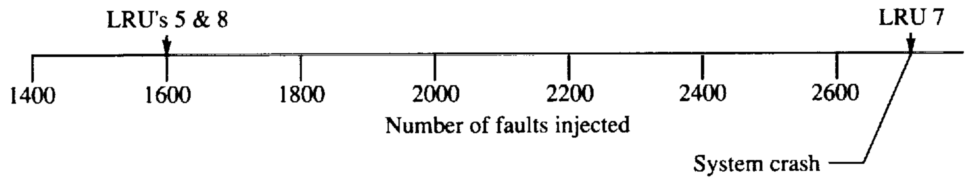
Using this model makes it possible to predict the expected sequence of LRU's that is disabled during a typical fault-injection experiment. In figure 7 the sequence of LRU's disabled is derived based only on two conditions: the absence of LRU 9 and the expected number of faults N_F. Using the average value of N_F to derive the sequence should lead to the most probable sequence. To derive the sequence it must be remembered that the LRU number is not important. The important information is the position that the LRU adopts in the sequence of error-latch read operations, i.e., the last active LRU in the sequence or the last LRU in the sequence that is a member of the triad containing the faulted processor.

In figure 7, LRU A (with an average N_F of 534) is disabled first, followed by LRU 5 (N_F = 1600). When LRU 5 fails, its triad partner, the faulted LRU 3, becomes a spare. The system software then assigns LRU 3 to a new triad (the triad composed of LRU's 0, 1, and 2) and swaps one of the processors of the triad (LRU 0) with LRU 3 so that fault injections can continue. On this triad (LRU's 1, 2, and 3), the last LRU capable of detecting T bus errors from LRU 3 is LRU 3. Therefore, in this case no lying can occur.

After LRU A is disabled, LRU 8 becomes the last active LRU in the error-latch read sequence (as did LRU A). Therefore, when LRU A is disabled, LRU 8 starts lying; and after the expected 1067 faults from this moment (1067 + 534 = 1601 faults from the beginning of the experiment), it is disabled. The expected number of faults for LRU's 5 and 8 is similar; therefore, it could be expected that the second and third LRU's to be disabled would flip between LRU's 5 and 8. The next LRU to be disabled (after 1067 + 1601 = 2668 faults) would be LRU 7 (because it is the last active LRU in the latch read sequence). After LRU 7 is disabled, LRU 6 should follow with 3735 faults injected, but this does not occur. The triad composed of LRU's 1, 2, and 3 is the last active triad in the system. (The rest of the nondisabled LRU's are spares.) This last triad is left with the faulted processor (LRU 3) and it is supposed to run all the system work load. Unfortunately, there are two problems that make this situation unworkable: first, the work load is too much for one triad to run on the allotted time, and second, the fault-management software is effectively disabled by this particular configuration.



(a) Sequence of LRU's disabled between 0 and 1200.



(b) Sequence of LRU's disabled between 1400 and 2600.

Figure 7. Theoretical sequence of LRU's disabled for a configuration without LRU 9.

The first problem (not enough time to run the work load) causes the system to slip frames so that all the high-priority tasks are executed first. Unfortunately, the fault-management software and the console-interface task are not among these. The low-priority tasks are run when there is time available (which does not occur very often). Consequently, the schedules for these tasks fall apart, taking 20 to 30 sec to execute each task. The second problem involves a check, performed in software, that prevents any triad from reconfiguring itself. Therefore, being the only active triad in the system, the triad of LRU's 1, 2, and 3 will not reconfigure under any circumstances, even when LRU 3 is faulted. Ultimately, the system does not have communication with the outside world (low priority), and the fault-injection software running on the minicomputer host indicates time-outs for fault detection, isolation, and reconfiguration. These symptoms are similar to a system crash and are the end result of almost all the experiments where 4000 or more faults are injected.

