ESD-TR-76-160

PROBABILISTIC MEASURES OF COMPROMISE

Honeywell Information Systems, Inc.
Federal Systems Operations
7900 Westpark Drive
McLean, Virginia 22101

(10)

January 1976

Approved for Public Release;
Distribution Unlimited.

Prepared for

DEPUTY FOR COMMAND AND MANAGEMENT SYSTEMS
ELECTRONIC SYSTEMS DIVISION
HANSCOM AIR FORCE BASE, MA 01731

DEFENSE ADVANCED RESEARCH PROJECTS AGENCY
1400 WILSON BOULEVARD
Arlington, VA 22209
ARPA Order No. 2641

LEGAL NOTICE

OTHER NOTICES

This technical report has been reviewed and is approved for publication.

Roger R. Schell
ROGER R. SCHELL, Major, USAF
Techniques Engineering Division

William R Price
WILLIAM R. PRICE, Captain, USAF
Techniques Engineering Division

FOR THE COMMANDER

Stanley P. Dereska
STANLEY P. DERESKA, Colonel, USAF
Chief, Techniques Engineering Division
Information Systems Technology
Applications Office

(19) **REPORT DOCUMENTATION PAGE**		READ INSTRUCTIONS BEFORE COMPLETING FORM
(18) 1. REPORT NUMBER ESD-TR-76-160	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
(6) 4. TITLE (and Subtitle) PROBABILISTIC MEASURES OF COMPROMISE.		5. TYPE OF REPORT & PERIOD COVERED
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) George A. Kilgore (10)		8. CONTRACT OR GRANT NUMBER(s) (15) F19628-74-C-0193, ARPA Order [redacted] 2641
9. PERFORMING ORGANIZATION NAME AND ADDRESS Honeywell Information Systems, Inc. Federal Systems Operations 7900 Westpark Drive, McLean, VA 22101		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS (12) 62p.
11. CONTROLLING OFFICE NAME AND ADDRESS Deputy for Command and Management Systems Electronic Systems Division Hanscom Air Force Base, MA 01731		12. REPORT DATE (11) Jan[redacted] 1976 13. NUMBER OF PAGES 62
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Defense Advanced Research Projects Agency 1400 Wilson Boulevard Arlington, VA 22209		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

(9) Final rept.,

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Secure Communications	Minicomputer
Simulators	Computer Security
Design Verification	Fault Tolerance
Probability	

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report describes the results of a trade-off study in which candidate methodologies for verification of a secure minicomputer hardware design were evaluated. Three verification elements appropriate to the problem were developed: (1) probabilistic measurement of security compromise due to hardware failure, (2) logic design certification, and (3) production hardware security criteria. The trade-off of techniques included evaluations of technical characteristics and cost effectiveness of both manual and computer

(Continued on back side.)

409 690

20. ABSTRACT (Continued)

aided analysis techniques. The architectures for two computer logic design simulators are described and evaluated. This report contains recommended verification methodologies suitable for a MULTICS compatible security front-end processor.

This report has been prepared under Air Force Contract F19628-74-C-0193. This is a final report on Secure Communications Processor (SCOMP) Hardware Verification, CDRL A019. The requirements to which this report is responsive are found in the Statement of Work for Secure MULTICS Design, Development and Certification, dated 22 June 1975, Section 4.8.6, first paragraph.

SCOMP HARDWARE VERIFICATION METHODOLOGIES

FINAL REPORT

TABLE OF CONTENTS

Page

I. INTRODUCTION 5

1.1 Purpose of the Study 5

1.2 Approach to SCOMP Hardware Verification Analysis 5

1.3 Observations on Sufficiency of Verification Methodologies 6

II. PROBABILISTIC MEASURE OF SECURITY COMPROMISE 7

2.1 Objectives and Criteria for Probabilistic Analysis 7

2.1.1 Failure Induced Security Compromises 8

2.1.2 Baseline SCOMP Hardware Configuration 9

2.1.3 Techniques for Numerical Probability Assessment 14

2.1.3.1 System Reliability Modeling 14

2.1.3.2 Failure Rate Basis for Probabilistic Analysis 14

2.1.3.3 Periodic System Health Checking Software 15

2.1.3.4 Circuit Failure Modes 20

2.2 Review of Hardware Failure Effects Analysis Methods 20

2.2.1 Probabilistic Measure - Manual Analysis 21

2.2.1.1 Fault Implantation 21

2.2.1.2 Failure Modes and Effects Analysis (FMEA) 23

2.2.2 Computer Fault Simulators 25

2.3 Recommended Probabilistic Measurement Methodology for SCOMP 30

III. SCOMP HARDWARE CERTIFICATION 31

3.1 Objectives and Criteria for Hardware Certification 31

Page

III. SCOMP HARDWARE CERTIFICATION (Continued)

3.2 Hardware Design Verification Analyses 31

3.2.1 Hardware Design Verification Descriptions 32

. Manual Analyses 32

. Instruction Simulator Description 32

. Register Transfer Level (RTL) Simulators 38

. Logic Circuit Simulators 42

. Boolean Logic Simulators 42
. Logic Device Simulators 42

. Security Proofs for Operating System Software 43

3.2.2 Recommended Hardware Design Analysis for SCOMP 44

3.3 Hardware Verification Tests 44

3.3.1 Prototype and Production Logic Test Criteria 45

3.3.2 Design Verification Testing 45

3.3.3 Acceptance Tests for Production Hardware 46

IV. CONCLUSIONS AND RECOMMENDATIONS 49

BIBLIOGRAPHY OF REFERENCES 51

APPENDIX A - A Formalism for Description of SCOMP Security Compromises 53

SECTION I

INTRODUCTION

1.1 Purpose of the Study

The objectives of this analysis were to examine available computer hardware verification methodologies applicable to a Secure Communications Processor (SCOMP) and to recommend techniques which accomplish each verification element. Two major verification elements were identified for analysis. They are:

- Probabilistic measures analysis of security compromise induced by hardware failure. For this element, the impact of unreliability in the physical hardware on Secure Communications Processor performance must be analyzed and quantified.

- Certification that the SCOMP hardware accomplishes the performance requirements of its design specifications. For this element, the hardware certification criteria for design analysis, design testing and production product control must be selected and specified.

The objectives were accomplished.

1.2 Approach to SCOMP Hardware Verification Analysis

This study was accomplished in two phases.

In the first phase, a general investigation of the form and character of available analytic tools and process techniques applicable to hardware verification was conducted. The investigation served to establish the specific tasks appropriate to accomplishing the probabilistic measurement analysis and the certification of the SCOMP hardware design and physical product. Additionally, the range of the available methodologies for each task which should be a candidate for detail study and/or trade-offs was also determined in the first phase. The first phase of this study culminated in October 1975 with the issuance of A Brief Technical Note on SCOMP Hardware Verification Methodologies. Contained in the note were descriptions of the work elements necessary to achieve probabilistic measurement and hardware certification and an overview of candidate methodologies which were to be examined in trade-off studies in the second phase of the study.

In the second phase, the methodology trade-offs described above were performed and suitable criteria were selected.

1.2 Approach to SCOMP Hardware Verification Analysis (Continued)

The trade-off results and recommendations are contained in this final report. Where further trade-offs were inappropriate to a specific task, the task criteria have been developed and specified. These criteria are contained in the appropriate Detail Specifications (D.S. Part I) for the Security Protection Module and the MULTICS Interface Unit, Quality Assurance Provisions sections. Paragraph 3.3 of Section III of this final report also contains a summary of these criteria.

1.3 Observations on Sufficiency of Verification Methodologies

The course of this study has led us to a set of conclusions which either define or scope specific SCOMP hardware verification tasks. In arriving at these conclusions, we have employed analytic, and sometimes subjective, tests on candidate methodologies. Stated generally, these tests are:

- Appropriateness of the task to achieving Project GUARDIAN objectives.
- Sufficiency of the methodology for accomplishing a defined technical task.
- Timeliness of the methodology for application to the design of a Secure Communications Processor.
- Cost efficiency of the methodology, consistent with technical sufficiency and timeliness.

In the specific circumstances where trade-off studies have been performed on candidate probabilistic measure and hardware design analysis methodologies, subjective views of technical sufficiency and cost efficiency were necessary. It is important to note that reasonably clear upgrading paths are identifiable in the event that they should be required at some later date. These are discussed together with the recommendations in Sections II and III.

SECTION II

PROBABILISTIC MEASURE OF SECURITY COMPROMISE

2.1 Objectives and Criteria for Probabilistic Analysis

The objectives of the probabilistic measure analysis are threefold:

a. To establish the numerical probability that any SCOMP hardware failure will induce a security compromise condition which remains undetected. The probability desired is an upper bound on failure probability rather than its exact value.

b. To insure the hardware design effectiveness as it addresses the problem of detecting security impacting device failures.

c. To determine the need for and frequency of SCOMP system exercise by "health checking" diagnostic software to supplement the hardware design.

Probabilistic measures of security compromise due to undetected computer hardware failures can be developed analytically using either manual or computer-aided methods. Additionally, it is feasible to employ a physical fault implantation evaluation test sequence which yields sufficient failure effects data to establish a measure of security compromise.

