NASA Contractor Report 165874

Fault Tolerant Software Modules for SIFT

Myron Hecht and Herbert Hecht

SoHaR, Inc.

Los Angeles, California 90035

NASA

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665

NASA Contractor Report 165874

Fault Tolerant Software Modules for SIFT

Myron Hecht and Herbert Hecht
SoHaR, Inc.
Los Angeles, California 90035

NASA

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

SECTION 1 - INTRODUCTION

This is the Final Engineering Report prepared for SRI International under Subcontract No. 14395 covering the implementation of software fault tolerance for critical modules of the SIFT operating software. The SIFT (Software Implemented Fault Tolerance) is an advanced computer concept developed by SRI for the NASA Langley Research Center under Contract NAS1-15428 to support the computational and reliability requirements of advanced fly-by-wire transport aircraft.

This report complies with the requirements of Article IV Item D of SRI Subcontract No. 14395.

Although this project constituted only a minor part of the SIFT effort, considerable advances in concepts and implementation of software fault tolerance were achieve under it. These are summarized in the paragraphs immediately following. Part 1.2 of the introduction provides an overview of the specific modules for which fault tolerant designs were generated, the error reporter and the global executive. Part 1.3 describes the organization of the body of this report, and Part 1.4 acknowledges the contribution of individuals outside our organization to this work.

1.1 ADVANCES IN SOFTWARE FAULT TOLERANCE IN THIS EFFORT

Because the software in the SIFT operating system is essential for both scheduling of application tasks and recovery from hardware failures, special efforts have been made to verify this software in a formal manner. In addition, it is being subjected to an extensive test program. Nevertheless, provision of fault tolerance features was deemed desirable for selected portions of these programs that have a key role in the recovery from failures. Note that this software must perform in accordance with its specification in the presence of faults in one or more of the component computers of SIFT or in their interconnections.

The fault tolerance technique selected for this purpose is that of the recovery block [RAND75]. Specific implementations of this technique to real-time applications and the transport aircraft environment had already been described prior to the effort reported here [HECH76, AERO78]. The basic structure for a recovery block is

<div align="center">

Ensure T

By P

Else by Q

Else Error

</div>

where T is an acceptance test condition, i. e. a condition which is expected to

be met by succesful execution of either the primary routine P or the alternate Q. The internal control of the recovery block transfers to Q if the test condition is not met by executing P.

The effectiveness of the fault tolerance provisons depends on the coverage of the acceptance test and the avoidance of correlated failure mechanisms in P and Q. Prior work had dealt primarily with software associated with a physical process (e. g., attitude control), where the environment could be depended on to furnish clues on the 'true' state of the process (e. g., by means of sensors independent of those that furnished the primary input data).

The uses served by the fault tolerant modules for SIFT are of an intrinsically logical nature, dealing with the reporting of errors and the action to be taken after positive reports. For applications of this type, the environment does not furnish independent clues, and the 'truth' has to be teased out of the logical process itself. Athough the routines to which fault tolerance was applied were quite small, the work was therefore quite challenging. The main contribution of the effort reported here to the field of fault tolerant software is the evolution of a technique for formulating acceptance tests in logic oriented applications based on conditions that are inherently orthogonal to the logic implemented by the primary routine. A very clear example of this technique is presented in the acceptance test for the error reporter in 2.2.

Further contributions will be found in the use of fault trees to identify the requirements for acceptance tests and to determine the completeness of the coverage of these tests. Some limitations of the recovery block technique were encountered in constructing alternate routines that are truly independent of the primary ones (and also of the acceptance test) for applications in which the principal operations are addition and subtraction (comparison). In all cases it was at least possible to change the order of operations and thereby to avoid common sequence dependent failures. Greater independence might be achievable by permitting alternate routines a larger scope (i. e., by letting one alternate routine perform the computations carried out in several primary routines). This concept seems worthy of exploration in future studies.

1.2. OVERVIEW OF THE FAULT TOLERANT ERROR REPORTER AND GLOBAL EXECUTIVE

SIFT achieves its high reliability by use of multiple processors with an excess of computing capacity. When a single processor fails, it is configured out of the system, a measure which ensures survival of the computer as a whole. Thus, an important function of the SIFT operating system is the retiring of faulty processors. A processor is defined as faulty if its output differs from those of other processors for a given task. The SIFT error reporter and global executive tasks collect information on disagreeing processors, process it, and designate processors for retirement to the reconfiguration task.

The error reporter analyzes error data collected by the voter to determine what processors appear to be faulty and indicates these in an error report. Because a processor can not report itself as faulty (even if the voter data would tend to indict it), error reports from each processor may differ. The global executive reviews all error reports, and if two or more processors point to a third as being faulty, then the result is transmitted to the reconfiguration

task.

The error reporter and global executive have been made fault tolerant by applying the recovery block principle described In section 1.1. Both tasks have an acceptance test and recovery block associated with them. Thus, there now exists a primary error reporter and global executive as well as alternates. Very few changes were necessary to the primary routines In order to Implement the recovery blocks, and, with the exception of the addition of a single Integer variable, no changes were made to the remainder of the system software.

As noted above, the error reporter acceptance test establishes that all processors with an excessive number of disagreements with the voter output are detected, and ensures that no properly functioning processors are designated as faulty. The alternate error reporter operates Independently of the primary routine, but produces an Identical output. The acceptance test Involves approximately twenty PASCAL statements, and the length of the alternate error reporter Is approximately the same as the primary. Thus, neither routine will have a significant effect on the timing of the SIFT operating system.

The global executive acceptance test Is coded In two modules: the first, which Is run before the primary routine, verifies that all Input to the global executive Is current, and the second, which Is run after the primary global executive, checks for correct execution. If errors are detected by either module of the acceptance test, the alternate global executive Is Invoked.

Execution of each of these routines Is checked by the other. Thus, the global executive checks on the execution of the error reporter acceptance test on each processor by means of the frame count encoded In the error words. Similarly, an output of the global executive which also has a frame count encoded within It Is checked by the error reporter In the subsequent frame. Notification to the system Is provided In the case of either error.

In addition to verifying correct execution of their Immediately associated primary routines, these acceptance tests can be expanded to give some Indication of the functioning of the reconfiguration task. If a processor Indicated as not working In the system status vector Is generating error reports, then obviously, It has not retired. Although diagnosis of the discrepancy Is beyond the scope of the tasks of the software developed here, an Indication Is made to the system that an off-normal condition exists, and appropriate action can be taken by the operating system.

A major portion of the coding effort went toward the validation of the five Pascal procedures developed as part of the error reporter and global executive recovery blocks. Driver routines with approximately 8 to 10 times the amount of code In these routines were developed In order to adequately support the large number of test cases which had to be run during validation.

1.3. ORGANIZATION OF THE REPORT

Section 2 describes the fault tolerant error reporter. Included are a description of the acceptance test, the error conditions which It covers, a description of the alternate routine, Implementation requirements for

Integration of the fault tolerant error reporter into the operating system, and a description of the software validation. Section 3, which describes the fault tolerant global executive, has a similar organization.

1.4 ACKNOWLEDGEMENTS

The voter routine of each processor in SIFT maintains its own record of the number of disagreements from the majority of all other processors. The SIFT error reporter marks processors as being faulty based on the disagreement count generated by the voter. The error reporter acceptance test compares the number of recorded processor disagreements with the output of the error reporter, and if processors are incorrectly characterized as working or failed, it invokes the alternate routine.

2.1. ERROR REPORTER ACCEPTANCE TEST

The SIFT voter routine marks individual processor disagreements from the majority in an array designated as errors. The error reporter sets a bit in a word called err for each processor with an excessive number of disagreements as reported in errors. Bits 0 through 7 in err represent the correspondingly numbered processors. The acceptance test checks that te error reporter was invoked in the previous subframe, and calls the alternate error reporter upon detection of a discrepancy between err and errors.

Figure 2.1 is a flow chart of the proposed error reporter acceptance test, and figure 2.2 is a Pascal listing of the procedure which has been developed and tested. The test counts the number of non-disagreeing processors in a counter designated as right and outvoted processors in a counter designated as wrong. It then checks the number of disagreements and the operational status of every processor designated as faulty. A Boolean variable to invoke the alternate error reporter is set to TRUE if a working processor marked as faulty has fewer than the threshold number of disagreements. The final segment adds right and wrong; if this sum does not equal the total number of processors, the acceptance test will invoke the alternate error reporter.

If the error reporter acceptance test does not detect any failures, it writes the frame count in the 8 most significant bits of err. When the global executive acceptance test checks these bits for the frame count, it will verify that the error reporter acceptance test has been executed in the current frame, and that consequently, err reports are current. If a discrepancy between the current frame and that encoded in the 8 most significant bits of err from a particular processor is encountered, the global executive sets a corresponding bit in an integer variable called mismatch along with the frame count in the 8 most significant bits. The error reporter acceptance test will then increment errors in the appropriate position in the subsequent frame. Thus, failure to execute the error reporter in the current frame will increase the likelihood that the processor will be retired by the alternate error reporter.

2.2. COVERAGE OF THE ERROR REPORTER ACCEPTANCE TEST

The error reporter acceptance test detects the following faults:
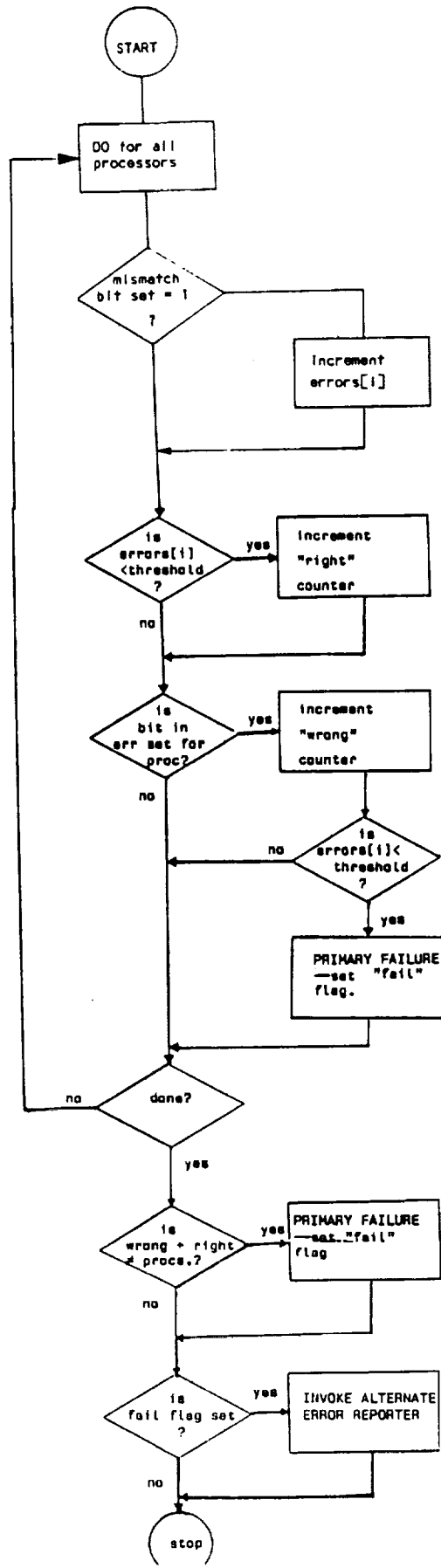
Figure 2.1. Flow chart of Error Reporter Acceptance Test

5a

```
21800    PROCEDURE ACCEPTANCE_TEST;
21900    (*error reporter acceptance test*)
22000    VAR
22100            EXCOUNT,WRONG,RIGHT,DIVISOR,CHECK,I,J,MISM:INTEGER;
22200            FAILFLG:BOOLEAN;
22300
22400    begin
22500
22600            excount:= mismatch div 256;
22650                            (*check execution count of global exec*)
22700            If (framecount mod 256)<>(excount - 1) then erfails:=true;
22750                            (*erfails is a global variable which notifies
22775                             the system that the global exec has not run*)
22800            mism:=mismatch div 256;
22900            wrong:=0;
23000            failflg:=false;
23100            right:=0;
23200            divisor:=1;
23300            for J:=0 to maxprocessors do    (*check for omission errors*)
23400                    begin
23430                    mism:=mism div divisor
23460                            (*processor has 1 strike against it if
23490                             error reporter didn't run in prev. frame*)
23495                    If odd (mism) then errors[J] :=errors[J] +1;
23500                    If (errors[J]<threshold) and (working[J])
23600                     then right:=right+1;
23700                                            (*count for omissions test*)
23800                    check:=err div divisor;
23900                                            (*shift err appropriate
24000                                             no. of places to the right*)
24100                    If odd(check) then begin
24200                            wrong:=wrong+1;   (*count for omissions test*)
24300                            If (errors[J]<threshold) and (working[J])
24400                            then failflg:=true (*check for false positives*)
24500                    end;
24600                    divisor:=divisor*2;
24700            end;
24800            If wrong+right<>maxprocessors +1 then failflg:=true;
24900                                    (*omissions test*)
25000            If failflg then alt_error_reporter
25100            else err:=err + 256 * (framecount mod 256);
25900    end;
*
```

Figure 2.2. Pascal listing of Error Reporter Acceptance Test

(1) failure to invoke the error reporter during each frame

(2) failure to report processors with an excessive number of
    disagreements as faulty to the global executive, and

(3) designation of a properly functioning processor as faulty

The validity of the input to the test (e.g. framecount, working, and errors) is
not checked, and it is possible that errors in these variables could be
propagated into err. However, to a certain extent, these failures are covered
by other processor error reports in the global executive.

The primary consideration in the design of this acceptance test was that the
verification and failure detection be performed in a manner independent of the
primary error reporter. the following subsections describe the means by which
the errors listed above are detected.

2.2.1. Failure to Execute During Each Frame

As noted above, the global executive acceptance test checks the frame count mod
256 encoded in the front part of each error report. Consequences of the
failure to execute the error reporter on a given processor are limited; a
consistent pattern of failures will be detected by means of the error reports of
other processors. Discrepancies will ultimately lead to the retiring of
processors which do not execute the error reporter. The present acceptance test
implementation calls for the retirement of the processor if any other
discrepancy from the system (i.e. voter) output occurs.

Just as the global executive checks execution of the error reporter, the
converse also occurs. If the frame count encoded in the front eight bits of
mismatch minus the frame count mod 256 is not equal to 1, then the global
executive acceptance test has not been executed in the previous frame, and the
system is notifed. Failure to execute the global executive may result in more
serious consequences than failure to execute the error reporter, and the "one
count against you" strategy described in the previous paragraph is not
appropriate.

2.2.2. Failure to Report Processors with an Excessive Number of
Disagreements to the Global Executive.

In order to achieve independence from the primary error reporter algorithm, the
acceptance test checks for this failure indirectly by testing for the following
conditions:

(a) the total number of processors reported as faulty is correct, and

(b) all processors designated as faulty have greater than the threshold
    number of disagreements

In this acceptance test, the number of processors with less than the threshold number of disagreements is counted in a variable designated as right, and the number having excess disagreements are counted on a second counter labeld wrong. If the sum of wrong and right is equal to the total number of processors, then the error reporter can be shown to have performed correctly when the third part of the acceptance test, described in the following section, has not detected any failures. This acceptance test is a particularly clear example of using algorithms which are orthogonal to the primary routine.

2.2.3. Designated a Properly Functioning Processor as Faulty

The final part of the acceptance test is to ensure that all processors designated as malfunctioning have at least the threshold number of disagreements. This determination is made by checking the number of disagreements of these processors. If any values of the array are below the threshold for working processors marked as faulty, then the primary error reporter has failed, and the alternate is invoked.

2.3. ALTERNATE ERROR REPORTER

Independence in the structure and operation from the primary error reporter was a chief objective in the alternate routine design. In addition, its output had to be compatible with the global executive.

These requirements resulted in a routine which is essentially the inverse of the primary error reporter. An alternate error word, designated as erra, is initially set ot all 1's; the alternate error reporter sets erra bits to 0 if the number of disagreements in the appropriate element of the errors array is less than the threshold. If there are more bits in erra than there are processors (e.g. if there are six processors and eight bits in erra), the leading bits are set to 0. Finally, the primary error word, err, is set equal to erra, loaded with frame count information, and placed in the pre-broadcast buffer. The complementary nature of this routine is maintained in the order of setting the error word bits -- the processors are checked in ascending order rather than the descending order used in the primary error report. Figure 2.4.1s a Pascal listing of the alternate error report.