On very rare occasions, the system can really crash. These crashes, when they occur, always occur when there are only two triads left and a lying LRU is detected. Data obtained during these situations indicate that the system software cannot handle the situation where the faulted unit is isolated and reconfigured out in the same cycle where a lying unit has been detected. The system software, in its confusion, assigns the same R buses to two different processors on the same triad and tries to pull apart the two remaining triads (and it succeeds). The complexity and rarity of the events have prevented a complete solution on the case of the "confused software."

The model sequence of disabled LRU's (A $\rightarrow$ 5 (or 8) $\rightarrow$ 8 (or 5) $\rightarrow$ 7 $\rightarrow$ system failure) is the most commonly observed (approximately 99 percent of the experiments). Other less probable sequences have been observed, and all involve permutations of the A $\rightarrow$ 5 $\rightarrow$ 8 $\rightarrow$ 7 sequence. If the variability of N_F is included, these less probable sequences result from the overlapping of the N_F values for the different LRU's.

All these numbers are highly uncertain because of the probabilistic nature of these events, which is reflected in the variability of the values for the parameters $t_\gamma, t_\rho, \sigma_\gamma$, and σ_ρ and the limited size of the data sample. In table 3 comparisons between experimental data and the model results are shown. General agreement exists between the model and the experiment sequence of LRU's disabled and the number of faults injected before each LRU was disabled. In particular, when the variability of N_F was included (third column in table 3), all the LRU's were disabled in their predicted 2σ intervals 86 percent of the time.

Table 3. Comparison of Fault-Injection and Model Experiments

	Experiment	
LRU	Fault injection (*a*)	Model (*b*)
A	170 <444>[a] 1090	200 <534> 1000[b]
5	790 <1280> 2290	600 <1600> 3000
8	760 <1370> 1990	600 <1601> 3000
7	2500 <3670> 5100	1000 <2668> 5000

[a]Lower 2σ limit <average> upper 2σ limit.
[b]Numbers on right (1000, 3000, 3000, 5000) denote upper bounds computed using $m = 1$ and $n = -1$ in equation (4).

When LRU 9 is present the initial conditions change. Now, LRU's 5 and A have the same probability of being disabled; therefore there are two equally probable events competing for the first position in the sequence. The second LRU will depend on the first; i.e., if LRU A is first, then LRU 5 should be next and vice versa. The third LRU is LRU 9, independent of the first or the second LRU disabled. The reason for such behavior is apparent if the position of LRU 9 in the configuration is taken into consideration; i.e., LRU 9 is a spare before the first LRU is disabled. If the first LRU is 5, then LRU 9 takes 5's position in the triad (with LRU's 3 and 4). This makes LRU 9 sensitive to a lying event because it is the last LRU in the error-latch read sequence that can detect a T bus error from LRU 3. If the first LRU to be disabled is A (active on a memory triad), then LRU 9 is the last LRU in the error-latch read sequence, thus becoming as sensitive to a lying event (in the P bus) as LRU A was. After both LRU's 5 and A are disabled, LRU 9 occupies both sensitive positions for detecting P and T bus errors, thus doubling its probability of lying. After LRU 9 is disabled, the sequence of LRU's disabled is similar to the case when LRU 9 is absent, with LRU's 8 and 7 failing before the system fails. The result is not a sequence but an upside-down tree. (See fig. 8.)

Supporting evidence for this model of the observed behavior includes the fact that physically interchanging the LRU boxes did not affect the observed patterns as the model predicts. The LRU identification number refers to the slot, not the box; therefore by interchanging the identical LRU boxes, the possibility that a latent fault in one of the boxes is affecting the behavior of the system during the fault-injection experiment is eliminated.

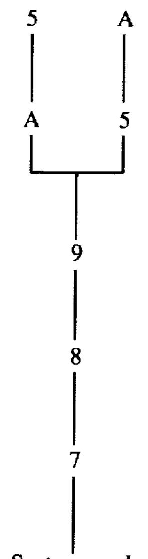


Figure 8. Sequence of LRU's disabled (with LRU 9 in the configuration).