All classes of probabilistic measurement methods considered herein result in a single indicator of design effectiveness in precluding security compromise, a probability of security compromise due to hardware failure per unit time. The SCOMP design goal for the probability that a security compromise due to hardware failure will occur is less than 0.000001 per hour. Restated, this equates to a steady state secure operation 99.9999 percent of the time. The objective of the probabilistic analysis is to establish an upper bound, rather than a precise value, for the probability of compromise.

Three prerequisite criteria must be established prior to proceeding with any detail review of candidate probabilistic measurement methodologies. The first, and most important criterion, is the existance of a definition of the SCOMP operating conditions which represent a security compromise. Second, a baseline SCOMP hardware system configuration is necessary to scope the analysis task size for trade-off purposes. Third, specific techniques for numerical probability assessment must be established. Failure to develop these tools may result in evaluation of candidate probabilistic measure methodologies in terms of the entire Secure Communications Processor instead of smaller, more manageable modules. This in turn could cause a methodology to be discarded because the technical or economic factors grow exponentially instead of linearly with module size.

2.1.1 Failure Induced Security Compromises

Hardware failure induced computer security compromises for the SCOMP stated in terms which can be directly correlated with specific hardware mechanizations are essential to the probabilistic measure analysis. Because the intended utilization is to establish the yardsticks by which security responsibilities of specific hardware functions are measured, the hardware failure tabulations must be correlatable to individual hardware elements such as functional circuit interfaces and registers.

Initially, it appeared that a list of security compromises could be assembled easily through inspection of the problem using the SCOMP architecture specification and hardware functional diagrams. Just such a list is shown as Table I.

Table I is presented in three parts:

- Faults outside the SPM in devices having complex functional subsystems but within a front end processor security perimeter.
- Faults inside the SPM hardware within functional SPM subsystems.
- Detail of control and power distribution faults outside the SPM as seen at the bus.

These are identified as Parts 1, 2 and 3 of Table I, respectively.

While we do not believe that the technique of using fault tables should be abandoned altogether, the rather obvious deficiencies of the example were a clear indication that a more rigorous approach should at least be explored. A readily available alternate method for developing the desired tabulation of security compromises for a digital computer was not found. An attempt at structuring a suitable formalism which would result in the desired tabulation was performed. By taking a functional view of the SCOMP system (both hardware and software), a more precise and certainly more rigorous determination of the results of any hardware malfunction can be made. This approach is illustrated in a partially completed example in Appendix A. It is unnecessary at this time either to proceed further with the formalism or to refine Table I. The insight provided by the process of their development to this point is sufficient to support the probabilistic measures methodology trade-offs.

2.1.2 Baseline SCOMP Hardware Configuration

A representative SCOMP configuration has been determined necessary to size the hardware certification and the probabilistic measures task. The baseline SFEP (Secure Front End Processor) shown in Figure 2.1.2-1 is intended to illustrate hardware elements and functional interconnections which mechanize SCOMP architecture.

Initially, this diagram, and supporting functional interface diagrams, have been used to assess the scope of and the modularity with which the probabilistic measures task and the hardware design certification task could be approached. Figure 2.1.2-2 illustrates the Central Processor-Security Protection Module (CPU-SPM) dedicated interface in this context. Another utility of the functional diagrams is determination of circuit complexity of major functional elements. This was useful to the simulator trade-offs (see paragraph 2.2.2, especially Table V).

These functional block diagrams of the Secure Communications Processor architecture are an effective tool used to identify functional element interfaces within the SCOMP and, together with the tabulation of compromises, the security responsibilities of signal sets within functional interfaces.

Resultant from our study of the relationships shown on the diagrams and a review of the specifications is a Security Failure Model represented in Figure 2.1.2-1 as the SFEP Security Perimeter. This perimeter defines the approximate analysis boundary for the probabilistic measures analysis task.

TABLE I

	Subsystem Elements in Analysis	Security-Related Processing by the Subsystem Element
1.	Address bus & checks and memory module address circuitry	Absolute addresses only, if the new address is out of user's space
	Address control	Absolute addresses only, if stuck at Logic "1" failure (time-out if control is stuck at Logic "0")
	Data bus & checks	Only when passing descriptor-parts, critical state information or absolute device identification, and classes of errors as on A-bus
	Data control	Stuck at "1" failure; either Absolute bus or Virtual bus
	Other bus controls	For modules within the security perimeter (see Item 3)
	Interrupt network priority-resolved	Only for SPM security fault condition being transformed to other fault condition
	Timing information	If withheld from kernel, or if a "unique-name" generated by the clock is repeated
	Power and ground	For modules within the security perimeter (see Item 3)
2.	Inside SPM	
	Associator identifying descriptor	Only for false "hit" indication in the SPM Cache
	Descriptor permission information: permission checking logic and storage	Only for false extension of permission
	Address within descriptor	If the altered address base is out of the user's space
	Limit within descriptor	If the limit is effectively increased, and overlaps another user's resource
	Current user id, operating ring	Always potential breach regardless of system operating mode

TABLE I

(Continued)

3. System Considerations

- Device identification duplicated due to hardware fault

  (double routing of message)

- Device fails to recognize its identifier

  (data link cannot be established)

- Direction bit from device on bus (including SPM) stuck at 1 = output or 0 = input

  (transaction is one way only to device)

- Tie breaking network fault which permits confusion of control bus protocols

- Function code bit error confuses read/write

- Status word bits 1-5 inoperative

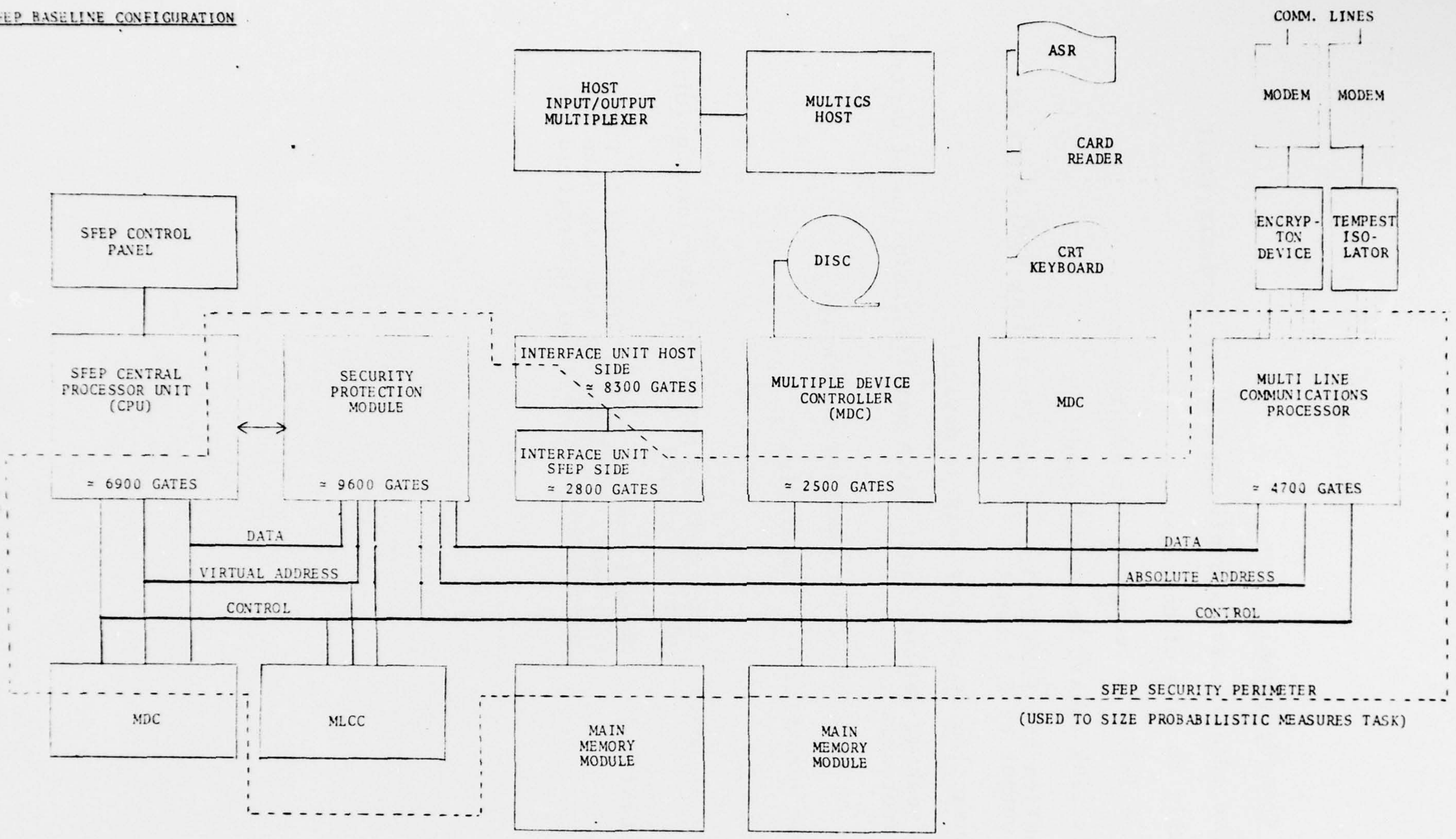
  (stuck off)