2.4. IMPLEMENTATION REQUIREMENTS

As noted previously, the acceptance test and the alternate error reporter are short and relatively simple procedures which were written to be compatible with the SIFT operating system. Additional local variables are required as shown in the listings for the error reporter acceptance test and alternate routine. In addition, some modifications to the primary error reporter are necessary to enable it to transmit processor states to the global executive and execution information to the acceptance test. No changes in the broadcasting protocol are required.

start

set erra bits
to all 1's

DO for all
processors [in
ascending order]

errors[j]
<threshold
?

yes

Set appropriate
bit in erra to
0.

no

done?

no

yes

set leading bits
of erra to 0.

stop
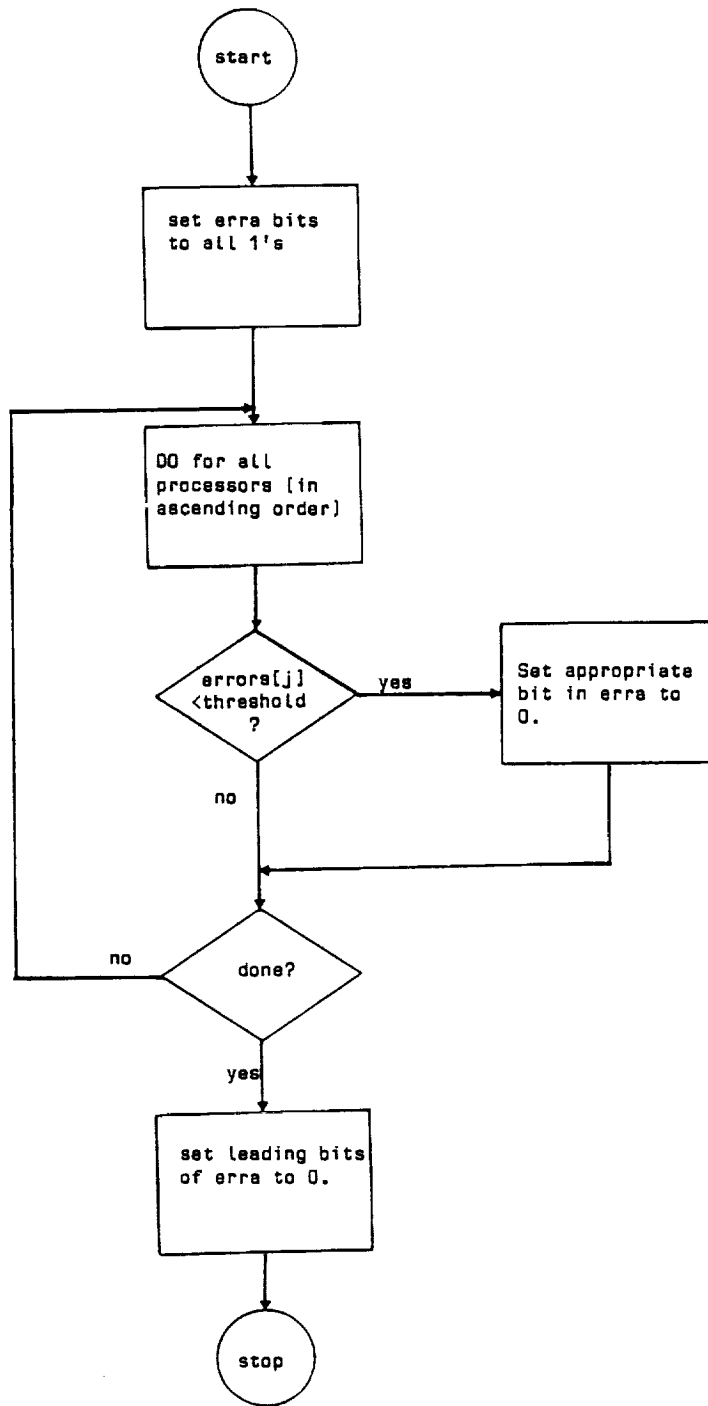
Figure 2.3. Flow chart of Alternate Error Reporter

9

```
19100    PROCEDURE ALT_ERROR_REPORTER;
19200    (*this is the alterate error reporter*)
19300
19400    CONST
19500            ALLONES=377B;
19600    VAR
19700            ERRA:INTEGER;   (*alternate error word*)
19800            I,K:INTEGER;
19900
20000    begin
20300            erra:=allones;
20400  ·        k:=1;
20500            for I:=0 to maxprocessors do
20600                    begin
20700                       if (errors[I]<threshold) and (working[I])
20800                        then erra:=erra-k;
20900                       k:=k*2;
21000            end;
21100            erra:=erra - (allones - k + 1);     (*remove leading bits*)
21300            err:=erra + 256*sfcount;
21400            prebroadcast(errerr,err);
21600    end;
*
```

Figure 2.4. Pascal listing of alternate error reporter

2.5. ERROR REPORTER RECOVERY BLOCK VALIDATION

The major objective of the testing performed on the error reporter recovery block was to provide a comprehensive set of cases which would demonstrate satisfactory performance when the error reporter was functioning properly and when it had failed. Figure 2.5 shows the top level fault tree that was used to define this set. The recovery block fails if the primary error reporter fails without detection by the acceptance test, or if the alternate fails after being invoked by the error reporter acceptance test. Failure due to an undetected primary routine fault will occur when both the primary routine fails and the acceptance test does not detect it. The same potential failures affect the acceptance test and the alternate routine and thus, they were both validated simultaneously.

Figure 2.6 continues the development. There are two major classes of errors: failure to identify a processor with excess disagreements, and reporting a processor with less than two disagreements in the error report. Under the first class of errors, one, two, or three processors could remain unidentified. Further expansion of the tree shows that failure to identify two outvoted processors is caused by failure to identify the first process and failure to identify the second. Similarly, failure to identify three processors having excess disagreements can be broken down into failure to identify the first processor and failure to identify the second and failure to identify the third.

Figure 2.7 continues this development. Any of the six processors could be identified as the first failure. Once the validation has established that the error reporter acceptance test and alternate can correctly identify the first error committed by the primary routine (i.e. failure to identify one processor with an excess number of disagreements), validation for the condition of two outvoted processors can be performed by holding fixed the first processor with excess disagreeements and only varying the second. Thus, processor 0 is assigned the first error, and processors 1 through five are each, in turn, given an excess number of disagreements in the errors array. Similar logic applies to the third and fourth processors with excess disagreements.

Figure 2.8 is a further development of the fault tree which summarizes the pattern in which the processors are tested. Transfer 1011 shows that all six processors are tested for the case in which the primary error reporter fails to detect one processor with excess disagreements. Transfer 1012 shows that when two disagree excessively, the primary error reporter is always assumed to have dectected an excess disagreement condition in processor 0, and that the acceptance test and alternate are tested with the second error in processors 1 through 5. For failure of the primary error reporter to detect a third excessively disagreeing processor, transfer 1013 shows that processors 0 and 1 are assumed to be the first two, and the third occurs in processor 2, 3, 4, or 5. Finally, for four errors, processors 0, 1, and 2 are assumed to have excess disagreements, and the final error varies between processors 3, 4, and 5.

The fault tree for the second class of errors, spurious identification of correctly functionining processors as having excessive disagreements is shown in figures 2.8 and 2.9. Incorrect identification of a processor as malfunctioning can occur when there are either no disagreements or a single disagreement.

11

Incorrect characterization of the processor can also occur when there are one, two, or three other processors which actually have excessively disagreed with the voter output. As previously, not all processors need to be considered. The testing scheme in this case is to ensure that the error reporter acceptance test can detect a false failure of each processor when any other processor has failed. Table 2.1 is a list of the validation tests required to verify the correctness of the error reporter acceptance test and alternate executive based on the fault trees described here.

Complete test required simulation of a major portion of the SIFT operating system. The simulation program, called DRIVER, prepares the errors and working arrays of the voter and err word of the error reporter based on external inputs. It next invokes the acceptance test, outputs results, and invokes the alternate if an error is detected. Appendix A shows a complete listing of the program.
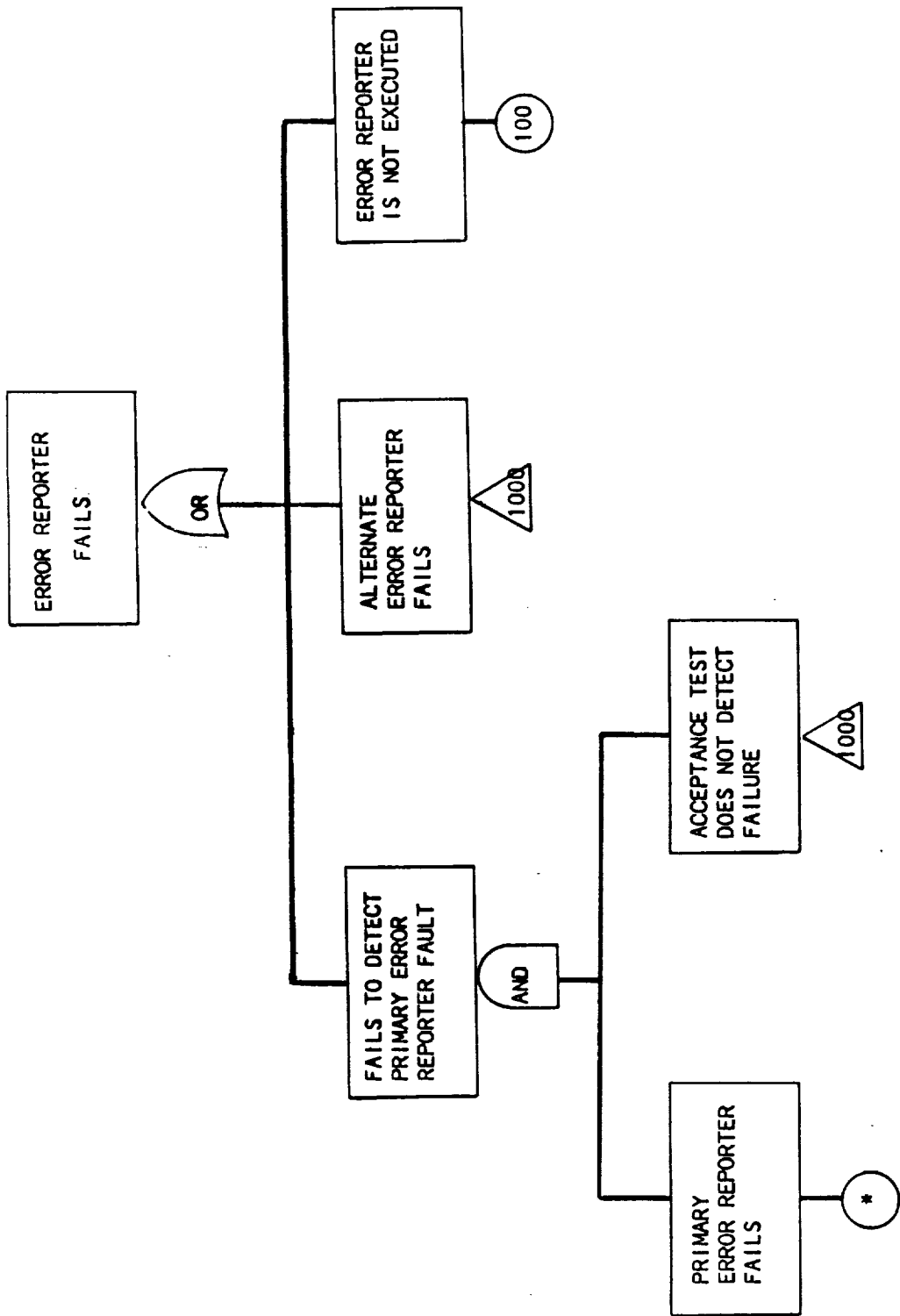
Figure 2.5. Top level tree for Error Reporter Failures
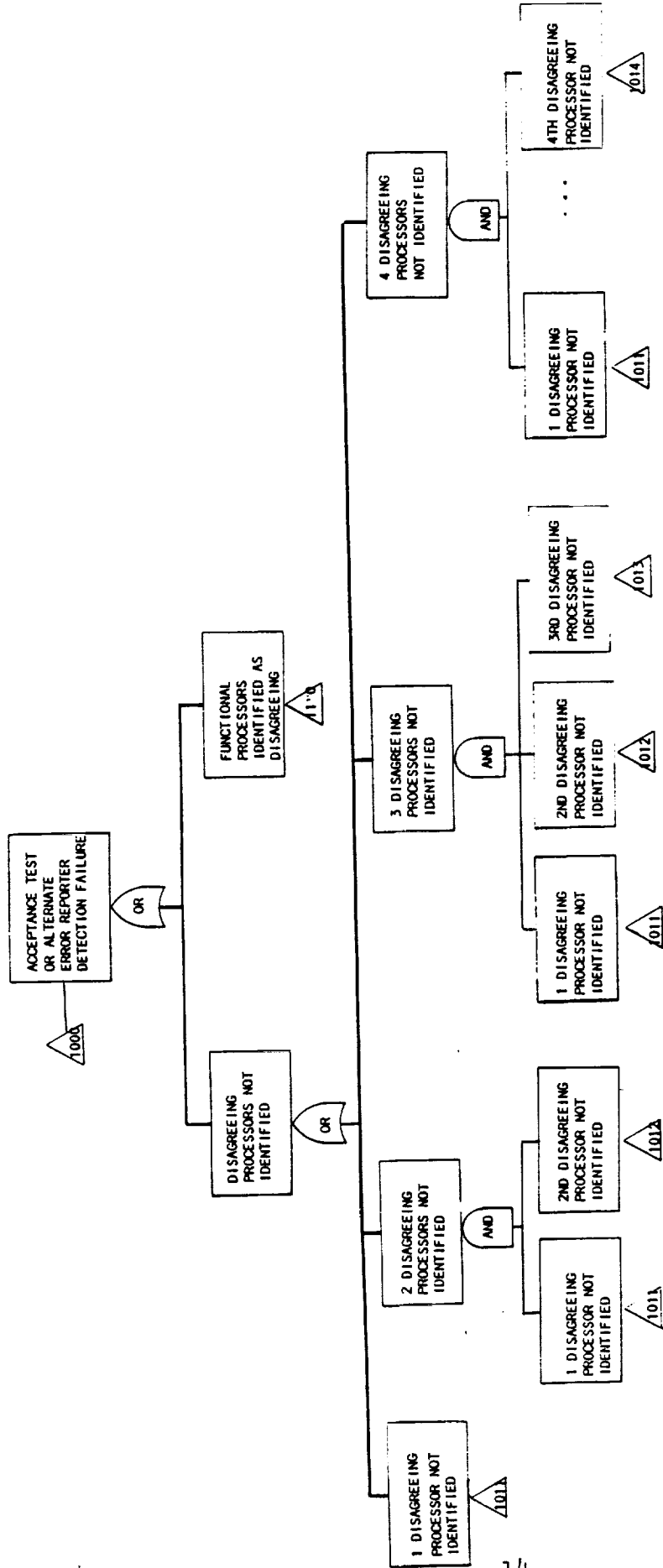
Figure 2.6. Classes of Error Reporter Failures

14
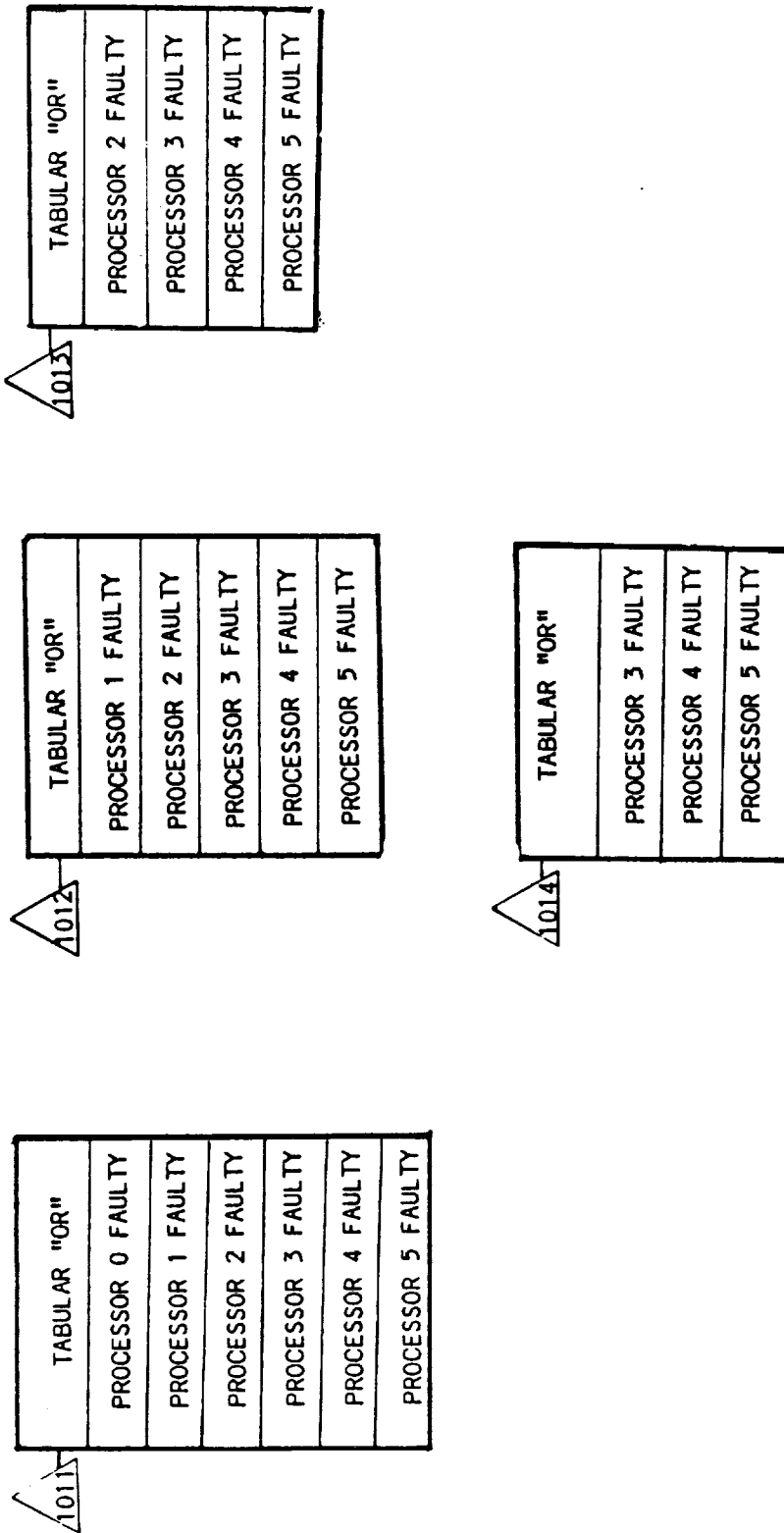
Figure 2.7. SIFT configurations used for detection failure validations