Confirmation evidence for this model required the capture of one event where the error occurred as described previously. (See figs. 5 and 6 and text.) Unfortunately, the data acquisition system did not have enough memory to hold, or enough disk space to store, an entire fault-injection experiment. (The acquisition/storage rate is approximately 8MB/sec; one experiment of 2000 fault injections lasts about 4 hours.) Therefore, the data acquisition system was used to acquire a data sample and transfer it to the host minicomputer so that a program could examine it. If the sample contained an event of interest, the program would store the data on disk; otherwise it discarded the data and started the cycle again. After several months of searching, an event was found. (See fig. 9 and table 4.) A T bus error occurred after the LRU 4 and before the LRU 5 error latches were read. The content of the error latches indicated that only LRU 5 saw the event, as expected.

Solution

As mentioned previously, it seems possible to prescribe a quick solution to the problem. Keeping control of the bus while the triad reads the error-latch information should prevent other triads from performing transactions during this time, thus preventing an error from occurring between error-latch readings. Unfortunately, holding the bus does not solve the problem entirely; i.e., P_e does not become zero. Any error manifestation of a naturally occurring intermittent or transient fault in an element of the triad reading the error-latch values could propagate to the system bus at the appropriate time to cause the lying fault behavior. Such an event will depend on the error production behavior of the fault and is therefore difficult to reproduce through fault-injection experiments. A similar situation exists in fault-injection experiments: the faults are injected at random times that produce a nonzero probability that the fault will be injected at the appropriate moment when the faulted unit is a member of the triad reading the latches, thus causing the lying behavior. The faulted unit is reenabled after each fault, thus providing many trials on which a lying event can occur. Experiments were performed in which the system was modified to hold the bus during error-latch transactions and then was subjected to fault injections. These experiments confirmed that the probability P_e is not reduced to zero by holding the bus.

Table 4. Confirmation Event

	Raw error-latch data				Interpreted data
					Buses where error
LRU	P.EL[a]	R.EL[b]	T.EL[c]	C.EL[d]	was detected
0	FFE1[e]	FFE0	FFE0	FFE0	P1
1	FFE1	FFE0	FFE0	FFE0	P1
2	FFE1	FFE0	FFE0	FFE0	P1
3	FFE1	FFE0	FFE0	FFE0	P1
4	FFE1	FFE0	FFE0	FFE0	P1
5	FFE1	FFE0	FFE1	FFE0	P1, T1
6	FFE1	FFE0	FFE0	FFE0	P1
7	FFE1	FFE0	FFE0	FFE0	P1
8	FFE1	FFE0	FFE0	FFE0	P1
[f]9					
A	FFE1	FFE0	FFE0	FFE0	P1
[f]B					

[a]P bus error latch.
[b]R bus error latch.
[c]T bus error latch.
[d]C bus error latch.
[e]All error-latch values in hex.
[f]Not in present configuration.

Elimination of the behavior seems possible by the following procedure. After reading and analyzing the error latches, and after determining that there is a lying event where LRU x is accusing LRU y of producing an error, the error latches of all the LRU's read before reading the LRU x latches should be read again. With this procedure the situation explained in this section is avoided; i.e., if an error occurs just before the LRU x latches are read, then reading the latches of all the LRU's read previous to the error event would yield the missing data and eliminate the appearance of a lying event. Unfortunately, this procedure extends the error-latch processing time by a factor of 2, which causes frame slippage and other scheduling problems on the FTMP.

The most promising solution would seem to be redundant latches on the LRU's. Redundant latches would make all the present latch processing unnecessary and, coupled with the above procedure for dealing with a single LRU reporting an error, would eliminate the processing overhead and the lying event problems.

Implications for Fault-Tolerant Multiprocessor

This section considers the effects of real faults on FTMP behavior with the lying-fault process taken into account. The emphasis here is to develop a model that combines the error-production behavior of the faults and the system architecture into equations that can be used to predict the probability of observing this behavior for real faults. These equations can then be generalized and applied to more general architectures.