- Interrupt Level - Interrupt may appear to have lower priority than it should

- Byte Confused - Word format on memory transfers to bus is confused, resulting in address, data or descriptor bits being misinterpreted

FIGURE 2.1.2-1

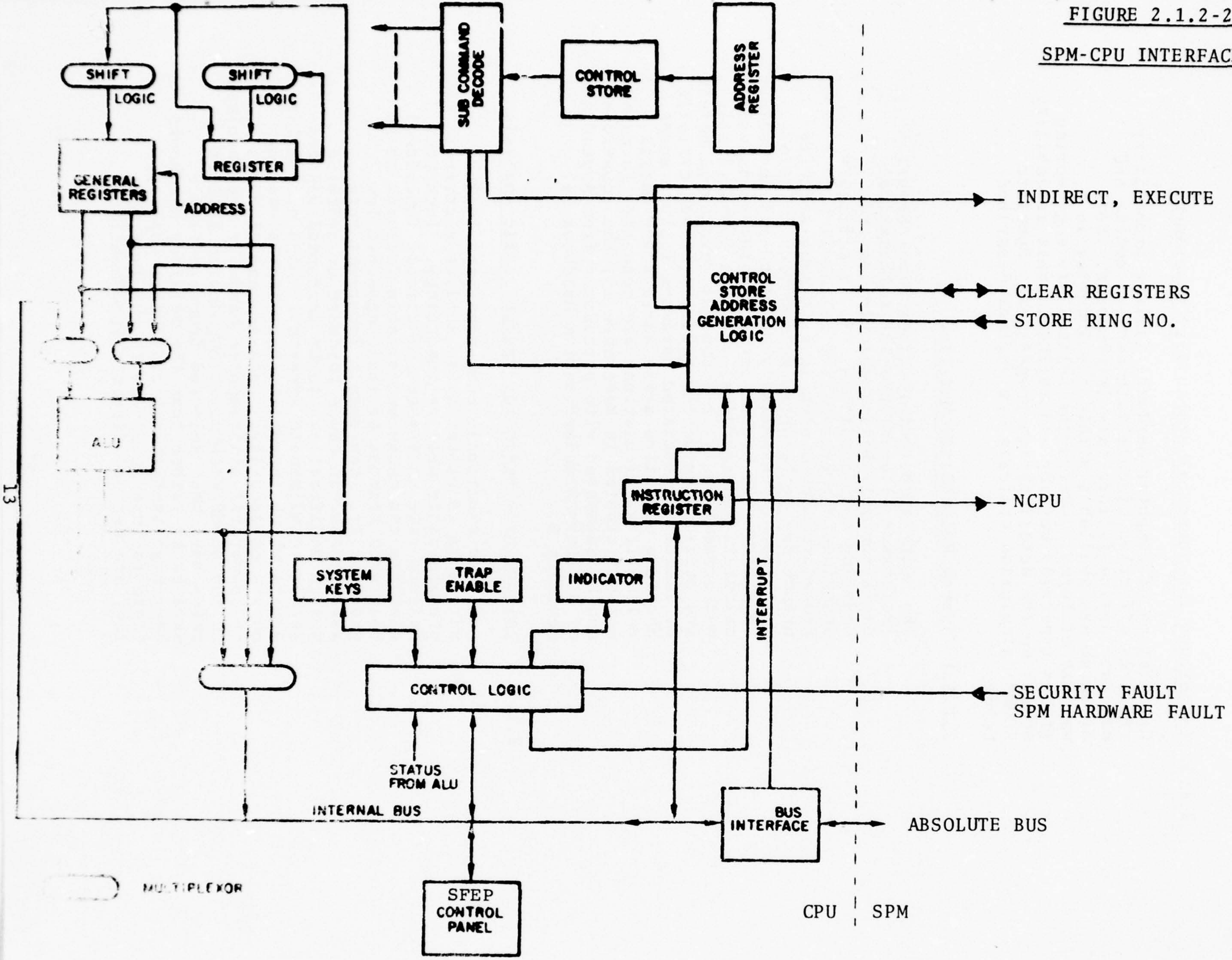
SFEP BASELINE CONFIGURATION



NOTE:
1. Gate counts exclude memory elements whose functionality can be readily simulated.
2. Device adapters used with MDCs are configuration dependent and not included in the above gate counts.

FIGURE 2.1.2-2

SPM-CPU INTERFACES



2.1.3 Techniques for Numerical Probability Assessment

Calculation of computer hardware failure probability requires effective circuit mathematical models and accurate device failure rate information. Both of these are available within the industry in a variety of forms. Contractor reliability engineering groups typically develop and refine circuit reliability models as the detail design progresses. Specific SCOMP calculation criteria are further described below.

2.1.3.1 System Reliability Modeling

The SCOMP Architecture Study Final Report (Reference 2) describes the mathematical basis for reliability calculation and modeling considerations. It specifically describes the probability calculation procedures to be used in assessing security breach due to hardware malfunction. These criteria are essentially complete and sufficient to perform the probabilistic measures analysis calculations regardless of which probabilistic measurement analytic techniques evaluated herein are utilized to define security breach criteria or identify associated hardware failure modes. Should a situation arise where state matrix reliability calculations are necessary (a case not anticipated in Reference 1), the equations can be augmented with procedures from Appendix A of Military Standardization Handbook 217, Revision B.

2.1.3.2 Failure Rate Basis for Probabilistic Analysis

Accurate electronic component part failure rate data is essential to achieve a correct probabilistic measure numerology. Incorrect failure rate assessment of circuit functional elements can cause undesirable failures to be tolerated because of their apparent low probability. Conversely incorrect rate assumptions can cause hardware or software design modifications to be performed unnecessarily to eliminate apparently high probability events which in reality have little bearing on system security. Because many elements of commercial minicomputer hardware are involved in the SCOMP mechanization, device failure rates have been selected from experience data banks rather than the military handbooks. Very high statistical confidence supports these failure rates due to the fact that they are derived directly from monitored

2.1.3.2 Failure Rate Basis for Probabilistic Analysis (Continued)

system installations containing practically identical hardware.

The microcircuit rates are the most critical to the calculation process. They are listed in Table II and specified in the SPM and MSIU detail specifications.

2.1.3.3 Periodic System Health Checking Software

The design of a Secure Communications Processor requires particular attention to the placement of hardware fault detection circuits if the probability of undetected security compromises induced by failures is to be minimized. Parity circuits, because they are electrically straightforward and economical to implement, are the most commonly used form of hardware fault detection. This additional circuitry, however, can itself fail undetected, creating a potential system security problem.

Failure of a parity checking circuit, regardless of where it occurs, does not create security breach. Generally, two separate failures are then required for a breach to be induced. The probability of two or more undetected failures occurring in any short time interval can be quite small. Nevertheless, after some arbitrarily long elapsed time, the failure probability will increase beyond any acceptance limit we set for secure computer performance. The relationship is a simple one:

$$P = 1 - e^{-(\lambda_1 \cdot \lambda_2)t}$$

Where:

P = the probability of undetected security compromise of the system.

λ_1 = the failure rate of the parity checking circuit element.

λ_2 = the failure rate of the circuit whose performance is being checked.

t = the total time that the secure computer has been used to process secure data.

2.1.3.3 Periodic System Health Checking Software (Continued)

It should be obvious that added fault checking hardware is not a perfect solution to the problem of insuring secure operation in the presence of failure. By extension of the above formulae, we can delay compromise by checking the hardware with redundant parity hardware. This approach can extend the time to any acceptable compromise probability limit out beyond the life of the computer and, hence, solve the whole problem. Unfortunately, redundant parity circuits aren't either straightforward or economical in their implementation, particularly if many circuits require parity checking.

An effective solution to the dilemma is the institution of periodic software checks, whose function is to exercise either the circuit element having security related functions and/or its parity check. The probability of undetected compromise resulting from hardware failure can be reduced to a level which can be neglected provided at least one of the two failure conditions is checked by the system software periodically.

2.1.3.3.1 Calculation Procedure

For any given circuit, with security processing, the probability of undetected failures per hour, in the presence of periodic diagnostics, can be developed using the following five steps:

1. Single IC MSI typical failure rate:

$$\lambda = 0.05 \times 10^{-6}$$

2. Probability of parity chip (one MSI circuit) failed:

$$P_P = 1 - e^{-\lambda t}$$

3. Probability of single bit failure of N bit word being checked:

$$P_W = 1 - e^{-N\lambda t}$$

4. Frequency of system "health check" software diagnostic of either the circuit or its parity:

f = number of checks per hour

2.1.3.3.1 Calculation Procedure (Continued)

5. Probability of both 2 and 3 simultaneously failed per hour:

$$P = (P_P * P_W)/f \simeq (N*10^{-14})/f$$

2.1.3.3.2 Example of Health Checking Applied to a Minicomputer with Parity

For a minicomputer complex, it may be safely presumed that less than 100 MSI microcircuits are dedicated to parity generation or checking. Hence, the total probability of all such occurrences is expected to be less than:

$$0.5* \sum_{i=1}^{i=m} N_i * 10^{-14}/f \text{ hours (from Equation 5)}$$

Where M is the number of parity circuits and N_i is the word length of the data checked by the ith parity circuit. In minicomputers, N_i is typically small (32 or less). Semiconductor memory matrix element (RAM) failure rates are approximately an order of magnitude greater than our example. However, the total probability for a SCOMP type minicomputer is still less than $10^{-10}/f$ per hour. This calculation is, of course, oversimplified in that it assumes the ability of a periodic system diagnostic to exercise every circuit or its parity check.