FUNCTIONAL PROCESSOR IDENTIFIED AS FAILED

1110

OR

PROCESSOR INCORRECTLY IDENTIFIED WHEN NO OTHERS ARE FAULTY

1110A

PROCESSOR INCORRECTLY IDENTIFIED WHEN 1 OTHER IS FAULTY

111

PROCESSOR INCORRECTLY IDENTIFIED WHEN 2 OTHERS ARE FAULTY

AND

PROCESSOR INCORRECTLY IDENTIFIED WHEN 1 OTHER HAS EXCESS DISAGREEMENTS

111

2ND PROCESSOR HAS EXCESS DISAGREEMENTS

1110B

PROCESSOR INCORRECTLY IDENTIFIED WHEN 3 OTHERS ARE FAULTY

AND

PROCESSOR INCORRECTLY IDENTIFIED WHEN 1 OTHER HAS EXCESS DISAGREEMENTS

111

3RD PROCESSOR HAS EXCESS DISAGREEMENTS

1110C

Figure 2.8. Incorrect Characterization of a Functional Processor as Failed

```
      /\
     /111\——— TABULAR "OR"
    /_____\
            | PROCESSOR 0
            | HAS EXCESS
            | DISAGREEMENTS
            |
            | PROCESSOR 1
            | HAS EXCESS
            | DISAGREEMENTS
            |
            | PROCESSOR 2
            | HAS EXCESS
            | DISAGREEMENTS
            |
            | PROCESSOR 3
            | HAS EXCESS
            | DISAGREEMENTS
            |
            | PROCESSOR 4
            | HAS EXCESS
            | DISAGREEMENTS
            |
            | PROCESSOR 5
            | HAS EXCESS
            | DISAGREEMENTS
```
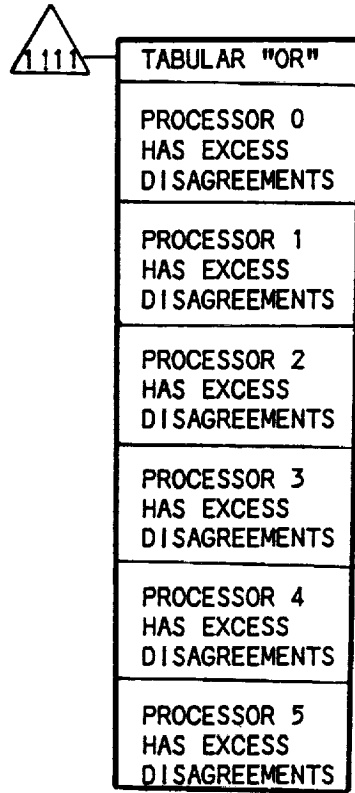
Figure 2.9. Final Development of Figure 2.8

TABLE 2.1. FAULTS FOR WHICH VALIDATION TESTING IS REQUIRED
FOR THE ERROR REPORTER ACCEPTANCE TEST AND ALTERNATE ERROR REPORTER

| Fault Tree Designation	Description
1011	Failure to detect primary error reporter's not identifying a single processor as having excess disagreements
1012	Failure to detect primary error reporter's not identifying a second processor as having excess disagreements given that the first has been identified
1013	Failure to detect primary error reporter's not identifying a third processor as having excess disagreements given that the first two have been identified
1014	Failure to detect primary error reporter's not identifying a fourth processor as having excess disagreements given that the first two have been identified.
1110A	Failure to detect primary error reporter's false identification of a functional processor as having excess disagreements given that no other processor has failed.
1110B	As 1110A, given that 1 other processor failed.
1110C	As 1110A, given that 2 other processors failed.
1110D	As 1110A, given that 3 other processors failed.
1111	Failure to detect primary error reporter's false identification of a functional processor as having excess disagreements given that one other processor has failed.

18

This section describes the acceptance test and alternate routine for the SIFT global executive. The acceptance test is coded in two modules: the first, which is run before the primary routine, verifies that all input to the global executive is current, and the second, which is run after the primary global executive, checks for correct execution. If execution errors are detected by either module of the acceptance test, the alternate global executive is invoked.

3.1. GLOBAL EXECUTIVE ACCEPTANCE TEST

The August, 1980 version of the SIFT operating system has the error reports for the active processors contained in the array prevote[errerr,*] where errerr is a constant set to 1. The error reports themselves are contained within the 8 least significant bits of each 16-bit element of prevote, and the frame count is encoded in the 8 most significant bits by means of the error reporter acceptance test. The global executive reads successive bits of each prevote element by shifting the word to the right. Because of this destructive read, it is necessary to reproduce the error report information prior to execution of the primary routine. This task is performed by the first module of the global executive acceptance test designated PREGEXEC. PREGEXEC also checks on the frame count which has been encoded by the error reporter acceptance test. After execution of the primary global executive, the second module of the global executive acceptance test, called GEXECTEST, is executed. GEXECTEST checks each position of each word in an order orthogonal to the primary global executive. It then compares this result with the appropriate bit in RECONF, the retirement word generated by the primary routine. If there is a discrepancy, the alternate global executive, ALTGEXEC, is called. ALTGEXEC is described in section 3.

Figures 3.1 and 3.2 are flow charts of the two modules of the global executive acceptance tests, and figure 3.3 contains the corresponding listings. The first module of the acceptance test, PREGEXEC, checks the framecount contained in the most significant 8 bits of each error report which have been written by the error reporter acceptance test, and then recopies the least significant half of the word into the most significant position in order to preserve them for the second module of the global executive acceptance test.

Those error words containing frame counts different from that of the system are set to zero as a means of masking them from the global executive, and a failure counter for the processor error report is incremented. The subsequent execution of the error reporter on other processors will count this indicator as a disagreement when writing their reports, an action which will result in retirement of this processor if at least one other discrepancy is detected.

Once the primary global executive has been run, the second module of the acceptance test checks the correctness of its execution, and invokes the alternate routine upon detection of an error. A major consideration in the design of the acceptance test was that it be independent of the primary routine. Thus, whereas the primary checks each position of an error report before moving on to the next, the acceptance test checks a given position of all error reports

19
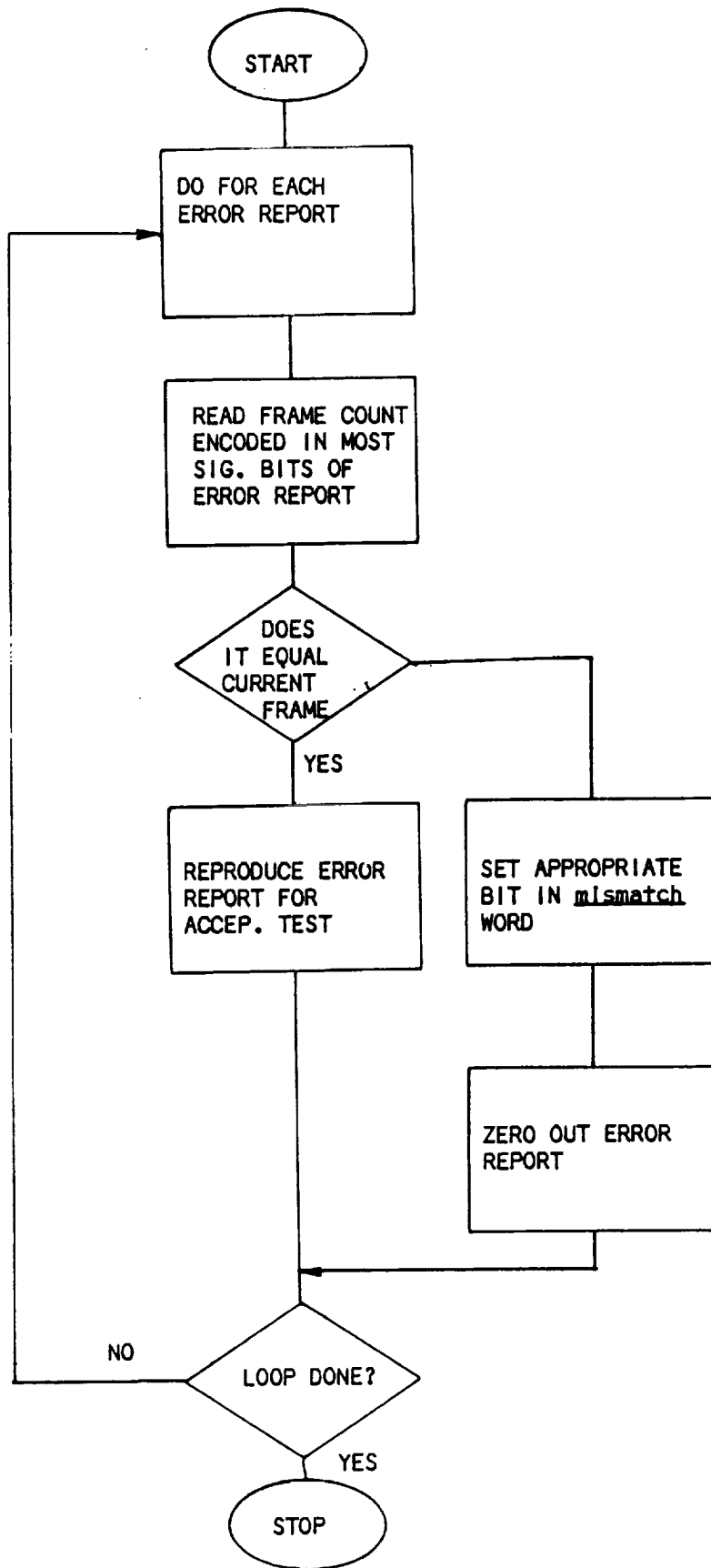
Figure 3.1. Flow chart for PREGEXEC

20

START

DO FOR EACH POSI-
TION OF ALL
ERROR REPORTS

DO FOR EACH
ERROR REPORT

NOT
WORKING OR
LOOKING AT
SELF?   YES

NO

ADD THIS
POSITION (1/0)
TO TOTAL.

RETIRED
PROCESSOR HAS   YES
ERR. RPT
?

NO

NOTIFY SYSTEM

LOOP DONE?

B

A
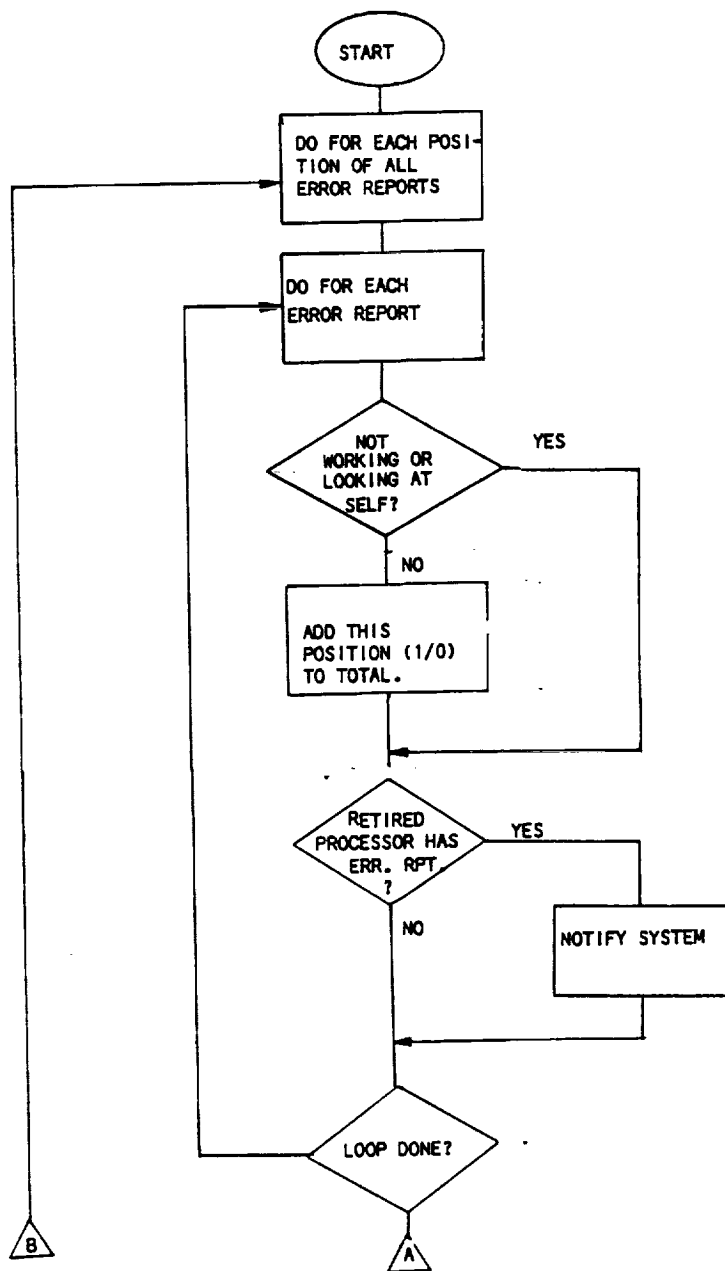
Figure 3.2. Flow chart of Global Executive Acceptance Test

21
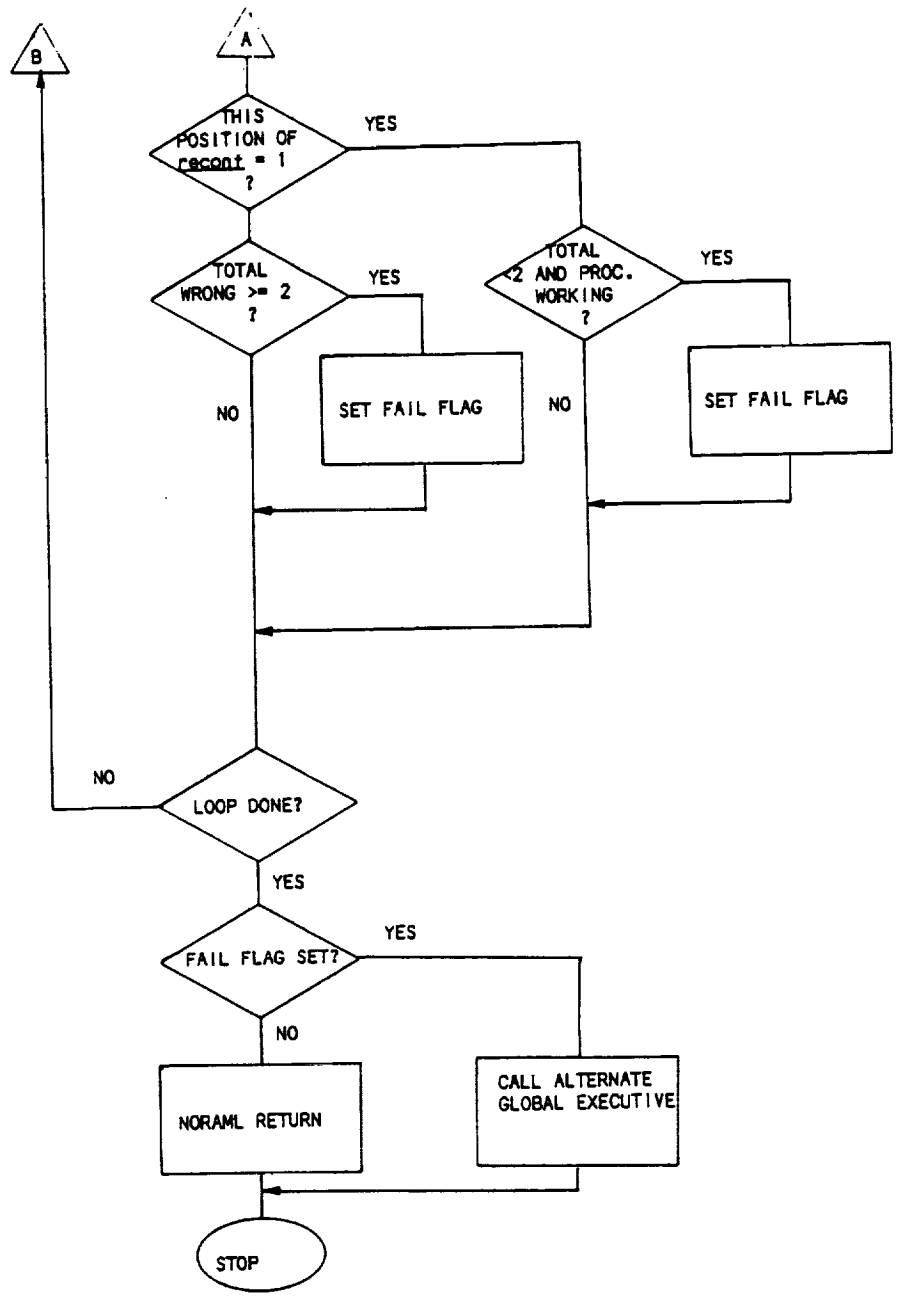
Figure 3.2 (continued). Flow chart of Global Executive Acceptance Test

before moving up to the next position. A second difference between the primary global executive and the acceptance test is the lack of an intermediate array (i.e. procs) for the storage of excess disagreements. Thus, once the number of discrepancies in a given position has been counted, it is immediately compared with the corresponding value in the reconfiguration word RECONF. If a discrepancy is detected, a flag is set that will result in the invocation of the alternate routine.