At the system bus level, a hard fault on the FTMP behaves as an intermittent fault; i.e., the fault-generated errors occur only when the faulty unit is on the bus. A hard fault is defined here as a fault that causes the affected LRU to generate an error every time it transmits on the bus. An intermittent or transient fault, in contrast, is defined here as a fault that causes the affected LRU to generate an error in a fraction of its transmissions on the bus. Therefore, in the FTMP the fault behaviors for hard, transient, and intermittent faults differ by the error production rate; i.e., hard faults produce errors at the same rate that the faulty unit performs I/O operations, whereas intermittent and transient faults produce errors at a fraction of the I/O rate.

Hard Faults

For a hard fault to cause a lying event, the assumption of errors occurring at random times during a cycle, and in particular during the error-latch read transactions, must apply. Clearly, this assumption is valid only in: (1) the first cycle when the fault occurs, and (2) subsequent cycles if the faulty unit is not an element of the triad reading the error latches. The reasoning behind the previous statement lies in the characteristics of a hard fault; i.e., a hard fault affects every operation of the faulty unit. If a fault occurs at any time during a cycle, it does not matter on which LRU it resides because the first error will occur at a random time in this cycle. For subsequent cycles, if the faulty unit is an element of the triad reading the error latches, its erroneous output will be detected from the first transaction (reading the P latch of LRU 0) by both its partners in the triad, thus preventing a lying-fault event. However, if the faulty unit is in another triad (not reading the error latches), the errors produced will appear random to the triad that is reading the latches.

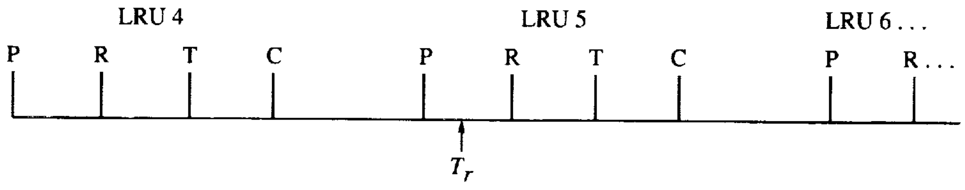


Figure 9. Confirmation event. (T_r denotes transaction that generates the first T bus error.)

For hard faults there is only one chance for lying events to occur: the events must occur before the faulty unit is isolated and reconfigured out. Hard faults could be isolated in 1 or more cycles. If it takes only 1 cycle to isolate the fault, the fault has only one chance of causing a lying event. If the faulty LRU is disabled during the first cycle, the error-producing hard fault is taken out of commission before it generates another lying event. If the system takes longer than 1 cycle, a chance exists that two lying events are generated in consecutive cycles. The parameters p_n (the probability that n cycles are required to recover from a fault) and N (the maximum number of cycles required to recover from a fault) can be estimated from fault-injection data for both hard and intermittent faults. The results are shown in table 5.

Table 5. Values for p_n and N Obtained From Fault-Injection Experiments

Number of cycles (n)	Parameter p_n for—	
	Hard faults	Intermittent faults[a]
1	0.3525	0
2	.6187	.1277
3	.0288	.8298
4	0	.0425
N. . . .	3	4

[a]For stuck-at-one faults injected at random times and lasting for 1 msec.

The rate at which a hard fault generates two lying events is given by

$$R = 3\lambda P_e^2 \left[2p_2 + p_3(1 - p_2 P_e)\right] \qquad (6)$$

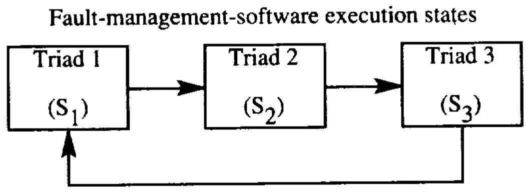
where

R	rate at which a lying LRU is disabled per hour
P_e	estimated probability of a lying LRU per cycle
λ	hard failure rate for an LRU
p_n	probability that n cycles are required to isolate the fault ($n = 2$ or 3 for hard faults)

To derive equation (6), the fact that the fault-management software moves from one triad to another on every cycle must be taken into account. (See fig. 10.)