For our example, we can derive a first order approximation of the probability of undetected compromise using the formula:

$$M \cdot N \cdot \lambda_1 \cdot \lambda_2 \cdot /f$$

Where:

M = the total number of parity circuits.

N = the maximum bit length of any word whose parity is checked.

λ_1 = the failure rate of the parity circuit

λ_2 = the failure rate of the circuit generating the word whose parity is being checked.

2.1.3.3.2 Example of Health Checking Applied to a Minicomputer with Parity (Continued)

f = the frequency with which software exercises either the word or its parity.

Using: $M = 100$ $N = 32$

$\lambda_1 = .5 \times 10^{-6}$ $\lambda_2 = 10^{-6}$

The probability becomes $1.6 \times 10^{-9}/f$.

If our maximum acceptable probability is 10^{-6}, or .00001, then f must be less than 625 hours.

TABLE II

FAILURE RATES FOR PROBABILISTIC ANALYSIS

Microcircuit Device Type	Failure Rate (Per 10^6 Hours)
SSI, less than 20 gates	0.03
MSI, 20 - 100 gates	0.05
LSI, greater than 100 gates	0.1
Bipolar memory, 256 bit RAM	0.3
MOS memory, 4096 bit RAM	1.0

2.1.3.4 Circuit Failure Modes

In addition to failure rate data on individual logic circuit elements, it is necessary to specify the circuit failure modes which will be employed in the assessment of hardware failure effects. Essentially, two classes of failures cover the logic; gate failures and flip-flop failures. The modes within these classes are stated in Table III.

TABLE III

LOGIC FAILURE MODES

1 - Gate Functions

- Outputs failed to logic one or zero
- Individual inputs failed to logic one or zero

2 - Flip-Flop Functions: Output Terminals Q = Normal
$\overline{Q}$ = Inverted

- Set or Reset failed to logic one or zero
- Data input failed to logic one or zero
- Q output failed to logic one or zero (without affecting $\overline{Q}$)
- $\overline{Q}$ output failed to logic one or zero (without affecting Q)
- Input failed to Q and $\overline{Q}$ without regard to clock

2.2 Review of Hardware Failure Effects Analysis Methods

Candidate manual and computer simulation analysis methodologies identified during Phase 1 of this study are addressed in detail in the section. Relative cost, task complexity and confidence data are discussed to facilitate a selection.

All candidate analytic techniques which support probabilistic measurement serve one purpose:

. To identify specific circuit elements which have failure modes that result in undetected security compromise of the system.

It is only when these specific physical points have been isolated that numerical assessment, as described in Reference 1 and supplemented by paragraph 2.1.3, can begin.

2.2.1 Probabilistic Measure - Manual Analysis

Manual circuit reliability analysis techniques are well established in the electronics equipment industry. Some of these failure modes analysis techniques are readily adaptable to problems such as the SCOMP. There are two major classes of manual techniques; fault implantation and failure modes and effects analysis (FMEA).

Fault implantation is a physical test technique where individual failures (shorts or opens) are inserted in the hardware and the resultant effects on performance assessed. Obviously, at least prototype functional modules must be available to use this technique.

Failure modes and effects analyses are a standard tool employed by Reliability and Systems engineers in both the large computer and Aerospace industries. The level of detail to which such analyses are conducted are, however, subject to substantial variation which affects both cost of and confidence in the analysis output. The restrictions of these analyses to the subset of hardware failure modes which induce security compromise is a trivial change from the original intended purpose of FMEAs.

2.2.1.1 Fault Implantation

Fault implantation tests can be employed to evaluate a computer system's actual responses in the presence of a simulated hardware failure. The available nodes at which short circuit or open circuit conditions may be inserted include:

- Connectors
    - pin-to-pin or pin-to-case shorts
    - individual pin open circuits
    - entire connector unmated
- Electronic Components or Modules
    - individual leads open circuited (including power and return terminals)
    - inputs or outputs shorted to return or to each other

It is also possible to insert series or parallel resistance, capacitance and inductance and even to inject currents of the above nodes. By so doing, a wide variety of parameter shifts, leakages and stray inductance or capacitance may be simulated.

To effect a fault implantation test requires functional computer hardware; preferably of a geometry closely resembling the final product configuration. Also necessary is sufficient test equipment and operating system software to mechanize an operating unit. Lastly, and very important to the success of the test, a representative computer test program which exercises as many system functions as possible is required. It is desirable (but not mandatory) to know beforehand which computer circuit nodes and which failure modes are of interest. This knowledge can cut down the amount of work involved substantially.

Given that the prerequisites stated above are satisfied, the test may begin. The duration of the test might range from several days to several months dependent on the nature of the test program, the number of nodes to be failed, and the number of failure modes to be simulated for each node. A scenario in which the fault implantation test could be accomplished would be as follows:

> Open and short failure modes would be individually failed for nodes of interest (as determined from the tabulation of security compromises, paragraph 2.1.1) using a prototype SPM and Interface Unit in a ruggedized minicomputer chassis. The system would be configured as a front end processor. A sample test routine developed on an instruction simulator developed separately would be used to exercise the system. * The probability of each node failure which resulted in a compromising change in performance which was not detected would be calculated from reduction of a data dump of stored variables.

* Preliminary KERNEL software would be utilized.

Assuming that an ongoing prototype program existed, the cost of this testing could be as little as a few man months of effort. While costs are attractive, there is little else to recommend it. The advantages of hard test data are offset by a long list of disadvantages. Among these are:

2.2.1.1 Fault Implantation (Continued)

1. The test program element in execution at the instant of fault implantation may not result in a compromise while a subsequent test may. Multiple tests and special test routines developed for this test would be required to overcome this.

2. Fault implantation is not timely for hardware proofing or analysis purposes, since most major design decisions are completed by the time the prototype becomes available.

2.2.1.2 Failure Modes and Effects Analysis (FMEA)

2.2.1.2.1 FMEA - General Description

FMEAs are a form of design analysis whose purpose is to insure that all system level failure effects which result from probable hardware failure modes are known. The FMEA permits assessments to be made of the design, which may result in minimizing the impact or elimination of those failure modes considered undesirable through circuit redesign. Ideally, an FMEA should be accomplished in parallel with the detail circuit design to be most efficient, though the pace of many military hardware developments often precludes this. For the SCOMP, the undesirable system level failure effects are those system operating states which result in security compromise.

In performing the analysis, existing design documentation (including block diagrams, circuit schematic diagrams and hardware performance specifications) is used. The analysis consists of a systematic review of this documentation to obtain an ordered understanding of the following factors:

1. The function of each hardware functional item being analyzed (brief description).

2. Possible failure modes of each item (an itemization).

3. Effects on item operation and system interfaces of all failure modes (an itemization).

4. Causes of each failure mode (an itemization).

2.2.1.2.1 FMEA - General Description (Continued)

5. Probability of occurrence of each failure mode (calculated estimate).

2.2.1.2.2 FMEA Analysis Detail

The scope of the FMEA is determined both by the complexity of the hardware being analyzed and the level of detail to which the analysis is conducted. Four different levels of detail are generally recognized. Certain very sophisticated equipments may require several of these analyses, or conceivably all of them.

1. Functional Level FMEA in which circuit interface signal groups are analyzed for their interaction in the presence of a postulated failure within the functional element. In this sense, elements include CPU, Memory, device controller, SPM, MSIU, etc. The signal groups are bus data lines, address lines, control lines, power distribution, and SPM-CPU interfaces.

2. Part Level FMEA in which failures at terminals of individual circuit elements (i.e., microcircuit output pins) are analyzed for their impact on functional element performance. The circuit element failures (shorts-opens) are postulated to occur due to malfunction within the circuit. This FMEA is a second level of detail supporting (1) above.

3. Single Failure Analysis (SFA) iterates (1) and (2) above another step into the workings of complex circuit elements. The SFA is employed where LSI elements containing many hundreds of gates are involved, such as with microprocessor chips. SFA is typically reserved for space mission equipment and certain classes of COMSEC equipment involving key generators and related decrypting equipment.

4. Piece Part Mechanical FMEA is very similar to SFA but is more concerned with the circuit element geometry and its placement in the functional element assembly. A piece part FMEA would be used only to insure that

2.2.1.2.2 FMEA Analysis Detail (Continued)

electrical circuit redundancy was not reduced by part characteristics or assembly factors. Mechanical FMEA considerations include using dual transistors as a redundancy switch where a single mechanical failure could easily disable supposedly independent electrical circuits.

2.2.1.2.3 FMEA Evaluation for SCOMP

For SCOMP, a system view of failure modes and effects is desirable to accomplish a probabilistic measure. A baseline SFEP system, such as is illustrated in Figure 2.1.2-1, is sufficiently general to conduct meaningful analysis. It is obvious that the FMEA results must be stated in terms which apply to some specific configuration. This is not a serious drawback due to the bus oriented structure of the SCOMP minicomputer. Table IV below shows typical man hour costs for the candidate techniques based upon the baseline SFEP configuration.