If a processor indicated as retired in the working array is indicated as having excess disagreements in the input processor error reports, then one of three conditions exists (1) a processor marked for retirement is still a functioning part of the system, (2) an error exists which affects the state of the working array, or (3) the error report(s) input to the global executive are not valid. Although the global executive can detect this discrepancy it cannot by itself isolate which of these three conditions caused the anomaly. As a result, the global executive acceptance test and alternate logic note the discrepancy to the system, but disregard the error reports in preparation of the reconf word.

3.2. COVERAGE OF THE GLOBAL EXECUTIVE ACCEPTANCE TEST

The global executive acceptance test described above detects the following faults:

   (1) failure to invoke the error reporter acceptance test

   (2) failure to retire processors reported by at least two other processors as having an excess number of disagreements with the voter result, and

   (3) marking for retirement processors which do not have an excess number of disagreements

Detection of the first fault occurs in the first module of the global executive acceptance test PREGEXEC. Two probable causes of the discrepancy are: (1) incorrect execution of the error reporter recovery block and (2) no invocation of the error reporter acceptance test. In either case, information reaching the global executive is suspect, and should be disregarded. If the rest of the system is properly functioning, the only penalty for no retirement at this point would be the unnecessary overhead necessitated by the higher number of active but not functional processors. Because the discrepancy is a processor disagreement from a majority vote, it should be counted in the total of the error reports of the other processors. If any other single disagreement occurs, the processor would be retired at the end of the next frame.

GEXECTEST detects both the second and third faults listed above. The number of processor disagreements registered in each processor error report are counted; retired or self-reporting processor disagreements are ignored. If the corresponding position in the reconfiguration word is zero when there are two or more reports which have bits set, or the reconfiguration word has a bit set when fewer than two (i.e. one or zero) processors are reported in the error words, then a boolean variable fallflg is set. The alternate global executive is invoked if fallflg is TRUE.

23

```
05600   PROCEDURE PREGEXEC;
05700   (*This procedure copies the least significant bits of the
05800    error reporter word bits into the most significant positions
05900      after checking the frame number *)
06000
06100   VAR
06200           excount:INTEGER;
06300           ERR:INTEGER;
06400           J,M:INTEGER;
06500
06600   begin
06640           mismatch:=0;                          (*mismatch is a global integer
06680                                                  variable used for marking
06690                                                  procs. not running ertask*)
06700           for J:=0 to maxprocessors do begin
06800                   excount:=prevote[errerr,J] div 256;
06900                   err:=prevote[errerr,J] mod 256;
07000                   if excount=(framecount mod 256) then
07100                           prevote[errerr,J]:=257*err
07150                   (*copy least sig. bits to most sig. position
07175                     if frame count OK*)
07200                   else mismatch:=mismatch + 1;
07225                   (*otherwise send word to error reporters in  subsequent
07237                     frame*)
07250                   mismatch:=mismatch * 2;
07300           end;
07800   end;
*
```

Figure 3.3. Listing for Global Executive Acceptance test; PREGEXEC

```
11000    PROCEDURE GEXECTEST;
11100    (*Global Executive Acceptance test*)
11200
11300    TYPE
11400            ZERO_ONE=0..1;
11500    VAR
11600            DIVISOR,CHECK,I,J,SUM:INTEGER;
11700            FAILFLG:BOOLEAN;
11800            LAST_DIG:ZERO_ONE;
11900    begin
12000            divisor:=1;
12100            failflg:=false;
12200            for I:=0 to maxprocessors do begin
12300                                    (*...do for each position of report*)
12340    (*This procedure is written under the assumption that the primary
12380       global executive has rotated the error reports a total of 8
12420       positions.  If this is not the case, additional division by
12460       (8 - 1 - maxprocessors)*2 for each error report is necessary *)
12500                    sum:=0;
12600                    for j:=0 to maxprocessors do begin
12700                                    (*...do for each error report*)
12800                            last_dig:=(prevote[errerr,j] div divisor)
12900                                    mod 2;
13000                    if (not working[j]) or (i=j)
13100                       then last_dig:=0;
13130                    if(not working[i]) and (odd(last_dig))
13160                       then begin
13190                            recfail:=recfail+divisor;
13191                            (*recfail is a global integer
13192                                showing a retired proc. working*)
13193                            last_dig:=0;
13196                        end;
13200                    sum:=sum + last_dig;
13300                end;
13400                check:=reconf div divisor;
13500                if odd(check)
13550                   then begin
13600                            if(sum<2)and(working[i]) then failflg:=true
13700                        end
13800                   else if sum>=2 then failflg:=true;
13900                divisor:=divisor*2;
14000            end;
14100        if failflg then altgexec
14200        mismatch:=mismatch + 256*(framecount mod 256);
14250                    (*indicate successful completion of acceptace test
14275                       to error reporters of next frame *)
14300    end;
*
```

Figure 3.3 (continued). Listing of Global Executive Acceptance Test: GEXECTEST.

3.3. ALTERNATE GLOBAL EXECUTIVE

The alternate global executive, ALTGEXEC performs a function identical to the primary routine, but in an independent manner. The flow chart and listing for this procedure are shown in figures 3.4 and 3.5. Input to the alternate routine is the same as that used by the acceptance test: i.e. the error reports replicated by PREGEXEC. Unlike the primary routine, ALTGEXEC sums the totals of the disagreeing processors in descending order, and stores these totals in an integer array. If the totals in this array are less than two, then a zero is placed in the corresponding position of an alternate reconfiguration word, reconfa. Otherwise, the position is set to 1. A second difference between the primary and alternate is that the error words are not destructively read, and can be saved by the system if desired. As a final step of execution, ALTGEXEC sets the value of the primary reconfiguration word to that of the alternate. The primary reconfiguration word value can also be saved prior to execution of this step.

3.4. IMPLEMENTATION REQUIREMENTS

Three new procedures: GEXECTEST, PREGEXEC, AND ALTGEXEC are required for the operating system. PREGEXEC must be invoked prior to the execution of the primary global executive (GEXECTASK), and GEXECTEST is executed at its completion. This latter routine will invoke procedure ALTGEXEC, the alternate global executive, if required. Although the routines are presently declared as procedures, they may be changed to functions in order to be compatible with the form of GEXECTASK.

An additional global integer variable, called mismatch, is required. Frame count disceprancies detected in the PREGEXEC routine are recorded in a manner similar to processor error reports, i.e. by placing a "1" in the appropriate position of the word. The error reporters of other processor will read mismatch and increment the error counter for the appropriate processor if PREGEXEC reports a frame count disagreement.

A second global integer variable designated as recfail is used to enable the global executive to indicate the unsuccessful retirement of a failed processor. As is the case with mismatch, the faulty processor is noted by a "1" in the appropriate position. As noted previously, the global executive is not capable of determining whether the processor actually did not respond to the reconfiguration order for retirement or whether the "working" array is incorrect and thus, no further action can be taken by the global executive.

Changes in the values of each element of the prevote [errerr,*] array will occur due to the implementation of the fault-tolerant error reporter and global executive. As noted previously, PREGEXEC requires the frame count be encoded in the first half of the error report from each processor by the error reporter recovery block. In addition, the least significant bits of the error reports are replicated in the most significant positions by PREGEXEC. It is not anticipated that these changes have any impact on the rest of the SIFT executive.

26
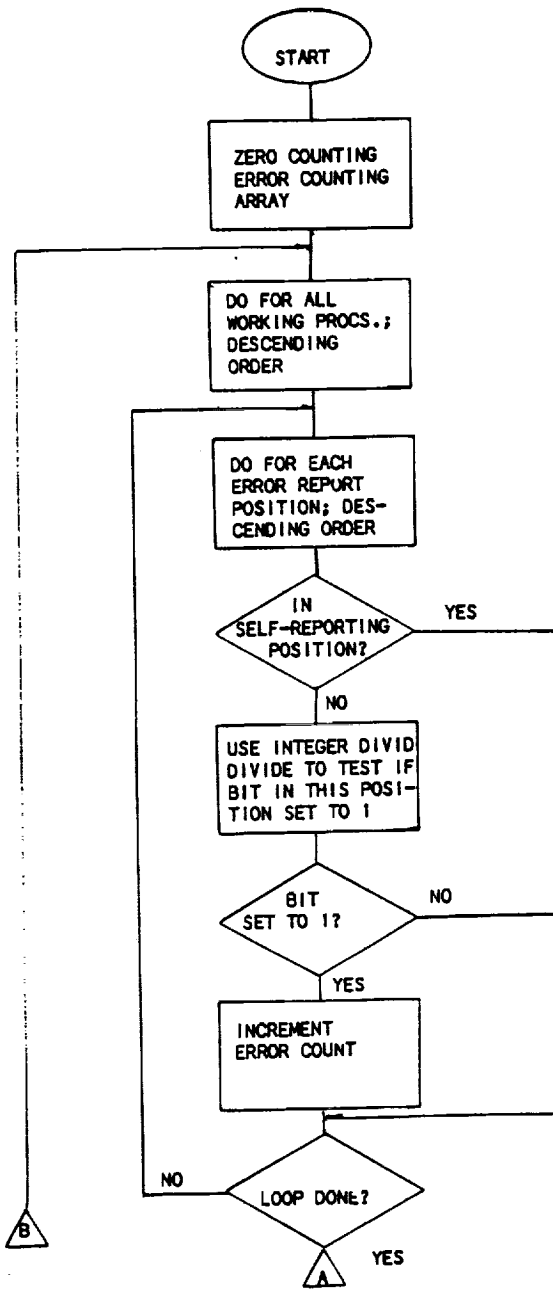
Figure 3.4. Flow chart of the Alternate Global Executive
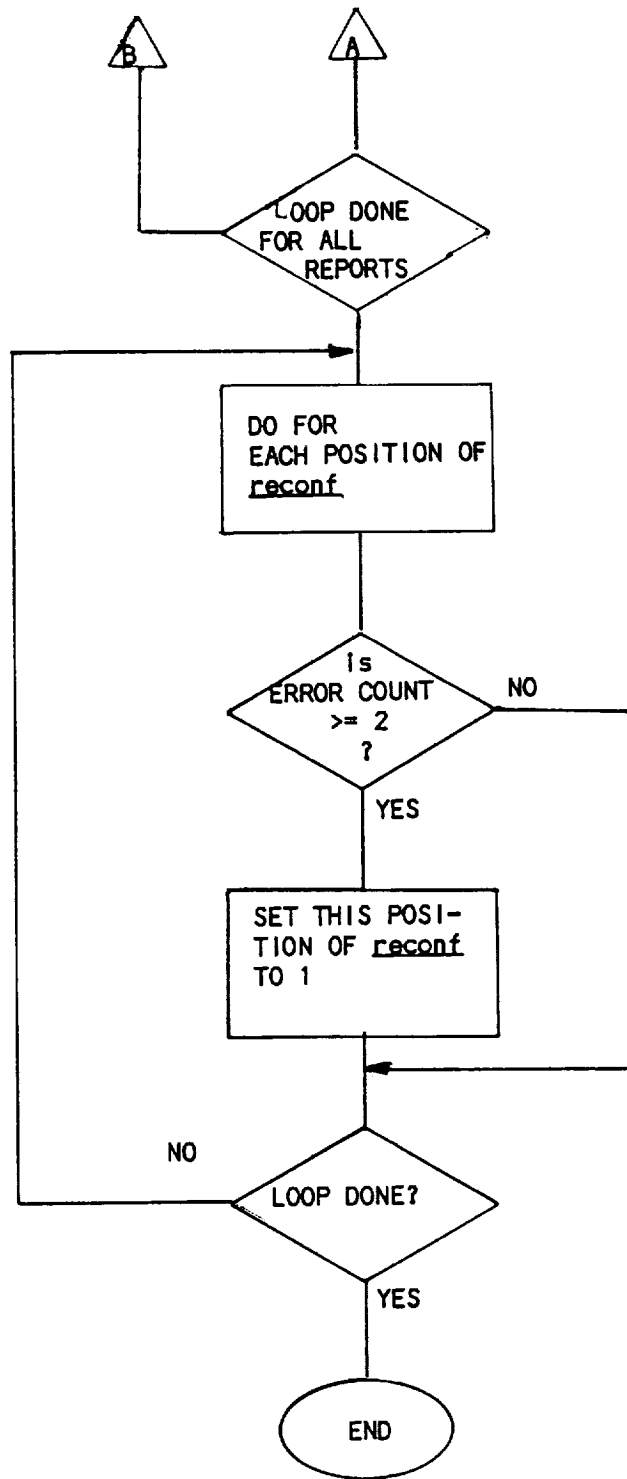
Figure 3.4. (CONTINUED). Flow chart for ALTGEXEC.

28

```
08000     PROCEDURE ALTGEXEC;
08100     (*This is the alternate global executive*)
08200
08250     const    maxdiv=32;
08300     VAR
08400              RECONFA,DIVISOR,MULT,J,K,L,M:INTEGER;
08500              ERCOUNT:PROCINT;
08550              LAST:INTEGER;
08600     begin
08700              for J:=0 to maxprocessors do ercount[J]:=0;
08800                                        (*...Initialize ercount*)
08900              FOR J:= maxprocessors downto 0 do
09000                if working[j] then begin
09100                                        (*...do for each error report*)
09150                     divisor:=maxdiv;
09200                     for k:=maxprocessors downto 0 do begin
09300                                        (*...do for each position of report*)
09400                          If J=k then last:=0
09500                           else last:=prevote[errerr,j] div divisor;
09550                          If odd(last) then ercount[k]:=ercount[k]+1;
09700                          divisor:=divisor div 2;
09800                     end
09850                end;
09900                                        (*...now write reconfa*)
10000              reconfa :=0;
10100              mult:=1;
10200              for I:=0 to maxprocessors do begin
10300                     If ercount[I]>=2 then reconfa:=reconfa+mult;
10400                     mult:=mult*2;
10500              end;
10600              pre_broadcast(gexecreconf,reconfa);
10800     end;
*
```

Figure 3.5. Listing of Alternate Global Executive

3.5. VALIDATION

The critical nature of the global executive acceptance test and the alternate global executive necessitates a comprehensive set of validation tests in order to demonstrate that the incorporation of these routines into the SIFT executive system do not negatively impact the overall reliability.

An exhaustive set of tests would involve testing each bit of each error report for the appropriate response for every possible configuration of all other error bits, the configuration of the working array, and the configuration of the reconf word. For six processors, there are a total of 281 trillion states of these variables, a rather intimidating number. However, the need for comprehenive testing remains. Thus, a major portion of the testing effort was devoted to the choice of an appropriate subset of these variables that would conclusively demonstrate that the global executive recovery block does not contain errors.

A fault-tree methodology was used to reduce the number of tests to a manageable number. The objective was to develop the trees to a sufficient level such that the primal events, i.e. those at the bottom of the tree, could be tested by a reasonable number of cases. If this testing showed that an insufficient number of primal events existed to make the top event (Failure of the global executive) true, then the validation would be complete.

The highest level tree is shown in figure 3.6. The top event, failure of the global executive recovery block, can be caused by either (1) a failure in the primary global executive and failure of the acceptance test to detect the failure or (2) the acceptance test invoking the alternate routine and failure of the alternate routine. For the purpose of this analysis, failure of the primary routine is a given, and thus, failures of the acceptance test and the alternate must be considered. However, because these routines function together as one unit, they are tested together in the validation procedure. Moreover, they both perform the same operation, i.e. determining the number of valid indications to discard a processor, and thus, are subject to the same types of faults. Hence, subsequent levels of development of these fault trees apply to both routines.

The next level of development shows the potential failed states of the acceptance test and the alternate global executive, which, as noted above, are the same. Two general classes of these possible failures exist: failure to identify a faulty processor (i.e. one where there are a sufficient number of agreeing error reports) in the reconf word, and failure to detect a "false positive" (i.e. the marking of a processor for retirement without the required number of agreeing error reports).

Figure 3.7 is the tree for the first class of failures: one, two or three faulty processors remaining unidentified. Validation test 1010 will test the software for each possible state of the error reports which would indicate a single processor as having failed. A large reduction in the number of states of the error reporter words can be achieved by consideration of the criteria for retirement: in order for a processor to be retired, the error reports of two other processors must indicate it had more than 2 disagreements from the majority in the previous frame. Thus, if the recovery block can be shown to detect any two working processors indicating a third processor as having failed then it

30

will designate this failure if more than two processors so report.