In figure 10 the fault-management execution cycle is depicted. The states represent the location of the fault-management software at any given time. Through time the states form an infinite sequence. For analysis, a starting point can be selected as the current sequence when a fault first occurred; e.g., a sequence order could be $S_2 \rightarrow S_3 \rightarrow S_1 \rightarrow S_2 \rightarrow \ldots$

In the first cycle where the fault produces errors, the faulty LRU can be on any triad and the system can be in any state. Therefore, the probability of a lying event in this cycle is $9\lambda P_e$. The probability that a second cycle is needed to recover from the fault is p_2, and the faulty LRU has a probability of two-thirds of not being in the triad reading the error data. Therefore, the probability of a lying event in the second cycle is $(2/3)P_e p_2$. As shown in table 5 for hard faults, a maximum of 3 cycles could be needed to recover from the fault (with a probability of p_3). After 3 cycles the fault-management software has moved through the three triads. As a result, if the faulty unit and the fault-management task fall within the same triad during the second or third cycles, the hard fault would be detected before causing a lying fault (as discussed previously). Therefore, of the three state sequences that end in a 3-cycle recovery, two have a nonzero probability of causing a lying event. These results are derived below.



State	Triad 1	Triad 2	Triad 3
1	FMS	T/I	T/I
2	T/I	FMS	T/I
3	T/I	T/I	FMS

Figure 10. The three possible states of fault-management-software execution on the triads. (S_x denotes state x; FMS denotes fault-management software; T/I denotes task/idle.)

In this discussion assume that the fault occurred in a component of triad 2. In one sequence, $S_3 \rightarrow S_1 \rightarrow S_2$, the fault occurs in a cycle (S_3) where the faulty unit is not reading the error latches. In the second cycle (S_1), the faulty unit and the software again do not "meet" on the same triad. Therefore, during the second cycle, the probability of a lying event is $(1/3)P_e p_2$. The 1/3 factor is required to account for the fact that this is one of the three possible sequences of 3 cycles. For the sequence $S_1 \rightarrow S_2 \rightarrow S_3$, the probability of a lying event in the second cycle is zero because during this cycle the FMS and the faulty unit reside on the same triad. The second sequence of 3 cycles with a nonzero probability of a lying event is $S_2 \rightarrow S_3 \rightarrow S_1$, where the fault and the software meet on the same triad in the first cycle. The fault arrival is random, and therefore there is a probability P_e that it will arrive at the proper time to cause a lying event. For this sequence the probability of two lying events by the second or third cycle is $(1/3)[p_2 P_e + (1 - p_2 P_e)p_3 P_e]$. The 1/3 accounts for the probability that the fault arrives at S_2. The total rate then is

$$R = 9\lambda P_e \left[\frac{1}{3}p_2 P_e + \frac{1}{3}p_2 P_e + \frac{1}{3}(1 - p_2 P_e)p_3 P_e\right]$$

Grouping common terms gives

$$R = 9\lambda P_e \left\{\frac{1}{3}P_e[2p_2 + p_3(1 - p_2 P_e)]\right\}$$

and simplifying further gives

$$R = 3\lambda P_e^2[2p_2 + p_3(1 - p_2 P_e)]$$

which is equation (6). Although this equation is derived for a faulty member of triad 2, it is still a general result as can be demonstrated by deriving the equation for the other two triads.

After the lying LRU is disabled, the faulty LRU is most likely disabled; therefore R also gives the rate at which two LRU's are disabled, the lying and the faulty LRU's.

Computing R (using the values for a hard fault in table 5) and assuming a failure rate of $\lambda = 10^{-4}$ per hour (see ref. 8) and $P_e = 1/618$ gives $R \approx 10^{-9}$ per hour; i.e., the rate at which two LRU's are disabled (the first one due to a lying event and the second to the real hard fault) is approximately 10^{-9} per hour.