TABLE IV

MANUAL FMEA COST FACTORS

FMEA Type	Extent of Analysis	Analysis Effort (MM)
1. Functional	15 Functional Units 10 Interfaces Each Unit	3
2. Part Level	3000 Parts 3 Failure Modes Per Part	26
3. Single Failure Analysis	20 LSI Types	25
4. Piece Part Mechanical	3000 Parts	2

2.2.2 Computer Fault Simulators

2.2.2.1 Fault Simulators - General

Digital fault simulators are available in a variety of well developed forms. Generally, they are structured to evaluate circuit stimulus-response characteristics for the purpose of generating fault detection

2.2.2.1 Fault Simulators - General (Continued)

tests for automatic tests and diagnostic dictionaries. Typically, such simulators consist of a collection of computer programs which analyze digital networks so as to perform the following functions.

1. Test Generation

   a. Generate stimulus and response capable of detecting all functional faults.

   b. Overlay stimulus whose functions can be performed simultaneously.

   c. Provide an accurate worst-case time analysis simulation, initializing the network first to all unknown states (X), so that the response 0s, 1s, and Xs may accurately reflect possible races, X-propagation, and initialization shortages, thus obtaining good test accuracy and repeatability.

   d. Utilize a criticality trace technique to determine for each response pattern/pin the set of failures which would cause that pin to fail. Reduce and process this information to provide a high-resolution fault isolation file or "fault dictionary."

   e. Given, in any specific test case, the set of patterns/pins which failed the stimulus-response tests, utilize the fault dictionary to determine and print out the most probable faults.

2. Design Verification

   a. Utilize the accurate simulator to verify that the network does in fact perform its intended functions. If not, utilize the fault-isolation capability to determine why not.

   b. Utilize the simulator's accurate worst-case timing analysis to eliminate all possible races, due either to close timing or to transient spikes, so as to eliminate costly trial-and-error engineering revisions, and to yield a more reliable product.

2.2.2.2 Fault Simulators - Operating Characteristics

Fault simulators can accomplish the same basic tasks for SCOMP that are obtainable by manual analysis means. Algorithmic simulations by computer do not eliminate all manual effort, however. Manual coding and manual interpretation of simulator outputs are still both necessary and significant cost items.

The LASAR (Logic Automatic Stimulus and Response) simulator is typical and perhaps the most highly developed fault simulator available. Originated by Digitest, this simulator has been upgraded both by University Computing Company and Honeywell. Basic circuit elements are modeled by LASAR as nand equivalents (most TTL small scale integrated circuit types have library models of their nand structures). These callable models greatly simplify the coding process. Unfortunately, the SCOMP minicomputer circuitry employs many MSI and LSI microcircuits of newer types for which library models must be developed. This situation results in a fairly high additional cost as the models are individually complex and approximately one-third of the 100 plus integrated circuit types used in SCOMP require modeling before system level simulation could begin.

LASAR type simulators are essentially data matrix manipulators. While this is both accurate and complete, it requires a substantial amount of CPU time to execute all possible combinations. Matrix manipulation by such computer program is a very limited technique due to the fact that run times are proportional to N**2.5, where N is the nand equivalents. The LASAR "fail-all" mode, for example, will drive all unique failure modes and simulate them one at a time, building a file which shows for each failure mode the output pattern (including patterns which represent compromise conditions) which it fails.

The fail-all approach, while simple and accurate, is costly since a 200 IC network has about 2000 nands, 6000 failures and 3000 stimulus patterns to simulate for each failure, or 18,000,000 simulations. Even at its fast 20 ms per pattern speed, 360,000 seconds, or 100 hours, would be required. For this reason, this most direct approach to fault isolation file generation

2.2.2.2 Fault Simulators - Operating Characteristics (Continued)

has been replaced by the DYSOGN or ISOGEN approach which is about 100 times as fast. DYSOGN's and ISOGEN's accuracy has been verified by comparison with the fail-all simulation output.

If the user desires to fail only IC interface pins in order to reduce the run time, a mode is available for this.

The Fast Sim mode performs the same functions as the Fail-All mode, but in about one-fourth the time. This speed-up is made possible by carrying 100 failures per pass, maintaining delta configuration states for each failures so that only the area in the vicinity of such deltas need be processed and only to the extent that such area interacts with an active region of the network for that stimulus.

Fail-All and Fast Sim run times are proportional to N**2.5, where N is the number of nands. Table V illustrates the relative fault simulator run costs for various SCOMP elements and approximate manual analysis support costs. If they take 100 hours and 25 hours, respectively, for a 2000 nand (200 IC) network, then they take roughly 1 hour and 1/4 hour, respectively, for a 30 IC network, which is thus about their applicable limit. While each may occasionally find special application, both have essentially given way to the ISOGEN system because of its greatly increased speed (about 50 minutes for 3000 patterns on a 200 IC network).

ISOGEN accomplishes a similar function to the techniques described above using a criticality trace to derive the fault dictionary. This results in a drastic reduction in run time of about 100 to 1 when compared to Fail-All. Run time proportionality is just N**1.5, a substantial improvement. Inherent problems plague criticality trace techniques, which affects their accuracy and ease of use. Multi-Zero and Zero-One effects (logic states and logic state transitions) create discontinuities in some logic conditions causing actually critical nand failure elements to be ignored. Networks involving memory elements or counters require elaborate history maintenance to determine true effects of a failure occurring at some arbitrary time.

TABLE V

RELATIVE FAULT SIMULATOR RUN COSTS

	Circuit to be Simulated	Gate Complexity	Relative Run Cost	Relative Eng. Analysis Cost	Normalized Total Cost
1.	Multiple device controller	2,500	1.0	2.0	1.0
2.	Direct interface unit - SCOMP side	2,800	1.3	2.0	1.1
3.	Multiple line communications controller	4,700	4,8	2.0	2.3
4.	Central processor unit	6,900	12.6	2.0	4.9
5.	Direct interface unit - multics side	8,300	20.0	2.0	7.3
6.	SPM	9,600	28.9	2.0	10.3
7.	Total of 1 to 6 above; taken individually	34,800	70.5	12.0	26.8
8.	Total of 1 to 6 above; simulated simultaneously	34,800	722.0	12.0	244.7
9.	SPM and CPU; simulated together	16,500	112.0	4.0	38.7
10.	SPM and CPU control interfaces only; simulated together	11,325	43.6	3.0	15.5

2.3 Recommended Probabilistic Measurement Methodology for SCOMP

The functional level FMEA, performed manually, is sufficient to achieve SCOMP probabilistic measurement objectives. Descriptions of security compromises, as in Table I, are sufficient to support a functional FMEA. The advantages of this selection are:

1. It's timely because it requires a minimum of prerequisite data which are expected to be available concurrent with detail design.
2. It's emphasis is on influencing circuit design architecture which should be the primary objective of the probabilistic measurement.
3. It's cost effective, yielding high confidence system level analysis at a fraction of the effort required by more detail evaluation.

Logical upgrading of the confidence in probabilistic measure data is achievable along several paths. While confidence determination is perhaps the most subjective element in the methodology selection process, the following order of upgrading appears reasonable should it be desired.

Methodology	Confidence
Manual FMEA, Functional	> 85%
FMEA, Part Level	> 90%
FMEA, Single Failure Analysis	> 95%
Fault Simulator; ISOGEN	> 97%
Fault Simulator; Fail-All	> 99%

Fault implantation and piece part mechanical FMEAs are not recommended for SCOMP.

SECTION III

SCOMP HARDWARE CERTIFICATION

3.1 Objectives and Criteria for Hardware Certification

Two major objectives must be addressed to achieve SCOMP hardware certification. These are design verification analyses and hardware verification tests. Each of these objectives has its own issues and criteria which establish boundaries on candidate methodologies that can be employed in satisfying the objective.

The hardware design verification objective is involved in the issue of desired confidence level. While design security is not directly at issue here, two related criteria require that relatively high confidence be established. These are:

1. Certification that the SCOMP design accomplishes the performance requirements of its design specifications; and
2. Verification that the hardware design is closed; that is, its mechanization does only that which it is specified by design to do.

Hardware verification testing must address the initial performance testing, as well as the controls upon which physical certification of production hardware are to be based.

3.2 Hardware Design Verification Analyses

The logic design verification techniques are primarily circuit design analyses to some level of detail which verify that the stated performance specifications are accomplished by the digital logic mechanization. If we assume that the SCOMP hardware functional design specifications for the SPM and Interface Unit (IU) correspond to the Secure Communications Processor architecture specification, we may proceed directly to analyze their circuit design mechanization in terms of design specification requirements (DS Part I).

In Section 3.2.1 which follows, the characteristics of available design analysis tools which accomplish design verification are described. Recommendations are contained in Sections 3.2.2 and 3.2.3.

3.2.1 Hardware Design Verification Descriptions

3.2.1.1 Manual Analysis

Circuit design analysis can be accomplished manually by the designer or an independent reviewer. There is a long list of design analysis types; each type of analysis addressing a specific design objective. The list includes logic correctness analysis, circuit timing analysis, worst case circuit loading (electrical stress) analysis, structural and thermal analysis.