As is shown in figure 3.8, failure to identify a single faulty processor may occur when 0, 1, 2, or 3 processors have been retired by the reconfiguration task. Considering all permutations of the SIFT configuration would lead to an impractical number of test cases, and the following logic describes the reduction in the validation process: there is only 1 SIFT configuration when all processors are working, and six possible configurations if a single processor is retired. These configurations are tested with all permutations of two processors indicating a third as faulty. Once the validations has establish that the global executive can correctly identify a failed processor with any single processor configured out of the system, validations for two processors configured out need only consider cases where the first retirement is held fixed (at processor 0) and the second is varied among the remaining 5. A similar line of reasoning can be used to consider three retired processors. Figure 3.9 shows the pattern of SIFT configurations that are tested for the single faulty processor case.

The next errors covered in this branch are the failure to identify two and three processors as having failed. In principle an exhaustive test should cover each possibility of two or three processors having failed. However, as implied in the fault tree, this can be broken into the failure to detect the first faulty processor, failure to detect the second, and failure to detect the third (if applicable). The failure to detect the first processor when no other processors have failed has been covered in test 1010A, along with arguments which extend the validity of this test to all states of working and reconf. This same argument can be easily extended to cover the case of more than one processor having failed.

Table 3.1   Illustrates the validation tests required to cover all failure possibilities under the tree 1000. The validation procedure calls for processor 0 to be designated as faulty by processors 1 and 2, and that processors 1 though 5 be tested in turn in a manner similar to the single processor failed validation described above. An analagous line of reasoning can be used for the validation of the third processor failed case: processors 0 and 2 designate processor 1 as failed, processors 1 and 2 designate 0 as failed, and the third processor can be designated from the remaining processors (2 through 5). Table 3.1 lists this procedure explicitly.

Figure 3.10 shows the development of the class of errors concerned with designation of a functional processor as faulty. The possible failures resulting in a spurious processor failure indication include counting the error report of a processor which is not working as part of the total disagreement count, counting a processor's vote on itself, or the designation of a functional processor on the basis of 1 or no other processor error reports. These failures will be tested in tests designated as 1100A, 1100B, 1100C, and 1100D. A reduction in the number of tests to be performed occurs by the fact that these failures will take place for any value of reconf. Also, because the global executive acceptance test and alternate operate in the same statement sequence regardless of the SIFT state (i.e. there is no branching to different modules of the code depending on the values of working, reconf, or the failure of a particular processor), the same tests apply to all values of working.

Table 3.2 shows the list of validation tests and the range of working, reconf,

and error reports. Tests 1100A and 1100B can be executed simultaneously with test 1000A. Test 1100C is executed by placing a single bit in all 36 possible error reporter positions, setting the corresponding position in reconf, and determining that both the acceptance test detects the error and the alternate routine functions correctly. Test 1100D is performed by setting each bit of reconf to 1 with no bits set in the error reporter words.
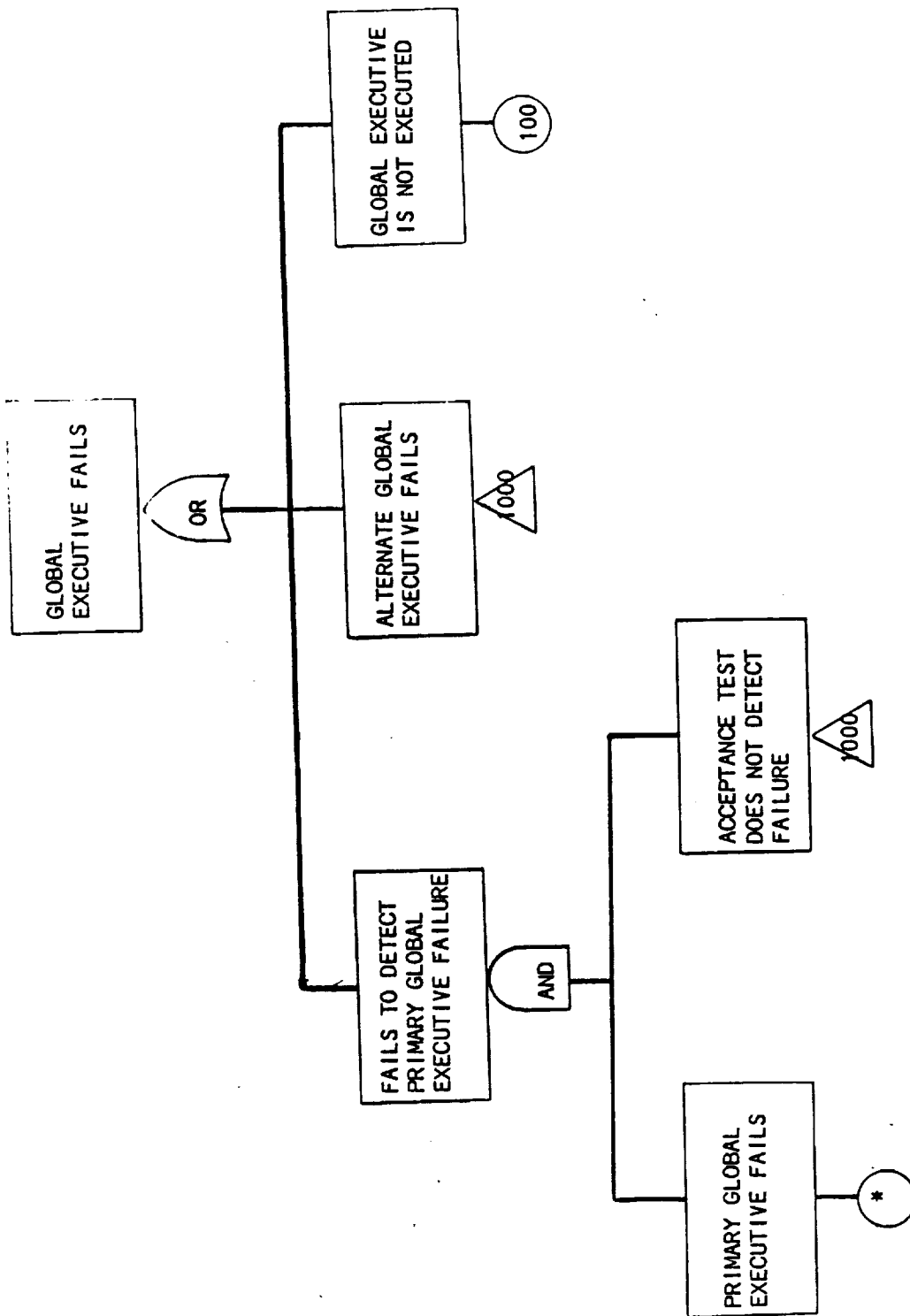
Figure 3.6. Top Level Fault Tree for Global Executive Validation

33
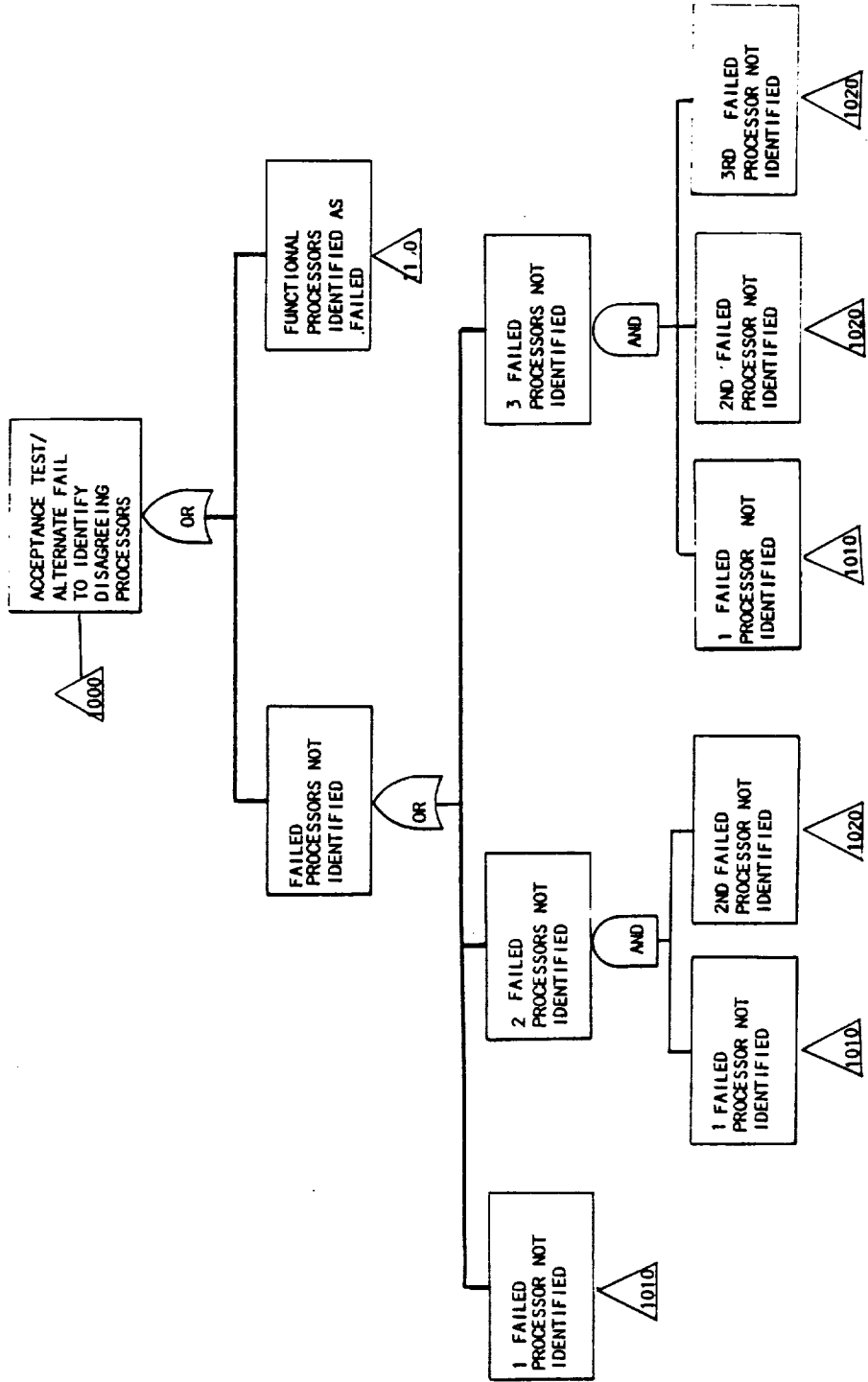
Figure 3.7. Classes of Global Executive Faults
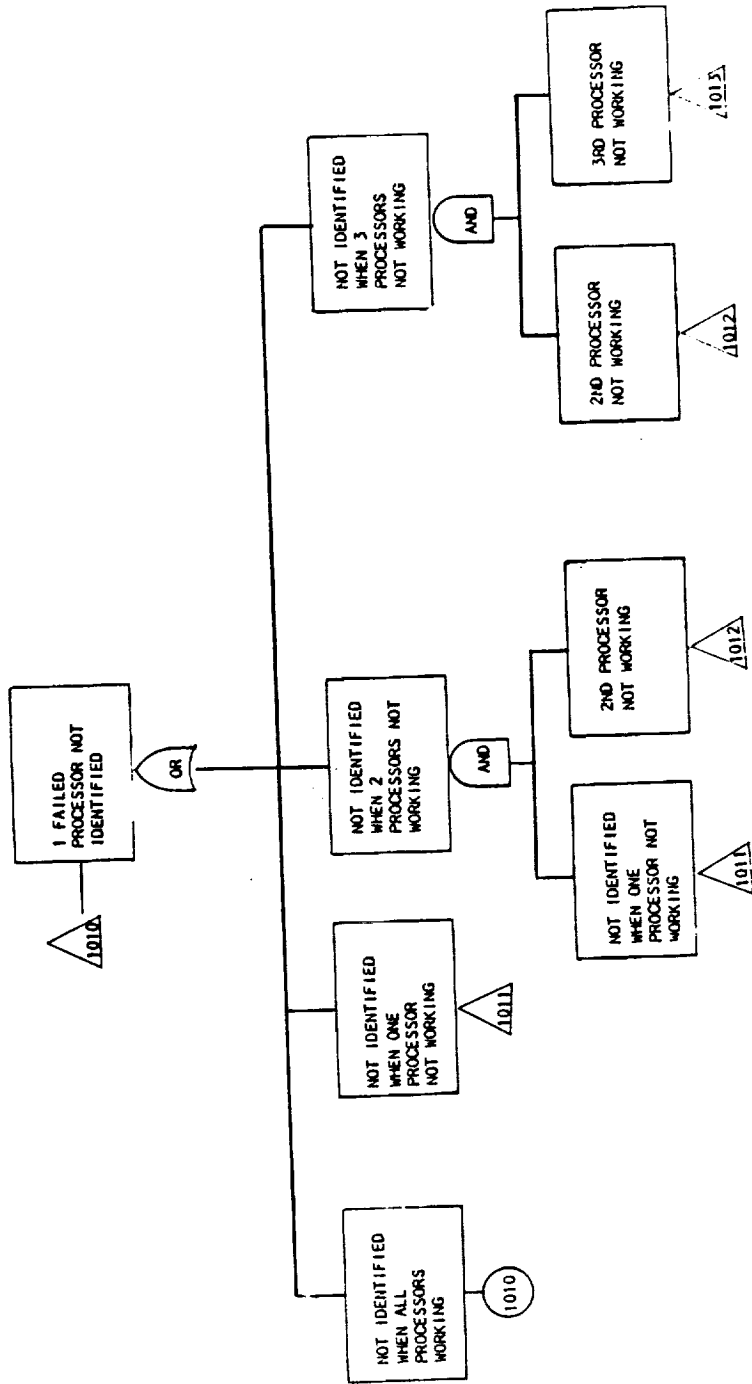
Figure 3.8. Global Executive Detection Failure

TABULAR "OR"

1011
PROCESSOR 0 NOT WORKING
PROCESSOR 1 NOT WORKING
PROCESSOR 2 NOT WORKING
PROCESSOR 3 NOT WORKING
PROCESSOR 4 NOT WORKING
PROCESSOR 5 NOT WORKING

TABULAR "OR"

1012
PROCESSOR 1 NOT WORKING
PROCESSOR 2 NOT WORKING
PROCESSOR 3 NOT WORKING
PROCESSOR 4 NOT WORKING
PROCESSOR 5 NOT WORKING

TABULAR "OR"

1013
PROCESSOR 2 NOT WORKING
PROCESSOR 3 NOT WORKING
PROCESSOR 4 NOT WORKING
PROCESSOR 5 NOT WORKING

Figure 3.9. Expansion of Figure 3.7.

1020

TABULAR "OR"
PROCESSOR 1 NOT IDENTIFIED
PROCESSOR 2 NOT IDENTIFIED
PROCESSOR 3 NOT IDENTIFIED
PROCESSOR 4 NOT IDENTIFIED
PROCESSOR 5 NOT IDENTIFIED

1030

TABULAR "OR"
PROCESSOR 2 NOT IDENTIFIED
PROCESSOR 3 NOT IDENTIFIED
PROCESSOR 4 NOT IDENTIFIED
PROCESSOR 5 NOT IDENTIFIED

Figure 3.9. Expansion of Figure 3.7 (continued)
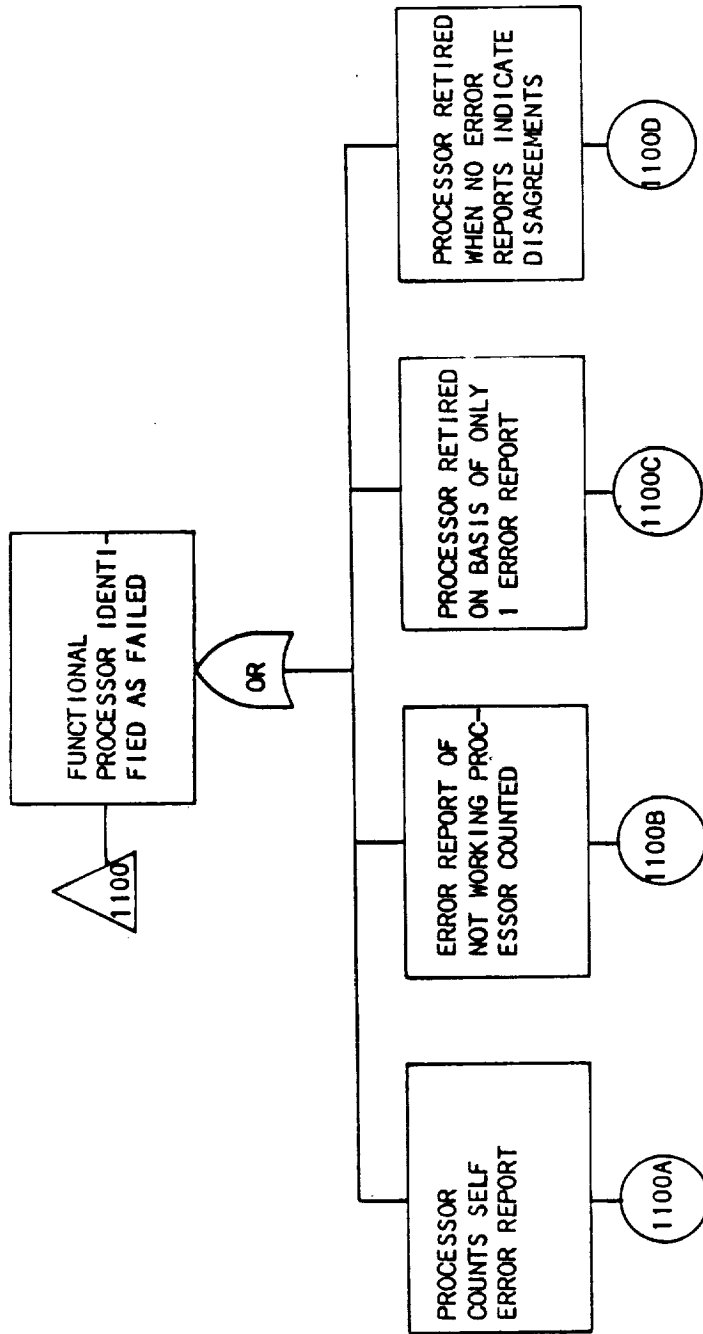
Figure 3.10. Spurious Identification of a Functional Processor

38

Table 3.1. Validation Tests for Global Executive Faulty Processor Dectection Failure

TEST	ERROR DESCR.	working	prevote	reconf	NOTE
1000A	1 FAILED PROC. UNDETECTED BY PRIMARY	0,1,2,3 not working	1 reported (1st indicated by any 2 other error reports)	0 retiring	1
1000B	2 FAILED PROC. UNDETECTED BY PRIMARY	0 not working	2 reported (2nd indicated by any 2 other error reports)	1 retiring (1st in any position of reconf)	2
1000C	3 FAILED PROC. UNDETECTED BY PRIMARY	0 not working	3 reported (3rd indicated by any 2 other error reports)	2 retiring (2nd in any position of reconf)	3

NOTES:

1. Failure of the primary global executive for this condition is manifested by both the following conditions: (1) one processor is identified as having excess disagreements by the individual error reports, and (2) the primary global executive did not mark this processor for retirement in the reconf word. This validation test is performed with 0,1,2,3 processors not working in order to determine whether the acceptance test and alternate are capable of detecting a single (or the first in the case of multiple) processor failure given any SIFT state. If any more than three processors are not working, the entire computer fails.