Intermittent Faults

The error-production behavior of intermittent faults is assumed to be random in time (although depending on where the fault is located and the utilization rate and input signals of the faulty hardware); therefore no consideration is necessary for the case where the faulty unit is part of the triad reading the error latches. The rate for two lying events is then the combinatorial probability of two events occurring in 4 cycles ($N = 4$ in table 5). The probability of a lying event in 1 cycle is different for intermittent faults (P_i) and hard faults (P_e). The difference lies in the fault behavior. If an intermittent fault generates faults at random times in a cycle, then P_i equals the probability of errors occurring at the appropriate intervals. The observed rate at which a good LRU would be disabled by intermittent faults is given by

$$R = 9\lambda \sum_{n=2}^{N} \left[p_n P_i^2 (1 - P_i)^{n-2} \frac{n!}{2!(n-2)!}\right] \quad (7)$$

For intermittent faults it usually takes more than 2 cycles to isolate the faulty unit, as shown in table 5. Computing R (using the value for P_i as an order of magnitude less than P_e for hard faults), using $\lambda = 10^{-3}$, and using the values given in table 5 for p_n for intermittent faults gives $R \approx 6.7 \times 10^{-10}$ per hour. For a rate of $R \leq 1 \times 10^{-10}$ per hour, the intermittent failure rate must be $\lambda \leq 1.5 \times 10^{-4}$ per hour.

For environment-related intermittent and transient errors, λ depends on the external environment where the system is located. Therefore, a value for the failure rate cannot be readily estimated from laboratory tests or general theory. Also, the p_n parameter depends on the displayed error behavior of the underlying error-producing phenomena, e.g., electromagnetic interference. Environment-caused errors do not stop after some units are disabled; therefore they can cause simple or multiple simultaneous errors and corruption of the simplex error-latch data until the system crashes.

Effect of Holding the Bus on the FTMP

The multiprocessing character of the FTMP provides enough redundancy for the FTMP to survive these events by degrading to two triads. Keeping control of the bus during the error-latch reading will not help avoid lying faults. By holding the system bus it is possible to keep triads from executing transactions when the error latches are read, but if an element of the triad performing the latch reading is affected by an intermittent fault, there is a chance for the fault to generate an error at the appropriate times to generate a lying event. Under the same assumptions as before, the rate of a lying event caused by an intermittent fault is given by

$$R = 3\lambda p_4 P_e^2 \tag{8}$$

Equation (8) is derived as follows: there are three active triads, and the rate that a fault occurs in any triad is 9λ. The probability that this triad is reading the error latches is one-third, at which time a lying event could be generated with probability P_e. After 4 cycles the faulty unit and the FMS come full-circle, at which time a second lying event could be generated with probability P_e. The probability that the fault lasts 4 cycles before the faulty unit is disabled is p_4. Therefore, the rate at which an intermittent fault generates two lying events is given by $R = (9\lambda)(1/3)(P_e)(p_4)(P_e)$, which becomes equation (8).

The value of P_e when the system holds the bus differs from the previous value used. Assuming that a random intermittent fault will produce errors at any random time within a cycle of length T, it is possible to approximate P_e as before; i.e., $P_e = \delta t/T$ where δt denotes the time window in which the system is susceptible to a lying event. While holding the bus there is no need to perform a bus polling sequence; therefore $t_\rho \approx (71 - 15) = 56$ μsec and $t_\gamma \approx (304 - 15) = 289$ μsec. These values for t_γ and t_ρ account for the time savings achieved by not performing a poll sequence ($\approx$ 15 μsec). The expression for P_e is

$$P_e = \frac{3t_\rho + t_\gamma}{T} \tag{9}$$

where $T = 320$ msec. Evaluating equation (9) gives $P_e = 1/700$. Evaluating equation (8) with this value of P_e, with $\lambda = 10^{-4}$ and with the appropriate values of p_n, results in $R = 2.6 \times 10^{-11}$ per hour as the rate of losing an LRU to a lying event.