Of greatest interest and necessity for SCOMP is a logic correctness analysis. The SPM and the 6000/60 IU are the only functional elements of the SCOMP minicomputer which do not yet have the benefit of sufficient correctness analysis. The functional complexity of the SPM (and perhaps also the 6000/60 IU) could require a very substantial manual effort to thoroughly explore the many intricate circuit interactions. The manual technique does not lend itself to effective documentation; and, by its very nature, is prone to human introduced analytic errors.

3.2.1.2 Instruction Simulator Description

The function of an Instruction Level Simulator would be to perform the same functionality as the minicomputer CPU and SPM hardware. This functionality would primarily be used to run and debug SCOMP test software. The intended life of the simulator is until the hardware is operational. As such, it will be used to give software design the opportunity to develop functioning software prior to the hardware availability. Hardware elements such as registers, memory, accumulator states, compare states, etc., as specified by the CPU and SPM specifications, would be simulated and available for interrogation and modification. The standard minicomputer order repertoire, including a limited I/O, and security unique instructions would be available.

The following definitions apply for the Instruction Simulator:

Order - The group of words required to define 1 computer function

	to be performed. The order comprises from 1 to n computer words.
Instruction -	The first word of the group of computer words that comprises the order.

The initial input/output for the simulator takes two forms. The first form is the actual minicomputer software program to be simulated along with the related supporting I/O. The second form is the data input/output processing the simulated program will use to manipulate the actual program data. The output is to be in the form of one MULTICS segment (file). The actual data file manipulation is initiated from the program by special simulator I/O orders.

The interface between the user and the simulator will be minimal. The interface consists of a numbered set of sub-commands with subsequent parameters as needed.

This interface supplies the following capabilities:

1. Execute 1 or more orders.
2. Dump memory in decimal or hexadecimal.
3. Print values of program counter, accumulator, base register or current memory location.
4. Print machine status; registers and last instruction.
5. Initialize, terminate, restart and continuous execution.
6. Load registers or memory.

These capabilities can provide sufficient visibility of hardware functions to effectively evaluate their performance.

3.2.1.2.1 Adaptation of the CPU-SPM Instruction Simulator for Hardware Verification

The existing CPU-SPM Instruction Simulator is basically a software development tool for SCOMP Kernel software and new security instructions added to the existing mini-computer CPU instruction set. In the form necessary for these tasks alone, this simulation is not sufficiently detailed in its view of the SPM hardware. To use this approach, the CPU-SPM Instruction Simulator would have to be modified to provide a detail view of SPM hardware functionality. One method of achieving this is shown in Figure 3.2.1.2.1-1. This structure provides for both detail (complex) and simple views of SPM functionality in one simulator. The required CPU-SPM simulator modifications are a straightforward process involving the following four task elements:

- Modify 12 of the existing CPU subroutines to accommodate SPM functionality.
- Create four administrative subroutines to provide both simple and complex SPM algorithms and input/output routines.
- Create nine SPM service routines based upon DS Part I descriptions of SPM functionality. These service routines would describe the following SPM functions and call to lower level register control routines.
  - Address Translation
  - Access (Cross Ring Validation)
  - Effective Ring Calculation
  - Argument Validation
  - Memory Descriptor Handler/Interpreter
  - Device-to-Memory Interface
  - Device-to-Processor Interface
  - Processor-to-Processor Interface
  - Operator-to-Processor Interface

3.2.1.2.1 Adaptation of the CPU-SPM Instruction Simulator for Hardware Verification (Continued)

- Create an SPM register control routine based on SPM circuit interconnections and register functional links. This routine would contain entries to the above described service routines and would simulate individual SPM register actions.

These modifications effectively overlay ongoing effort to develop Kernel software. The interaction of these task activities is shown for the Instruction Simulator approach in Figure 3.2.1.2.1-2.

This has definite advantages in that the software analyses are always in step with the hardware verification analyses.

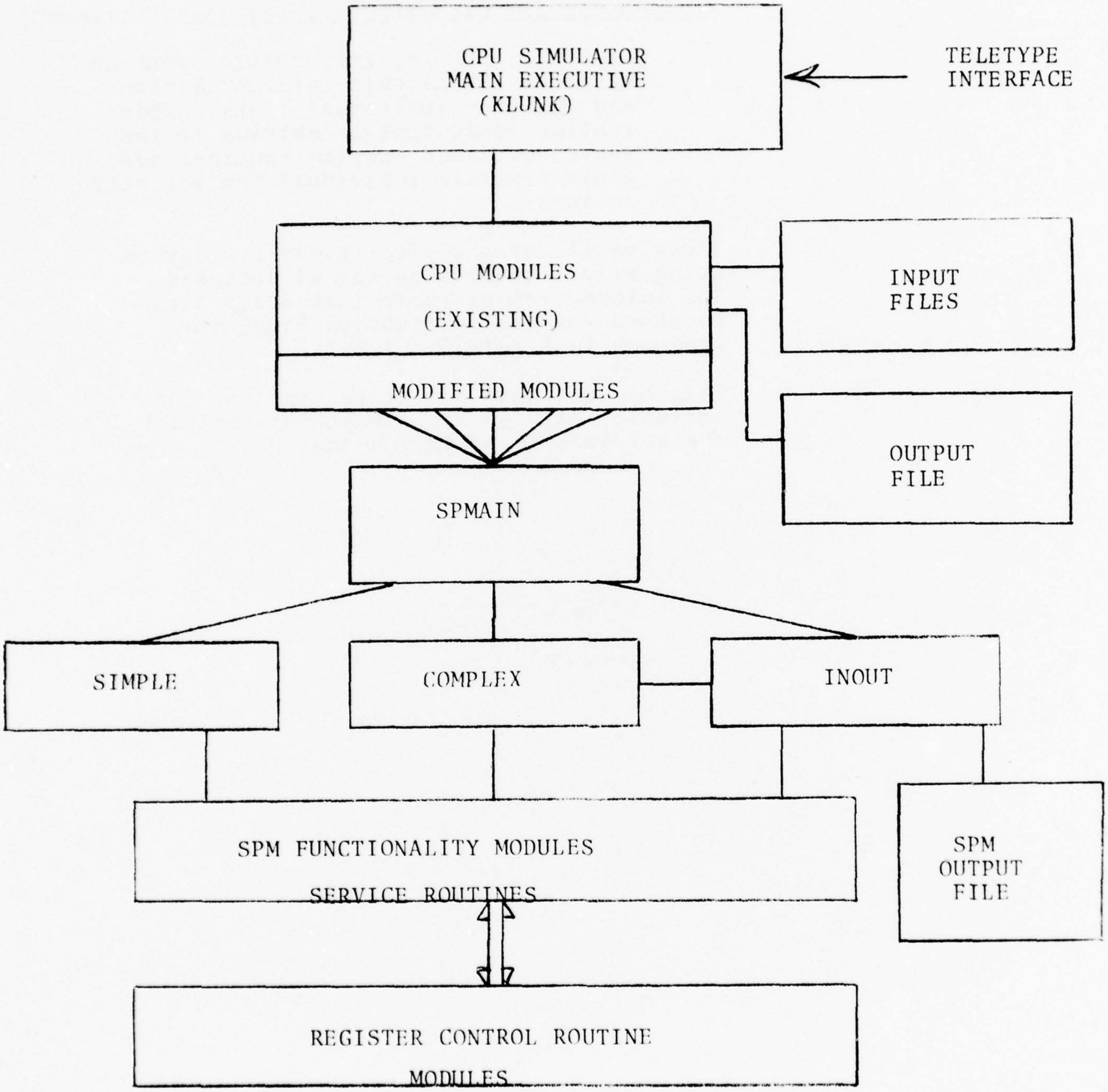
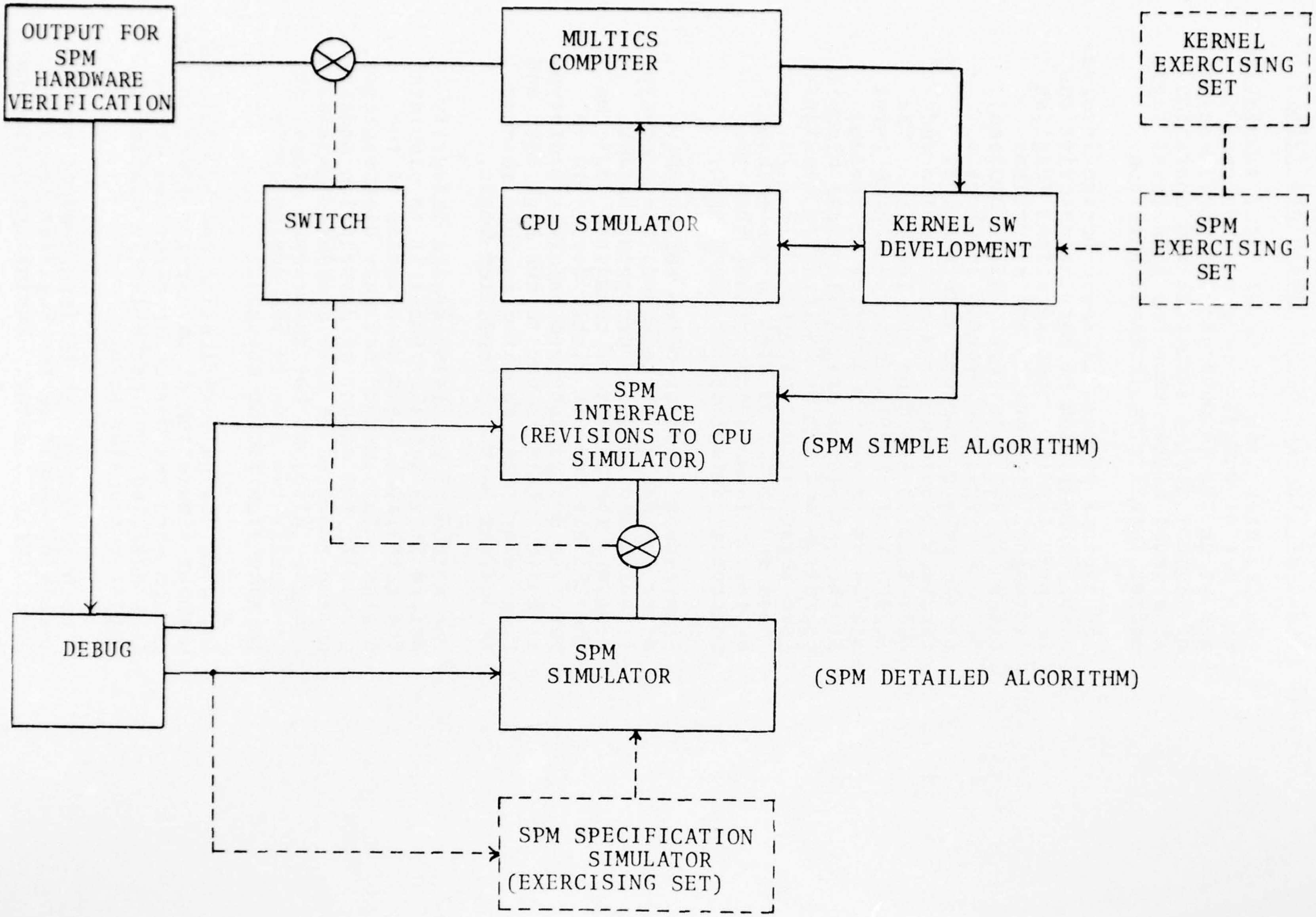