2. Failure of the primary global executive for this condition is manifested by the following conditions: (1) two processors are identified as having excess disagreements by the error reports, (2) the primary global executive marked the first processor for retirement in reconf, and (3) the primary global executive did not mark the second processor for retirement. Validation testing for detection of the first processor given any configuration of working with 0, 1, or 2 processors out and no processors marked for retirement has already been performed in 1000A. Thus, this validation need only establish that the acceptance test can detect a second processor as having failed when the primary has marked only a single processor for retirement in reconf.

3. Failure of the primary global executive for this condition is manfested by the following conditions: (1) three processors are identified as having excess disagreements by the error reports, (2) the primary global executive marked the first two processors for retirement in reconf, and (3) the primary global executive did not mark the third processor for retirement. Validation testing for detection of the first processor given any configuration of working with 0, 1, or 2 processors out and no processors marked for retirement has already been performed in 1000A. Validation of the ability of the acceptance test to detect the second processor failure has been performed in 1000B. Thus, this validation need only establish that the acceptance test can detect a third processor as

having failed when the primary has   marked only two processors for
retirement in _reconf_.

Table 3.2.   Validation Tests for Incorrect Retirement Errors of
Global Executive

TEST	ERROR DESCR.	working	prevote	reconf
1100A	PROC. RETIRED ON BASIS OF SELF DIAGNOSIS	0,1,2,3 not working	1 other proc. reporting (any position)	1 proc. marked for retirement (any position)
1100B	PROC. RETIRED ON BASIS OF NOT WORKING PROC. REPORT	1 not working	1 other proc. reporting (any position)	1 proc. marked for retirement (any position)
1100C	PROC. RETIRED ON BASIS OF ONLY 1 ERROR REPORT	0 not working	1 other proc. reporting (any position)	1 proc. marked for retirement (any position)
1100D	PROC. RETIRED ON BASIS OF NO ERROR REPORTS	0 not working	no other proc. reporting	1 proc. marked for retirement (any position)

APPENDIX A. ERROR REPORTER DRIVER ROUTINES

Although both the error reporter acceptance test and the alternate routine are relatively brief procedures, a complete test required the simulation of a major portion of the SIFT operating system. The simulation program, called DRIVER, prepares the errors and working arrays of the voter and the err word output of the error reporter based on externally input data. It next invokes the acceptance test, outputs its results to file TTY (for diagnostic purposes), and invokes the procedure if an error is detected. A complete listing of the program follows this description.

Figure A.1 is a heirachical representation of the program organization. The main program first invokes procedure IOFILES which either opens a previously written test data input file, prepares to write a new file, or simply accepts input and outputs directly to file TTY. Each of the subsequent procedures contain branches for the data source and destination defined in this routine. The main program the invokes procedure LIMREP, which determines the number of iterations (i.e. frames). FRAME COUNTER, the next procedure invoked, sets the value of framecount against which excount, the internal counter of the error reporter, is compared. The program then invokes the VOTER and ERROR REPORTER procedures which, on the basis of input data, prepare the working and errors arrays and the err and excount variables. The ACCEPTANCE TEST procedure is then run, and the alternate error reporter is called by it in the event of the discrepancies discussed above. Subsequent iterations repeat the process from FRAME COUNTER through ACCEPTANCE TEST until the repetition limit is reached. Upon exiting the loop, the main program invokes procedure which closes any of the files opened in IOFILES and ends the simulation.

It should be noted that the actual error reporter acceptance test and alternate error reporter which were tested are shown in this listing, and that they are not identical to those shown in figures 2.2 and 2.3. These latter listing were changed to be compatible with the SIFT operating system (by including a prebroadcast(errerr,err) statement) and eliminating display related statements (e.g. outputs to TTY and the BINPARS routine which represented the error words as binary numbers). An additional alteration was made to the acceptance test routine to include testing of the mismatch variable. None of these changes are sufficiently significant to warrant additional validation testing.

Appendix C contains a sample output from this driver routine.
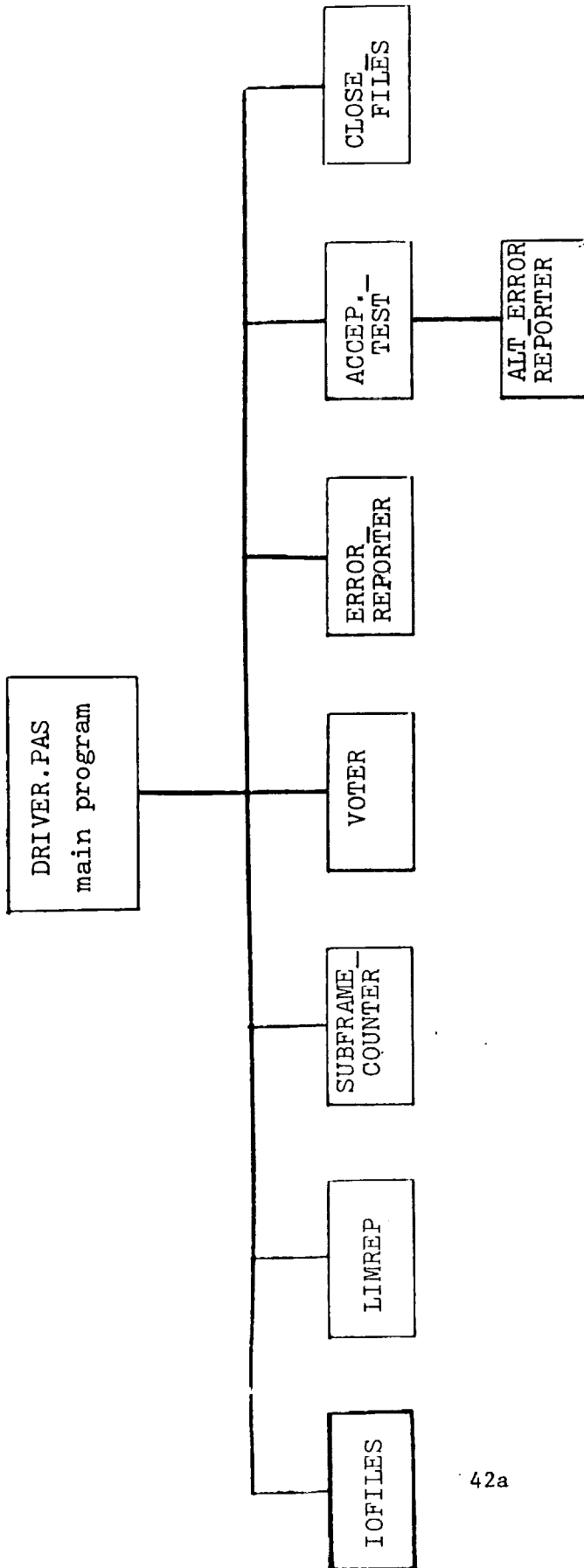
Figure A.1 Organization of Program DRIVER

42a

```
00100        PROGRAM DRIVER;
00200        CONST
00300                                      (*the following declarations are taken from
00400                                       the AUGUST. 1980 VERSION OF THE SIFT
00500                                       OPERATING SYSTEM *)
00600                      MAX PROCESSORS=5;
00700                      MAXframe=50;
00800                      THRESHOLD=2;
00900        TYPE
01000               PROCESSOR=0..MAXPROCESSORS;
01100               PROCINT=ARRAY[PROCESSOR] OF INTEGER;
01200               PROCBOOL=ARRAY[PROCESSOR] OF BOOLEAN;
01300        VAR
01400               ERR:INTEGER;
01500               ERRORS:PROCINT;
01600               REPORT:PROCINT;
01700               WORKING:PROCBOOL;
01800               framecount:INTEGER;
01900        (*
02000                      the following declarations are necessary for
02100                      the error reporter recovery block *)
02200               ERFAILS :integer;
02300        (.*
02400                      the following varables are necessry only
02500                      for the driver prcedures*)
02600               I,J,K:INTEGER;
02700               RPTLIM :INTEGER;
02800               FILENAME :PACKED ARRAY[1..8] OF CHAR;
02900               TITLE: PACKED ARRAY[1..40] OF CHAR;
03000               FIL:INTEGER;
03100               INTREP :PROCINT;
03200        PROCEDURE IOFILES;
03300        (*this program sets up files for both input and output
03400          as deterined by FIL input from the keyboard*)
03500        begin
03600               writeln(tty,'Test of Error Reporter Recovery Block');
03700               writeln(tty,'I/O options: tty alone(0), input file(1)');
03750               writeln(tty,'Create File(2)');
03800               read(tty,fil);
03900               if fil>0 then begin
04000                      writeln(tty,'enter filename');
04100                      readln(tty);
04200                      readln(tty,filename);
04300                      if fil=1 then reset(input,filename)
04400                      else rewrite(output. filename);
04500                      writeln(tty,filename.' readv');
04600               end
04700               else writeln(tty,'I/O through terminal only');
04800        end;
04900
05000        PROCEDURE LIMREP;
05100        (*SET REPETITION LIMIT FOR MAN PROCEDURE*)
05200        begin
05300               if fil<>1 then begin                 (*prompts for TY input*)
05400                      writeln(tty);
05500                      writeln(tty,'enter number of repetitions');
05600                      read(tty,rptlim);
05700                      if fil=2 then writeln(output,rptlim);
05800               end
05900               else begin
06000                      read(input,rptlim);
```
43

```
06100                        writeln(tty,rptlim, ' repetitions')
06200              end;
06300                        rptlim:=rptlim-1;
06400     end;
06500
06600     PROCEDURE VOTER;
06700     (*this procedure is to manuallv input  the error[p[i]]
06800       array generated in the voter routine*)
06900
07000     begin
07100        if fil<>1 then begin           (*tty iput*)
07200              writeln(tty,'procedure voter -- enter errors');
07300              for i:=0 to maxprocessors do begin
07400                        writeln(tty,'number of errors for processor ',
2);
07500                        read(tty,errors[i]);
07600                 writeln(tty,'working? (1/0)?');
07700                 read(tty,intrep[i]);
07800                 writeln(tty);
07900                 if fil=2 then write(output. errors[i],intrep[i]);
08000              end
08100        end
08200        else
08300              for i:=0 to maxprocessors do     (*file input*)
08400                        read(input,errors[i],intrep[i]);
08500
08600        for i:=0 to maxprocessors do           (*all*)
08700              if intrep[i]<1 then working[i]:=false
08800               else working[i]:=true;
08850     end;
08900     PROCEDURE BINPARS(VAR NUM:INTEGER);
09000     (*procedure to represent an integer as a 16 bit string *)
09100     var
09200              binr: array[0..15] of integer;
09300              tnum:integer;
09400              divis,i,j:integer;
09500              byte: packed array[1..20] of char;
09600     begin
09700              divis=32768;
09800              if num>65535 then begin
09900              writeln(tty,'overflow');
10000                        num:=num mod 65535;
10100              end;
10200              tnum:=num;
10300              j:=0;
10400              for i:=15 down to 0 do begin
10500                        if tnum div divis>=1 then begin
10600                                tnum:=tnum mod divis;
10700                                binr[i]:=1
10800                        end
10900                        else binr[i]:=0;
11000                        divis:=divis div 2;
11100                        j:=j+1;
11200                        if binr[i]=1 then byte[j]:='1'
11300                                else byte[j]:='0';
11400                        if (i mod 4=0) then begin
11500                                j:=j+1;
11600                                byte[j]:=' ';
11700                        end;
11800              end;
11900              writeln(tty,byte);
12000              writeln(tty);
```

44

```
12100      end;
12200      procedure error_reporter;
12300      (*this procedure is to manually input the report[p[i]]
12400        array assumed to be generayted by the error reporter*)
12500
12600      VAR EXCOUNT:INTEGER;
12700      begin
12800          if fil<>1 then begin                    (*tty input*)
12900              writeln(tty);
13000                                              (*initialize the frame count *)
13100              writeln(tty,'framecount is',framecount:2,' enter execution');
13200              read(tty,excount);
13300                                      (*error reporter would be incrementing
13400                                        its own frae counter here *)
13500              writeln(tty,'title');
13550              readln(tty);
13600              readln(tty,title);
13700              writeln(tty);
13800              if fil=2 then write(output,excount,title);
13900              writeln(tty,'procedure error repoter -- enter report');
14000              err:=0;
14100              for i:= maxprocessors downto 0 do begin
14200                      writeln(tty,'proc',i:2,'  err rpt.(1/0) =');
14300                      read(tty,report[i]);
14400                      err:=err*2;
14500                      if (not working[i]) or (report[i]>0)
14600                       then err:=err+1;
14800                      if fil=2 then write(output,report[i])
14900              end;
15000              writeln(tty);
15100              err:=err + 256*excount;  (*combine error and execution ct *)
15200              if fil=2 then write(output,err);
15300          end
15400          else begin
15500                                              (*file input *)
15600              read(input,excount,title);
15700              for i:=maxprocessors downto 0 do
15800                      read(input,report[i]);
15900              read(input,err)
16000          end;
16100              writeln(tty);
16200              writeln(tty,title);
16300              writeln(tty,'frame no.',framecount:3,'execution',excount:3);
16400              writeln(tty);
16500              writeln(tty,'processor':15,'voter error':20,'error report':20,
16600                      'working':20);
16700              for i:=0 to maxprocessors do begin
16800                      writeln(tty);
16900                      writeln(tty,i:10,errors[i]:20,report[i]:20,
17000                              intrep[i]:20);
17100              end;
17200              writeln(tty);
17300              writeln(tty,'primary error word= ',err:5);
17400              binpars(err);
17500      end;
17600      PROCEDURE FRAME_COUNTER;
17700      (*This procedure is to simulate the execution counter on the
17800        error reporter acceptance test by means of manual input *)
17900
18000              begin
*
```

```
18100                         framecount:=framecount+1;
18200     end;
18300
18400     PROCEDURE CLOSEFILES;
18500     (*Close the input or output files if necesary*)
18600     begin
18700             if fil=1 then close(input);
18800             if fil=2 then close(output);
18900     end;
19000
19100     PROCEDURE ALT_ERROR_reportER;
19200     (*this is the alterate error reporter*)
19300
19400     CONST
19500             ALLONES=377B;
19600     VAR
19700             ERRA:INTEGER;    (*aternate error word*)
19800             I,K:INTEGER;
19900
20000     begin
20100             writeln(tty);
20200             writeln(tty,'alterate error reporter invoked');
20300             erra:=allones;
20400             k:=1;
20500     .       for i:=0 to maxprocessors do
20600                     begin
20700                       if (errors[i]<threshold) and (working[i])
20800                         then erra:=erra-k;
20900                       k:=k*2;
21000                     end;
21100             erra:=erra - (allones - k + 1);      (*remove leading bits*)
21200             writeln(tty,'alternate error word=',erra:5);
21300             err:=erra + 256*framecount;
21400             binpars(err);
21500             writeln(tty)
21600     end;
21700
21800     PROCEDURE ACCEPTANCE_TEST;
21900     (*error reporter accetance test*)         .
22000     VAR
22100             EXCOUNT,WRONG,RIGHT,DIVISOR,CHECK,I,J:INTEGER;
22200             FAILFLG:BOOLEAN;
22300
22400     begin
22500
22600             excount:= err div 256;
22700             err:=err mod 256;
22800         if excount=framecount then begin
22900             wrong:=0;
23000             failflg:=false;
23100             right:=0;
23200             divisor:=1;
23300             for j:=0 to maxprocessors do   (*check for omission errors*)
23400                     begin
23500                     if (errors[j]<threshold) and (working[j])
23600                       then right:=right+1;
23700                                                 (*count for omissions test*)
23800                     check:=err div divisor;
23900                                                 (*shift err appropriate
24000                                                  no. of places to the right*
*
                                      46
```

```
24100                              if odd(check) then begin
24200                                    wrong:=wrong+1;  (*count for omissions test*)
24300                                    if (errors[j]<threshold) and (working[j])
24400                                       then failflg:=true (*check for false positives*
)
24500                                 end;
24600                                 divisor:=divisor*2;
24700                            end;
24800                      if wrong+right<>maxprocessors +1 then failflg:=true;
24900                                           (*omissions test*)
25000                      if failflg then alt_error_reporter
25100                      else writeln(tty,'error reporter OK');
25200                  end
25300                 else begin
25400                      writeln(tty);
25500                      writeln(tty,'primary error reporter did not run');
25600                      alt error_reporter;
25700                      writeln(tty);
25800                  end;
25900             end;
26000
26100      (*MAIN PROCEDURE*)
26200
26300      BEGIN
26400           IOFILES;
26500           LIMREP;
26600           REPEAT
26700                 frame_COUNTER;
26800                 VOTER;
26900                 ERROR_REPORTER;
27000                 ACCEPTANCE_TEST;
27100           UNTIL framecount>RPTLIM;
27200           CLOSEFILES;
27300      END.
*
```

48

APPENDIX B. GLOBAL EXECUTIVE DRIVER ROUTINES

A significantly larger set of test cases was necessary for the global executive validation, and thus, its driver routine, GEXEC, used file input exclusively for the validation test input data. Two routines were used to input test data: INGEX, which accepted data directly from a terminal for generation of a small number of test cases, and MVTEST, which had an internal procedure for generation of a larger number of cases.