Lying Events on a Triple Modular Redundancy System

For a fault-tolerant system based on adaptive-voting triple modular redundancy (TMR) principles (ref. 7), the probability of failure is greater or equal to the probability of losing a good LRU while there is a faulty LRU active. For this situation (after the good LRU is disabled, leaving the faulty one to produce more errors), the system has enough redundancy to detect errors, but not enough to mask them. Thus, it is assumed that the next error causes a system failure. The rate for a lying event in a TMR system is given by

$$R = 3\lambda \sum_{n=2}^{N} \left[p_n P_i^2 (1 - P_i)^{n-2} \frac{n!}{2!(n-2)!} \right] \tag{10}$$

Equation (10) is obtained from equation (7) by considering that in a TMR system there is only one triad that reads the error data. The system failure rate (R) versus the probability of a lying event (P_i) for a TMR system is shown in table 6 for different ranges of P_i and assuming that $\lambda = 1.0 \times 10^{-3}$. It is assumed here that the values for the probability of requiring n cycles to isolate and reconfigure from a fault (p_n), obtained from fault-injection experiments on the FTMP, apply to the hypothetical TMR system.

Table 6. Rate of Failure of TMR System (Lying Event)

P_i	R (from eq. (10)) (a)
1.0×10^{-5}	8.6×10^{-13}
5.0×10^{-5}	2.2×10^{-11}
1.0×10^{-4}	8.6×10^{-11}
5.0×10^{-4}	2.2×10^{-9}
1.0×10^{-3}	8.6×10^{-9}
5.0×10^{-3}	2.1×10^{-7}
1.0×10^{-2}	8.5×10^{-7}
5.0×10^{-2}	2.0×10^{-5}

[a] R and λ have the same units in equation (10).

Implications of New Technology

Although the large-scale integration of circuitry in a single chip makes pin-level hardware fault injection less effective, new technologies have evolved in this field. Utilizing lasers to induce a charge in a circuit node of a chip has been reported by many researchers at different laboratories. This method uses a laser of the appropriate wavelength to excite the valence electrons of the bulk semiconductor and electron donor impurities. The valence electrons absorb the laser energy and jump to the conduction band. The added electrons in the conduction band create a voltage in the node. This voltage perturbation is the desired effect in all hardware fault-injection techniques. Also, a new fault-injection technique solely based in software has been developed by Carnegie-Mellon University (ref. 9). All these techniques have limitations. It is up to the investigators to determine the most suitable technique to use for the system under test.

Concluding Remarks

A new fault-injection methodology was used successfully to investigate abnormal fault-recovery behavior of the Fault-Tolerant Multiprocessor (FTMP). With new instrumentation and fault-location selection criteria it was possible to observe and study the abnormal fault-recovery behavior of the FTMP. This behavior involved the reconfiguration of working units while the faulted unit was kept in the active configuration. The causes of the abnormal behavior were found and explained. Using a simple model of the behavior of the faulty system and data from fault-injection experiments to estimate the model parameters, the FTMP probability of losing two line replaceable units (LRU's) by a single fault was estimated. The model results indicate that the probability rate of losing a good unit to a lying fault is approximately 10^{-9} per hour.

Applying the model to a generic triplex system, in which a lying event represents a system failure, gives the system failure rate as approximately $10^{-9} + \delta$ per hour, where δ represents the contribution of any other failure mechanism to the failure rate. Therefore, the failure rate for the example triplex system is an order of magnitude greater than the desired rate for ultrareliable systems (10^{-10} per hour). This implies that a fault-tolerant system must either incorporate features to correctly handle this behavior or incorporate massive redundancy so that it can tolerate the loss of working units at the calculated rate.

It is evident from the discussion in this paper that the simplex sourcing of error-latch data is a weak link for systems depending on these data for fault recovery in a noisy environment or with design flaws. The unpredicted interaction between hardware, software, and faults can bring forth new, and most likely anomalous, behavior from the systems.

It has been shown that fault injection can help detect and analyze the behavior of a system in the ultrareliable regime. Although fault-injection testing cannot be exhaustive, it has been demonstrated here that it provides a unique capability to unmask problems in the system fault-management software and the software interaction with the system hardware. In view of the results obtained for the FTMP, it can be concluded that fault-injection experiments can provide very useful characterization data on the behavior of fault-tolerant systems.