FIGURE 3.2.1.2.1-1

STRUCTURE OF CPU-SPM SIMULATOR

FIGURE 3.2.1.2.1-2

PROCESSOR INSTRUCTION SIMULATOR APPROACH/TASKS



SCOMP KERNAL SOFTWARE CAN BE DEVELOPED USING THE SAME SIMULATOR WHICH IS USED FOR HARDWARE DESIGN VERIFICATION

3.2.1.3 Register Transfer Logic Simulators (RTL)

The Register Transfer Level (RTL) simulation is a set of computer programs designed to assist in the formulation and verification of digital device structure and operation at a level higher than the gate-level logic and/or detail circuit implementation.

The typical process of Logic Design involves manual preparation of machine descriptions at three levels: specifications (English language), flow charts and algorithms (graphic) and gate-level logic (Boolean). At each level, the process involves a choice between various alternatives, feedback resulting from that choice and modification based on the feedback. The feedback at the flow-chart and gate-level stages is primarily a matter of review. At the gate-level stage, illogical circuit operation and timing factors are considered (see Logic Circuit Simulators, 3.2.1). These are mostly related to the mechanics of the implementation, rather than the conceptual integrity, of the design.

Significant feedback often begins only after the design is released, the prototype is built and hardware debugging begins. However, the advent of committed logic MSI and LSI has made it highly desirable to have the design debugged before a prototype is built. This is due to the high cost and long delay times required to make changes to designs based on committed logic.

One method of verifying design integrity before it is actually built is to simulate the gate-level logic description of the design. This method has many disadvantages, primarily the amount of description modification required to make a major significant change. Also, at the gate-level stage, it is usually too late in the design cycle to make significant changes.

Improved design verification can be achieved without simulation at an earlier stage in the design when the device is described at the less detailed, but conceptually complete, register transfer level.

A machine design can be described at various levels of detail in the Register Transfer Level (RTL) language. During the preliminary

3.2.1.3 Register Transfer Logic Simulators (RTL) (Continued)

stages of machine design, the logician may be interested only in outlining the data flow, whereas at later stages, he will want to include more detail by specifying intermediate registers and portions of the control logic. Simulation of an RTL logic description provides the logician with an opportunity to evaluate machine algorithms with a minimum of design data. Thus, a number of alternate approaches may be explored and compared early in the design cycle when conceptual changes are far less expensive to implement.

RTL simulation of a design is accomplished by programs which create a simulation model from the register level description and then run designer specified test programs through the model. The various control states through which the model cycles and the contents of the simulated registers and memories may be checked against precalculated results. The simulator may be instructed to report the state of various model elements under a variety of conditions. Should the model not produce the correct results, these reports may be used to locate the design errors.

There are several important advantages to this type of simulation. The final design is conceptually debugged before build documentation is generated. For example, Read Only Memory (ROM) algorithms can be verified prior to detailed design of the ROM word layout. Also, very early in the design cycle, the logician can vary parameters, change algorithms and receive results for evaluation of speed, efficiency, etc.

The same preciseness and lack of ambiguity that are required for simulation conceptually allow gate-level synthesis from the RTL machine design. The initial gate-level description may be written directly from RTL reports, with the advantage that gate-level design is performed with verified conceptual integrity.

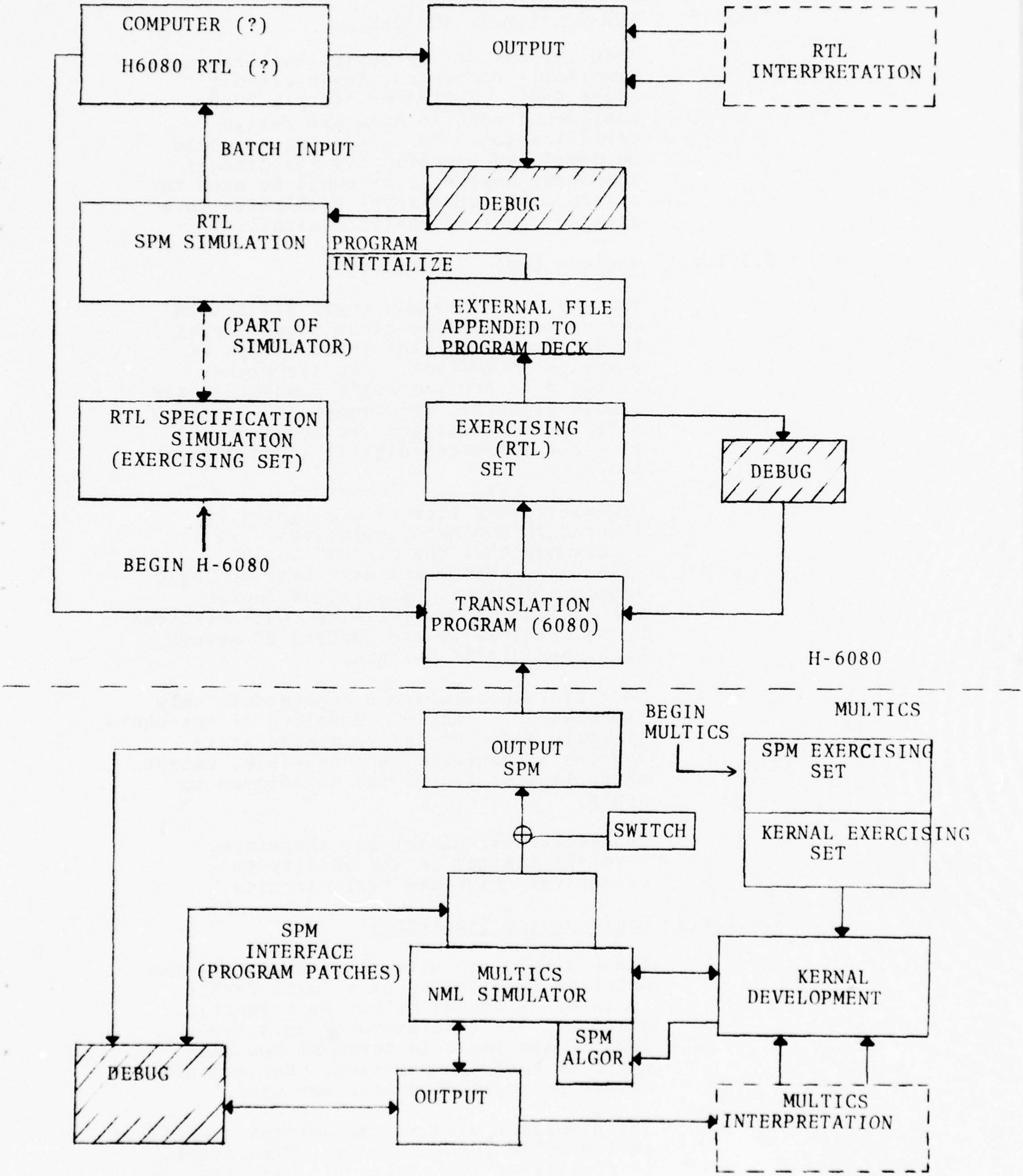
3.2.1.3.1 Use of RTL Simulators for SCOMP

An RTL simulation, if selected for SCOMP, would have to be written from the beginning, RTL providing only the framework and conventions for circuit definition. To do this would require two major tasks:

- Create the simulations of the CPU and the SPM in RTL.
- Develop an input problem to run on the new simulator.