Program INGEX consists of 5 procedures: BINPARS, which represents integers as 16-bit binary numbers, CONV, which converts the input error reports and retiring processors into integers (err and reconf) used by the global executive, PRELIM, which opens a file for the test cases, OUTFILE, which writes the data to the file, and INDATA, which issues prompts to file TTY and processes the resultant input.   The program first opens a file with procedure PRELIM, and then accepts input and writes to the file until the user specified number of test cases has been reached, and then saves the file for use by GEXEC.

MVTEST is composed of 4 procedures: ZERO, which zeros out the error reporter representation array for a new case, MVINIT which initializes an array containing all possible test cases for a given number of faulty processors, DISP, which performs additional processing and writes the cases to an output file, and MATCH, which selects a single test case from the possibilities generated by MV. Modifications to the main procedure, MATCH, and DISP were made for the generation of test cases for various configurations of the system (i.e. values of working) and number of processors becoming faulty in the current frame as described in section 3.5.

Program GEXEC contains 7 procedures: BINPARS, which was described above, PREGEXEC, the first module of the global executive acceptance test, ALTGEXEC, the alternate global executive, GEXECTEST, the second (and main) module of the acceptance test, INFILE, which reads files created by either INGEX or MVTEST, and PRELIM, which opens the files used by GEXEC. After PRELIM opens a file, the program flow is from INDATA, which prepares the input for the acceptance tests and alternate routine (if necessary), to PREGEXEC, GEXECTEST, and ALTGEXEC (if invoked by GEXECTEST).   This sequence is repeated until the end of file condition is reached.

A modification of GEXEC, designated VALGEX, was used for creating a more terse output. This was necessitated by the large number of test cases (almost two thousand).

As was the case with the error reporter, modifications of the PREGEXEC, GEXECTEST, and ALTGEXEC procedures were made to remove all TTY I/O, make the output of the routines compatible with the SIFT operating system, and to include references to the mismatch variable described in sections 2 and 3. These minor alterations are not expected to affect the correctness of the routines as established by this validation.

Listings of INGEX, MVTEST, and GEXEC follow this description, and the output of GEXEC is described in Appendix C.

PROGRAM INGEX

```
00100      PROGRAM INDEX;
00200
00300      CONST
00400              maxprocessors=5;
00500      TYPE
00600              processor=0..maxprocessors;
00700              procint=array[processor] of integer;
00800      VAR
00900              FILENAME : PACKED ARRAY[1..8] OF CHAR;
01000              CASENAME : PACKED ARRAY[1..40] OF CHAR;
01100              CASENO,MAXCASE,FRAMECOUNT:INTEGER;
01200              NUMREC.NUMOUT.NUMPROC,REPROC, FAULTPROC,
01300              NUMFAULT.PROCRET,PROCOUT:PROCESSOR;
01400              TVEC,INTREP,RETIRING:PROCINT;
01500              ERRORS:ARRAY[PROCESSOR] OF PROCINT;
01600
01700      PROCEDURE BINPARS(VAR NUM :INTEGER);
01800      (*procedure to represent an integer as a 16 bit string *)
01900      var
02000              binr: array[0..15] of integer;
02100              tnum :integer;
02200              divis,i.j:integer;
02300              byte: packed array[1..20] of char;
02400      begin
02500              divis:=32768;
02600              if num>65535 then begin
02700              writeln(tty,'overflow');
02800                      num:=num mod 65535;
02900              end;
03000              tnum:=num;
03100              j:=0;
03200              for i:=15 downto 0 do begin
03300                      if tnum div divis>=1 then begin
03400                              tnum:=tnum mod divis;
03500                              binr[i]:=1
03600                      end
03700                      else binr[i]:=0;
03800                      divis:=divis div 2;
03900                      j:=j+1;
04000                      if binr[i] =1 then byte[j]:='1'
04100                              else byte[j]:='0';
04200                      if (i mod 4=0) then begin
04300                              j:=j+1;
04400                              byte[j]:=' ';
04500                      end;
04600              end;
04700              writeln(tty);
04800              writeln(tty,byte);
04900              writeln(tty);
05000      end;
05100      FUNCTION CONV(ARAY:PROCINT): INTEGER;
05200      VAR l,j,k:integer;
05300      begin
05400              j:=1;
05500              k:=0;
05600              for l:=0 to maxprocessors do begin
05700                      k:=k+aray[l]*j;
05800                      j:=j*2;
*
```

```
05900              end;
06000              conv:=k;
06100       end;
06200
06300       PROCEDURE PRELIM;
06400       begin
06500              writeln(tty,'Enter file name');
06600              readln(tty);
06700              read(tty,filename);
06800              writeln(tty,'Enter total number of processors');
06900              read(tty,numproc);
07000              writeln(tty,'Enter number of cases');
07100              read(tty,maxcase);
07200       end;
07300
07400       PROCEDURE OUTFILE;
07500       (*write output to file and report to tty*)
07600       VAR prevote,j,k,reconf,numfault:integer;
07700       begin
07800              writeln(output.casename);
07900              writeln(tty,'case ',casename);
08000              writeln(tty);
08100              for k:=0 to maxprocessors do write(output.intrep[k]);
08200              writeln(tty,'working status');
08300              for k:=0 to maxprocessors do write(tty,intrep[k]);
08400              writeln(tty);
08500              for k:=0 to maxprocessors do begin
08600                     for j:=0 to maxprocessors do tvec[j]:=errors[k,j];
08700                     prevote:=conv(tvec) + 256*framecount;
08800                     writeln(tty,'error report for processor ',k:2);
08900                     binpars(prevote);
09000                     write(output.prevote);
09100              end;
09200              reconf:=conv(retiring) + 256*framecount;
09300              writeln(tty,'Reconfiguration word');
09400              binpars(reconf);
09500              writeln(output,reconf);
09600       end;
09700
09800       PROCEDURE INDATA;
09900       (*This procedure does the actual test case input*)
10000       VAR i,m,n,j:integer;
10100       begin
10200              writeln(tty,'enter case name');
10300              readln(tty);
10400              read(tty,casename);
10500              writeln(tty,'Enter framecount');
10600              read(tty,framecount);
10700              for m:=0 to maxprocessors do begin
10800                     intrep[m]:=0;
10900                     retiring[m]:=0;
11000                     for n:=0 to maxprocessors do errors[m,n]:=0;
11100              end;
11200                               (*...Prepare the intrep array*)
11300              writeln(tty,'How many processors are not working?');
11400              read(tty,numout);
11500              if numout>0 then begin
11600                     writeln(tty,'which processors not working?');
11700                     for i:=1 to numout do begin
11800                            read(tty,procout);
```

52

```
1 19 0 0                               intrep[procout]:=1;
1 20 00                    end;
1 210 0            end;
1 220 0                        (*...Prepare the errors array*)
1 230 0            writeln(tty,'How many processors are faulty?');
1 24 0 0           j:=0;
1 25 00            readln(tty,numfault);
1 26 00            if numfault>0 then repeat
1 27 0 0                   j:=j+1;
1 28 0 0                   writeln(tty,'wrong processor', j:3);
1 29 0 0                   writeln(tty,'which processor is faulty?');
1 30 00                    read(tty,faultproc);
1 310 0                    writeln(tty,'how many processors report it as faulty?');
1 320 0                    read(tty,numproc);
1 33 0 0                   writeln(tty,'which processors reported it?');
1 34 0 0                   for i:=1 to numproc do begin
1 3500                             read(tty,reproc);
1 36 0 0                           errors[reproc,faultproc]:=1;
1 37 0 0                   end
1 380 0            until j=numfault;
1390 0             writeln(tty);
1400 0             writeln(tty,'Summary of Error Reports of all processors');
1 410 0            writeln(tty,'Reporting        Faulty');
1 420 0            writeln(tty,'processors       processors');
1 430 0            writeln(tty);
1 44 0 0           for i:=0 to maxprocessors do begin
1 4500                     writeln(tty);
1 46 00                    write(tty,i:3,'               ');
1 47 0 0                   for m:=0 to maxprocessors do
1 48 00                            write(tty,errors[i,m]:3);
1 49 0 0           end;
1 50 0 0                       (*...Prepare the reconf word*)
1 510 0            writeln(tty);
1 520 0            writeln(tty);
1 530 0            writeln(tty,'How many processors are reconfigured out?');
1 54 0 0           read(tty,numrec);
1 55 0 0           if numrec>0 then begin
1 560 0                    writeln(tty,'which processors are reconfigured out?');
1 57 0 0                   for i:=1 to numrec do begin
```

```
15700                        for i:=1 to numrec do begin
15800                              read(tty,procret);
15900                              retiring[procret]:=1;
16000                        end;
16100                   end;
16200     end;
16300
16400     (*MAIN PROCEDURE*)
16500     begin
16600           prelim;
16700           if numproc=maxprocessors then begin
16800                 rewrite(output,filename);
16900                 for caseno:=1 to maxcase do begin
17000                        indata;
17100                        outfile;
17200                 end;
17300                 close(output);
17400           end
17500           else writeln(tty,'change maxprocessors, current value is',
17600                        maxprocessors);
17700     end.
*
```

PROGRAM MVTEST

```
00100     PROGRAM MVTEST;
00200     CONST MAXPROCESSORS=5;
00300            maxv=14;
00400     VAR
00500            kount,kw:integer;
00600            reconf:integer;
00700            filename: packed array[1..8] of char;
00800            working: array[0..maxprocessors] of integer;
00900            MV:ARRAY[0..MAXPROCESSORS,0..MAXV] OF INTEGER;
00930                     (* THE MV ARRAY COLUMNS WILL BE USED IN E TO
00960                        FORM THE DIFFERENT COMBINATIONS OF ERROR
00990                        REPORTS REQUIRED FOR THE VALIDATION *)
01000            E:ARRAY[0..MAXPROCESSORS,0..MAXPROCESSORS] OF INTEGER;
01050                     (* E IS THE ARRAY REPRESENTING ERROR REPORTS *)
01100            A:ARRAY[0..1,0..MAXV] OF INTEGER;
01150                     (* A IS THE ARRAY FOR MARKING WHICH PROCS REPORT *)
01200     PROCEDURE ZERO;
01250        (* zero the e array *)
01300     VAR I,J:INTEGER;
01400     begin
01500            for i:=0 to maxprocessors do
01600                    for j:=0 to maxprocessors do
01700                            e[i.j]:=0;
01800     end;
01900     PROCEDURE MVINIT;
01950        (* initialize the mv and associated a arrays *)
02000     VAR I,J,K,L,M:INTEGER;
02100     begin
02200            for l:=0 to maxprocessors do
02300                    for m:=0 to maxv do mv[l.m]:=0;
02400            j:=0;
02500            for i:=0 to maxprocessors - 1 do
02600                    for k:=i+1 to maxprocessors do begin
02700                            mv[i.j]:=1;
02800                            mv[k,j]:=1;
02900                            A[0,J]:=I:
03000                            A[1,J]:=K;
03100                            j:=j+1;
03200                    end;
03300            for l:=0 to maxprocessors do working[l]:=0;
03312        (*
03325            working[0]:=1;
03337            working[1]:=1;
03350            working[kw]:=1;
03375        *)
03400     END;
03500     PROCEDURE DISP(VAR L.J:INTEGER);
03550        (* write the output file for use by GEXEC *)
03600     VAR I,K,M,S :INTEGER;
03700     begin
03800            kount:=kount+1;
03900            writeln(output,'proc',j:2,' outvoted; procs',
04000               a[0,l]:2,a[1,l]:2,' reporting. proc. 0 failure report
);
04100            for i:=0 to maxprocessors do write(output,working[i]);
04200            for i:=0 to maxprocessors do begin
04300                    m:=1;
04400                    s:=0;
04500                    for k:=0 to maxprocessors do begin
04600                            s:=e[i.k]*m+s;
04700                            m:=m*2;
04800                    end;
```
56 ;

```
0 4900                          s:=s + 256*kount;
0 5000                          write(output.s);
0 5100                  end;
0 5200          reconf:=reconf + 256*kount;
0 5203                  (*...for more than 1 proc. out, reconf should have
0 5206                      constants added to it:
0 5209                          1 - for one proc. out
0 5212                          3 - for two procs. out
0 5215                          7 - for 3 procs. out  *)
0 5300          writeln(output,reconf);
0 5400  end;
0 5500  PROCEDURE MATCH;
0 5600  VAR I,J,K,L,M:INTEGER;
0 5700  begin
0 5800          for l:=0 to maxv do       (*.. l is col. of mv*)
0 5900              for j:=0 to maxprocessors do begin
0 6000                  zero;
0 6100                  for i:=0 to maxprocessors do
0 6200                      e[i.j]:=mv[i.l];
0 6230          (* mark procs 0 and 1 excess disagreements here*)
0 6260                  e[4,0]:=1;
0 6290                  e[5,0]:=1;
0 6291                  e[4,2]:=1;
0 6292                  e[5,2]:=1;
0 6294                  e[4,1]:=1;
0 6298                  e[5,1]:=1;
0 6300                      disp(L,J);
0 6400                      end;
0 6500  end;
0 6600  (*MAIN PROCEDURE*)
0 6700  BEGIN
0 6800          writeln(tty,'2 processors tet, enter filename');
0 6900          readln(tty);
0 7000          read(tty,filename);
0 7100          rewrite(output,filename);
0 7200          kount:=0;
0 7300          writeln(tty,'enter reconf');
0 7400          read(tty,reconf);
0 7415      (*
0 7430          writeln(tty,'which additional. proc. out?');
0 7460          read(tty,kw);
0 7480      *)
0 7500          MVINIT;
0 7600          MATCH;
0 7700          close(output);
0 7800          writeln(tty,'file complete');
0 7900  END.
*
```