NASA Langley Research Center
Hampton, VA 23665-5225
January 25, 1991

References

1. Smith, T. Basil, III; and Lala, Jaynarayan H.: *Development and Evaluation of a Fault-Tolerant Multiprocessor (FTMP) Computer. Volume I—FTMP Principles of Operation.* NASA CR-166071, 1983.
2. Lala, Jaynarayan H.; and Smith, T. Basil, III: *Development and Evaluation of a Fault-Tolerant Multiprocessor (FTMP) Computer. Volume III—FTMP Test and Evaluation.* NASA CR-166073, 1983.
3. Finelli, George B.: Characterization of Fault Recovery Through Fault Injection on FTMP. *IEEE Trans. Reliab.*, vol. R-36, no. 2, June 1987, pp. 164–170.
4. Padilla, Peter A.: *FTMP Data Acquisition Environment.* NASA TM-100636, 1988.
5. Padilla, Peter A.: *In-Circuit Fault Injector User's Guide.* NASA TM-100478, 1987.
6. Lala, Jaynarayan H.; and Smith, T. Basil, III: *Development and Evaluation of a Fault-Tolerant Multiprocessor (FTMP) Computer. Volume II—FTMP Software.* NASA CR-166072, 1983.
7. Siewiorek, Daniel P.; and Swarz, Robert S.: *The Theory and Practice of Reliable System Design.* Digital Press, c.1982.
8. Hopkins, Albert L., Jr.; Smith, T. Basil, III; and Lala, Jaynarayan H.: FTMP—A Highly Reliable Fault-Tolerant Multiprocessor for Aircraft. *Proc. IEEE,* vol. 66, no. 10, Oct. 1978, pp. 1221–1239.
9. Czeck, Edward W.; Siewiorek, Daniel P.; and Segall, Zary Z.: *Predeployment Validation of Fault-Tolerant Systems Through Software-Implemented Fault Insertion.* NASA CR-4244, 1989.

NASA
National Aeronautics and Space Administration

Report Documentation Page

1. Report No. NASA TM-4218	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle Abnormal Fault-Recovery Characteristics of the Fault-Tolerant Multiprocessor Uncovered Using a New Fault-Injection Methodology		5. Report Date March 1991
		6. Performing Organization Code
7. Author(s) Peter A. Padilla		8. Performing Organization Report No. L-16630
9. Performing Organization Name and Address NASA Langley Research Center Hampton, VA 23665-5225		10. Work Unit No. 506-46-21-05
		11. Contract or Grant No.
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, DC 20546-0001		13. Type of Report and Period Covered Technical Memorandum
		14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

An investigation was made in the Langley Avionics Integration Research Laboratory (AIRLAB) of the fault-handling performance of the Fault-Tolerant Multiprocessor (FTMP). Fault-handling errors detected during fault-injection experiments were characterized. In these fault-injection experiments, the FTMP disabled a working unit instead of the faulted unit once every 500 faults, on the average. System design weaknesses allow active faults to exercise a part of the fault-management software that handles Byzantine or "lying" faults. Byzantine faults behave in such a way that the faulted unit points to a working unit as the source of errors. The design problems involve: (1) the design and interface between the simplex error-detection hardware and the error-processing software, (2) the functional capabilities of the FTMP system bus, and (3) the communication requirements of a multiprocessor architecture. These weak areas in the FTMP design increase the probability that for any hardware fault, a good line replaceable unit (LRU) is mistakenly disabled by the fault-management software.

17. Key Words (Suggested by Authors(s)) Fault-tolerant multiprocessor Fault management Reconfiguration		18. Distribution Statement Unclassified—Unlimited Subject Category 33	
19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages 21	22. Price A03

NASA FORM 1626 OCT 86 NASA-Langley, 1991

For sale by the National Technical Information Service, Springfield, Virginia 22161-2171