PROGRAM GEXEC

58

```
00100    PROGRAM GEXEC;
00200    (*This is the set of routines associated with the global
00300       executive *)
00400
00500    CONST
00600            MAXPROCESSORS=5;
00700            maxsubframe=50;
00800            threshold=2;
00900            maxbufs=1;
01000            errerr=1;
01100    TYPE
01200            processor=0..maxprocessors;
01300            procint=array[processor] of integer;
01400            procbool=array[processor] of boolean;
01500            buffer=0..maxbufs;
01600    VAR
01700            WORKING:PROCBOOL;
01800            FRAMECOUNT,CASENO:INTEGER;
01900            PREVOTE:ARRAY[BUFFER] OF PROCINT;
02000            RECONF:INTEGER;
02100
02200
02300    PROCEDURE BINPARS(VAR NUM:INTEGER);
02400    (*procedure to represent an integer as a 16 bit string *)
02500    var
02600            binr: array[0..15] of integer;
02700            tnum :integer;
02800            divis,i,j:integer;
02900            byte: packed array[1..20] of char;
03000    begin
03100            divis:=32768;
03200            if num>65535 then begin
03300            writeln(tty,.overflow.);
03400                    num:=num mod 65535;
03500            end;
03600            tnum:=num;
03700            j:=0;
03800            for i:=15 downto 0 do begin
03900                    if tnum div divis>=1 then begin
04000                            tnum:=tnum mod divis;
04100                            binr[i]:=1
04200                    end
04300                    else binr[i]:=0;
04400                    divis:=divis div 2;
04500                    j:=j+1;
04600                    if binr[i] =1 then byte[j]:=.1.
04700                            else byte[j]:=.0.;
04800                    if (i mod 4=0) then begin
04900                            j:=j+1;
05000                            byte[j]:=. .;
05100                    end:
05200            end;
05300            writeln(tty,byte);
05400            writeln(tty);
*
```

```
05500    end;
05600    PROCEDURE PREGEXEC;
05700    (*This procedure copies the least significant bits of the
05800     error reporter word bits into the most significant positions
05900       after checking the frame number *)

06000
06100    VAR
06200            excount:INTEGER;
06300            ERR:INTEGER;
06400            J,M:INTEGER;

06500
06600    begin
06700            for j:=0 to maxprocessors do begin
06800                    excount:=prevote[errerr,j] div 256;
06900                    err:=prevote[errerr,j] mod 256;
07000                    if excount=framecount then
07100                            prevote[errerr,j]:=257*err
07200                    else writeln(tty,.processor.,j:3,. excount mismatch.);
07300            end;
07800    end;

07900
08000    PROCEDURE ALTGEXEC;
08100    (*This is the alternate global executive*)

08200
08250    const    maxdiv=32;
08300    VAR
08400            RECONFA,DIVISOR,MULT,J,K,L,M:INTEGER;
08500            ERCOUNT:PROCINT;
08550            LAST:INTEGER;
08600    begin
08700            for j:=0 to maxprocessors do ercount[j]:=0;
08800                                            (*...initialize ercount*)
08900            FOR J:= maxprocessors downto 0 do
09000              if working[j] then begin
09100                                    (*...do for each error report*)
09150                    divisor:=maxdiv;
09200                    for k:=maxprocessors downto 0 do begin
09300                                            (*...do for each position of report*)
09400                            if j=k then last:=0
09500                             else last:=prevote[errerr,j] div divisor;
09550                            if odd(last) then ercount[k]:=ercount[k]+1 ;
09700                            divisor:=divisor div 2;
09800                    end
09850            end;
09900                                    (*...now write reconfa*)
10000            reconfa :=0;
10100            mult:=1;
10200            for l:=0 to maxprocessors do begin
10300                    if ercount[l]>=2 then reconfa:=reconfa+mult;
10400                    mult:=mult*2;
10500            end;
10600            writeln(tty,.alternate reconf word.);
10700            binpars(reconfa);
10800    end;
*
```

60

```
10900
11000      PROCEDURE GEXECTEST;
11100      (*Global Executive Acceptance test*)
11200
11300      TYPE
11400              ZERO_ONE=0..1;
11500      VAR
11600              DIVISOR,CHECK,I,J,SUM:INTEGER;
11700              FAILFLG:BOOLEAN;
11800              LAST_DIG:ZERO_ONE;
11900      begin
12000              divisor:=1;
12100              failflg:=false;
12200              for i:=0 to maxprocessors do begin
12300                                      (*...do for each position of report*)
12400              (*implement error word shifts here*)
12500                      sum:=0;
12600                      for j:=0 to maxprocessors do begin
12700                                      (*...do for each error report*)
12800                              last_dig:=(prevote[errerr,j] div divisor)
12900                                          mod 2;
13000                              if (not working[i]) or (i=j)
13100                                then last_dig:=0;
13200                              sum:=sum + last_dig;
13300                      end;
13400                      check:=reconf div divisor;
13500                      if odd(check) then begin
13600                              if(sum<2)and(working[i]) then failflg:=true
13700                      end
13800                      else if sum>=2 then failflg:=true;
13900                      divisor:=divisor*2;
14000              end;
14100              if failflg then altgexec
14200              else writeln(tty,.global Executive OK.);
14300      end;
14400      PROCEDURE INFILE;
14500      (*Read data from file input after main procedure has opened it*)
14600      var
14700              casename:packed array[1..40] of char;
14800              intrep:procint;
14900              k:integer;
15000
15100      begin
15200              readln(input,casename);
15300              for k:=0 to maxprocessors do read(input,intrep[k]);
15400              for k:=0 to maxprocessors do read(input,prevote[errerr,k]);
15500              readln(input,reconf);
15600              writeln(tty);
15700              writeln(tty);
15800              writeln(tty,casename);
15900              write(tty,.Case.,caseno:3,.  Enter framecount .);
16000              read(tty,framecount);
16100              writeln(tty);
16200              writeln(tty,.Failed processors.);
*
```

```
16300            for k:=0 to maxprocessors do begin
16400                    write(tty,intrep[k]:3);
16500                    if intrep[k]=1 then working[k]:=false
16600                     else working[k]:=true;
16700            end;
16800            writeln(tty);
16900            for k:=0 to maxprocessors do begin
17000                    writeln(tty,.error report for processor.,k:3);
17100                    binpars(prevote[errerr,k]);
17200            end;
17300            writeln(tty,.Reconfiguration Word.);
17400            binpars(reconf);
17500    end;
17600
17700    PROCEDURE PRELIM;
17800    (*Initial prompts and opening of data file*)
17900    var filename:packed array[1..8] of char;
18000    begin
18100            writeln(tty,.Global Executive Recovery Block Driver --.);
18200            writeln(tty,.Enter Data File.);
18300            readln(tty);
18400            read(tty,filename);
18500            reset(input,filename);
18600    end;
18700
18800    (*MAIN PROCEDURE*)
18900    begin
19000            prelim;
19100            caseno:=0;
19200            while not eof(input) do begin
19300                    caseno:=caseno+1;
19400                    infile;
19500                    preqexec;
19600                    qexectest
19700            end;
19800            writeln(tty,.Tests Complete.);
19900            close(input);
20000    end.
*
```

APPENDIX C.   DEMONSTRATION OF VALIDATION PROCEDURES

This appendix contains examples of output which demonstrate the manner in which the fault-tolerant software for the error reporter and the global executive were validated. Section C.1. describes the output from DRIVER used to demonstrate the correctness of the error reporter, and section C.2 describes the GEXEC output which showed the proper operation of the fault tolerant global executive.

C.1. Error Reporter Validation

Figure C.1 is the output generated by the DRIVER program using data for 1 processor out. A total of five "frames" (i.e. test cases) are shown. The first line is the abbreviated title "proc 1 exc undtctd err", which is the designation for processor no. 1 having an excess number of errors undetected by the primary error reporter. The next line shows the value of framecount and excount (which were taken to be the same for the cases shown here). The next item on the output is a table showing the number of errors counted by the voter, the error reporter output (0 = no excess disagreements, 1 = excess disagreements), and the working status (0 = not working, 1 = working) for each of the six processors. The following line shows the integer value of the error word including the frame count encoded in the 8 most significant bits, and immediately below it is the binary representation produced by procedure BINPARS (see appendix B).

Because the primary error report (contained in the file) was incorrect, the error reporter acceptance test invoked the alternate, which generated an error report whose integer value (not including the frame count) is shown on the next line and whose binary representation (including the frame count) is shown immediately below.

This particular case demonstrates that the acceptance test can detect failure of the primary error reporter to note an excess number of disagreements in processor 1 when no other processors have failed and when all are working. Succeeding cases shown in this output demonstrate that failure of the primary routine to detect excess disagreements for processors 2, 3, 4, and 5 when no processors have been retired or have become faulty in this frame. The entire validation sequence described in section 2.5 consists of performing a sequence similar to this for processors 0 through 6 when 1, 2, or 3 additional processors become faulty in the current frame and when 1, 2, or 3 other processors have been retired. Although these validations were performed, they are not included in this report for the sake of brevity.

63

err1   ready
            5 repetitions

proc 1 exc undtctd errr
frame no.   1execution 1

processor	voter error	error report	working
0	0	0	1
1	3	0	1
2	0	0	1
3	.	0	1
4	0	0	1
5	0	0	1

primary error word= 256
0 000 0001 0000 0000

alterate error reporter invoked
alternate error word=   2
0 000 0001 0000 0010

proc 2 exc undtctd err
frame no.   2execution 2

processor	voter error	error report	working
0	0	0	1
1	0	0	1
2	5	0	1
3	0	0	1
4	0	0	1
5	0	0	1

primary error word= 512
0 000 0010 0000 0000

alterate error reporter invoked
alternate error word=   4
0 000 0010 0000 0100

FIGURE C.1. Error Reporter Validation Output

64

proc 3 exc undtctd disgr
frame no. 3execution 3

processor	voter error	error report	working
0	0	0	1
1	0	0	1
2	0	0	1
3	4	0	1
4	0	0	1
5	0	0	1

primary error word= 768
0 000 0011 0 000 0000

alterate error reporter invoked
alternate error word= 8
0 000 0011 0 000 1000

proc 4 exc undtctd err
frame no. 4execution 4

processor	voter error	error report	working
0	0	0	1
1	0	0	1
2	0	0	1
3	0	0	1
4	6	0	1
5	0	0	1

primary error word= 1024
0 000 0100 0000 0000

alterate error reporter invoked
alternate error word= 16
0 000 0100 0001 0000

Figure C.1. (continued) Error Reporter Validation Output

65

```
proc 5 exc undtctd err
frame no.  5execution  5
```

processor	voter error	error report	working
0	0	0	1
1	0	0	1
2	0	0	1
3	0	0	1
4	0	0	1
5	3	0	1

```
primary error word= 1280
0000 0101 0 000 0000


alterate error reporter invoked
alternate error word=    32
0 000 0101 0 010 0000
```

Figure C.1. (continued) Error Reporter Validation Output

C.2. Global Executive

Figure C.2 shows an excerpt from the output generated by program GEXEC. Two cases are shown from an MVTEST generated file containing cases in which processor 1 is marked as having failed by processors 5 and 6, and 1 additional processor is marked for retirement in the reconfiguration word by the primary global executive. The first line shows the title of the case, i.e. "proc 0 outvoted; procs 0, 1 reporting" . Thus, processor 0 is marked as having excess disagreements by processors 0 and 1, and processor 1 is indicated as having excess disagreements in the error reports of processors 5 and 6. The second line of the output is the frame count check, which, in this case is matches the execution count so that PREGEXEC finds that all error reports are current.

The following line gives the configuration of the system, and shows that no processors are failed (i.e. retired). The following 6 output items are the binary representations of the six processor error reports. The error report from processor 0 is marking itself for retirement; the report from processor 1 agrees. No processors are indicated as faulty in the error reports of processors 2 and 3, but processors 4 and 5 indicate that processor 1 should be retired.

The next item is the reconfiguration word generated by the primary global executive. It indicates that processor 0 should be retired, and that the current frame count is 1 (in bit position 8). The global executive acceptance test detects an error, and invokes the alternate routine, which marks processor 1 for retirement as shown in the last output item.

This particular case demonstrates that the acceptance test can detect the error of simultaneously incorrectly marking a functional processor as for retirement (processor 0) and not detecting a failed processor (processor 1). The second case shown in figure C.2 shows that processors 0, 1, 4, and 5 all indicate that processor 1 should be retired, but that the primary reconfiguration word marks processor 0 for retirement. Once again, the recovery block can detect and correct the error.

Close to 2,000 cases of this type were run, and in order to reduce the amount of output, GEXEC was modified to show only the case title, whether or not a processor which should have been retired was still generating error reports, whether the primary global executive output was accepted, and if not, the value of the alternate acceptance test was shown. Figure C.3 shows the beginning of such an output for failure to detect one faulty processor when one other was retired. The first item on the page is the prompt generated by the modified GEXEC program for the data file name. The next items show that the reconfiguration word is given as 0 throughout the file (i.e. no processors are marked for retirement by the primary global executive in this set of test cases) and that processor 1 is indicated as not working. The set of possibilities generated within GEXEC did not exclude processors marking themselves for retirement or having not working processors generating error reports. Thus, the first test case of figure C.3, processors 0 and 1 marking processor 0 for retirement. Because this condition would not lead to the retirement of processor 0, the primary error word is correct. In the second case, processor 1 is indicated as having excess disagreements by processors 0 and 1. Because processor 1 should have been retired, this is possibly a serious condition, and

the global executive indicates that there may be a problem (by itself, the global executive can not diagnose and trace the problem) to the system in the message "retired processor working". In the third case, the error report from a retired processor along with only one other processor indicates that a third should be marked for retirement. This is not a sufficiently strong case for retiring processor 2, so the reconfiguration word is correct.

```
proc 0 outvoted; procs 0 1 reporting
Case   1   Enter framecount 1

Failed processors
   0   0   0   0   0   0
error report for processor   0
0000 0001 0000 0001

error report for processor   1
0 000 0001 0 000 0001

error report for processor   2
0 000 0001 0 000 0000

error report for processor   3
0 000 0001 0 000 0000

error report for processor   4
0 000 0001 0 000 0010

error report for processor   5
0 000 0001 0 000 0010

Reconfiguration Word ~
0 000 0001 0 000 0001

alternate reconf word
0 000 0000 0 000 0010
```

Figure C.2. Global Executive Validation Output

```
proc 1 outvoted; procs 0 1 reporting
Case  2  Enter framecount 2

Failed processors
  0   0   0   0   0   0
error report for processor  0
0000 0010 0000 0010

error report for processor  1
0 000 0010 0000 0010

error report for processor  2
0 000 0010 0000 0000

error report for processor  3
0 000 0010 0000 0000

error report for processor  4
0 000 0010 0000 0010

error report for processor  5
0 000 0010 0000 0010

Reconfiguration Word
0 000 001 1 0 000 0 001

alternate reconf word
0 000 0000 0 000 0010
```

Figure C.2. (continued) Global Executive Validation Output

```
reconf and workim held constant
Reconfiguration word (reconf)
0000 0001 0000 0000

Processor statuses; 0 working/ 1 failed
   0  1  0   0   0   0



proc 0 outvoted; procs 0 1 reportim
global Executive OK

proc 1 outvoted; procs 0 1 reporting
retired proc. workim
global Executive OK

proc 2 outvoted; procs 0 1 reportim
global Executive OK

proc 3 outvoted; procs 0 1 reporting
global Executive OK

proc 4 outvoted; procs 0 1 reportim
global Executive OK

proc 5 outvoted; procs 0 1 reportim
global Executive OK

proc 0 outvoted; procs 0 2 reportim
global Executive OK

proc 1 outvoted; procs 0 2 reportim
retired proc. workim
retired proc. workim
global Executive OK

proc 2 outvoted; procs 0 2 reportim
global Executive OK

proc 3 outvoted; procs 0 2 reporting
alternate reconf word
0000 0000 0000 1000

proc 4 outvoted; procs 0 2 reportim
alternate reconf word
0000 0000 0001 0000

proc 5 outvoted; procs 0 2 reportim
alternate reconf word
0000 0000 0010 0000

proc 0 outvoted; procs 0 3 reportim
global Executive OK

proc 1 outvoted; procs 0 3 reportim
retired proc. workim
retired proc. workim
global Executive OK

proc 2 outvoted; procs 0 3 reportim
alternate reconf word
0000 0000 0000 0100
```

Figure C.3. Global Executive (VALGEX) Validation Output

71

REFERENCES

AERO78

Aerospace Corp., <u>Fault Tolerant Software Study</u>, NASA Contractor Report No. 145298, Advanced Programs Division, Aerospace Corp., February, 1978

HECH76

H. Hecht, "Fault Tolerant Software for Real Time Applications", <u>ACM Computing Surveys</u>, Vol. 8, No. 4, p. 391, December, 1976

RAND75

B. Randell, "System Structure for Software Fault Tolerance", <u>IEEE Transactions on Software Engineering</u>, Vol. SE-1, No. 2, p. 220, June, 1975

1. Report No. 165874	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle Fault Tolerant Software Modules for SIFT		5. Report Date April 1981
		6. Performing Organization Code
7. Author(s) Myron Hecht and Herbert Hecht		8. Performing Organization Report No. SoHaR-TR-81-04
		10. Work Unit No.
9. Performing Organization Name and Address SoHaR, Inc. 1040 South LaJolla Ave. Los Angeles, CA 90035		11. Contract or Grant No. NAS1-15428
		13. Type of Report and Period Covered Final Engineering Report
12. Sponsoring Agency Name and Address NASA-Langley Research Center Hampton, VA 23665		14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

The Recovery Block technique for fault-tolerant software was applied to the operating system of the SIFT fault-tolerant computer. The original operating system serves to implement algorithms for hardware fault tolerance, and has been subjected to rigorous logical analysis, but does not incorporate redundancy for tolerating its own faults, (e.g., programming errors). Fault-tree analysis was used to validate acceptance tests for application to alternate (redundant) versions of several operating system functions. The tests and several alternate program versions were implemented in Pascal. This application of the Recovery Block technique was more difficult than usual because the subject program was essentially logical in nature. Some limitations were encountered in constructing alternate routines that are truly independent of the primary ones and also of the acceptance test.

17. Key Words (Suggested by Author(s)) Fault-Tolerant Software, Recovery Blocks, Fault-Tolerant Computers, Fault-Tree Analysis, Software Validation	18. Distribution Statement

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

N-305