While both tasks are substantial, the creation of the input problem is the larger technical challenge. It is believed that the only effective SPM exercising problem will come from the CPU-SPM simulator used for Kernel development. Unfortunately, RTL provides only batch operation on the H6080 computer and the Instruction Simulator is interactive on the MULTICS system. The creation of an RTL simulation therefore requires a complex translation from MULTICS to get its input problem. Figure 3.2.1.3.1-1 illustrates the approach.

FIGURE 3.2.1.3.1-1
RTL SIMULATION APPROACH



3.2.1.4 Logic Circuit Simulators

When circuit detail design has been substantially completed, logic circuit algorithmic simulators offer a high confidence path to hardware design certification. These simulators could be developed specifically for circuit component analysis, or could be used to extend a register level simulation to a greater level of analytic detail.

3.2.1.4.1 Boolean Logic Simulators

The family of Boolean Logic Simulators are employed at the circuit gate level to determine that the logic design of a device is correct. The algorithms utilized in Boolean Logic Simulators are simple algebraic relationships. Boolean gate level simulators are among the earliest automated digital logic design tools.

Circuit timing factors are typically ignored in Boolean simulations. It is assumed that the circuit logic stabilizes before the next test or logic sequence occurs. This is, of course, not always the case; timing considerations being critical to the success of modern high speed logic designs.

Most Boolean simulators accommodate only two states: 1 and 0. Modeling of one-shots, tristate logic or indeterminate state devices is generally not possible, except where logical 1 or 0 may be assumed to apply.

The Boolean simulator is, therefore, severely limited in its ability to effectively simulate real circuits.

3.2.1.4.2 Logic Device Simulators

These simulators attempt to duplicate the detailed functioning of a logic device in terms of signal values as a function of time. All simulators of this type look at the logic in terms of how each piece of hardware performs. For example, actual propogation delays are used.

The simulator used at the Aerospace Division, called HISIM (Honeywell Inc. Simulator), is typical of the class. It has five states: 0, 1, X (static unknown), Z (tri-

3.2.1.4.2 Logic Device Simulators (Continued)

state high impedance) and I (initial undefined). It uses real time circuit delays for 0→1, 1→0, 0→Z, 1→Z, Z→0 and Z→1. It simulates synchronous and asynchronous logic for gates, flip-flops, one-shots and MSIs. The MSIs may contain RAMs, ROMs and/or truth tables, as well as gates.

It optionally detects failure to stabilize time(s) and stops or continues at user discretion. It detects inputs less than gate delays and ignores inputs less than a user specified minimum pulse width. It has several diagnostics for RAMs and ROMs, such as: detecting an undefined RAM address while write enabled, undefined RAM/ROM content, and reading and writing the same RAM address if it's illegal.

HISIM uses two libraries: a logic library that contains gate and MSI logic; and, a RAM/ROM Truth Table library which contains the data for each device. HISIM has a formattable output that can be in any arrangement desired, with or without labels and a line printer plot as a function of time.

3.2.1.5 Security Proofs for Operating System Software

Correlation between the software certification methodology and the probabilistic measures and hardware certification methodologies is not immediately obvious. The approaches to the problems are, however, similar in several respects. The need to employ a modular view of the system is evident in both hardware and software certification tasks. In the hardware case, partitioning of the SCOMP into functional modules is particularly useful to the study of failure induced system effects.

In the course of our trade-offs, an extensive review of published articles pertaining to software systems was conducted. Unfortunately, there is little evidence that the theoretical development work now in progress could be applied to the SCOMP hardware certification in any meaningful way at this time. An approach similar to the software mathematical model and specification language proofs (such as those advanced by Bell and Lapadula and Neumann, Levitt, et al) applicable to

3.2.1.5 Security Proofs for Operating System Software (Continued)

software certification may eventually become a viable alternative. In this regard, Neumann and Levitt, et al (Reference 2) offer a five-stage decomposition of proof in which the fifth, and last, stage is "the actual implementation in terms of hardware or a programming language."

We determined that the current state of development of these software techniques does not provide a clear and bounded methodology which could be effectively evaluated in our hardware verification trade-off studies.

3.2.2 Recommended Hardware Design Analysis for SCOMP

A CPU-SPM Instruction Level simulation is recommended for the SPM and the minicomputer CPU logic dedicated to the SPM interface. The need for additional simulation of the 6000/60 Interface Unit is believed to be of lesser priority. In making this selection, both manual circuit analysis and software proof type approaches were determined to be inappropriate to the problem.

While RTL simulation is a powerful tool that can be very effectively applied to logic as complex as that of the SPM and the minicomputer Central Processor, the unique character of the SCOMP problem (security Kernel software) causes RTL to be a second choice. By using a complex, as well as a simple, view of the SPM in the SPM-CPU Instruction Simulator, the rigor of analysis, which is RTL's strongest recommendation, is equalled. RTL would have provided simpler upgrading paths if additional detail analyses were desired at a later date.

Logic circuit level simulations are always viable contenders as a design analysis tool. Of the two circuit level simulator types, the HISIM type would have been preferred; the Boolean approach being technically obsolete. The cost of circuit level simulations is, however, relatively high, although its algorithms yield the most precise simulation available. The design verification testing (see paragraph 3.3.2) can accomplish much the same confidence in the hardware. As some design verification testing is considered essential in any event, detail circuit level simulation must be viewed as less cost effective.

3.3 Hardware Verification Tests

Hardware verification testing is necessary in both prototype and production environments. The elements of which hardware verification is comprised are described below.

3.3.1 Prototype and Production Logic Test Criteria

Functionality of the hardware security functions of the physical hardware is verified in the course of prototype and production product performance tests. These tests need to be structured to ensure that the functionality of each hardware element which has a reference monitor function is correct. While it is probably impractical to exhaustively exercise every element of the SPM, it is practical to expand performance testing to include typical routines which exercise the operational characteristics of each SCOMP performance specification. The most efficient method of achieving this is to develop test software in a systematic way structured toward this objective.

It is recommended that evaluation software be developed with the aid of a CPU-SPM instruction simulator having the characteristics described in paragraph 3.2.1.2. This approach insures that functional acceptance tests will have desired and predictable characteristics.

3.3.2 Design Verification Testing

Design Performance Verification Test techniques are intended primarily to ensure that the logic design (and analog support circuit designs) maintain the specified performance characteristics over the environments of the application. These elements of the hardware verification include voltage and timing margin tests and environmental performance tests, such as temperature extremes, vibration, etc. Trade-offs are not appropriate for the design verification test element of the hardware certification task. The hardware design tests that are appropriate to the SCOMP include the following:

- Temperature Altitude Testing
- Humidity Exposure - Endurance Testing
- Physical Shock Testing
- Sine Vibration Testing
- Electromagnetic Compatability (EMC) Testing

These tests are the qualification tests established for the ruggedized minicomputer, which has been selected for application in the SCOMP. It is not necessary to repeat these tests for the SPM or the 6000/60 IU. The SPM and IU should be considered qualified by structural similarity to the minicomputer due to the similarity of interfaces and form factors. Qualification tests are planned for the minicomputer as part of a separate project.

3.3.2 Design Verification Testing (Continued)

It is necessary to augment these qualification tests for both the SPM and the IU with selected circuit performance tests. These additional tests should be structured to exercise circuit performance operating margins in at least the following areas:

1. Worst case voltage extremes of input power and of internally generated logic operating voltages.
2. Worst case clock frequency variation to isolate critical timing chains (if any) and establish operating margins.
3. High and low temperature operating tests, including monitoring of critical performance parameters.

3.3.3 Acceptance Criteria for Production Hardware

It is not practical to attempt a complete design certification of production computer hardware in the absence of a controlled build environment. There is a wide range of techniques available to establish a controlled build environment for digital computers such as the SCOMP. These range from simple configuration inspection of the finished product which accomplishes a verification that the product is like its design drawings to elaborate access controlled build areas where access to the hardware is limited to cleared personnel who are trusted not to maliciously modify the hardware. Regardless of the extent of manufacturing line controls, which though not trivial can be left to the Quality Control discipline, it is necessary to perform product acceptance tests which verify that security related hardware functions of the SCOMP are operational.

The following SCOMP production item control elements have been specified in the DS Part I specifications for both the SPM and the 6000/60 IU.

Configuration

Each production SPM shall be visually examined in individual parts kit form prior to issuance to assembly and, again, upon completion prior to acceptance testing. Configuration examination shall include:

- Verification that correct part types have been issued for manufacture.

3.3.3 Acceptance Criteria for Production Hardware (Continued)

- Completed assemblies are complete and visually identical to a standard reference SPM or photograph thereof.

Electronic Parts Inspection

The logic functionality, damage and marking of integrated circuits to be assembled into production SPMs shall be verified by inspection and test prior to assembly. Appropriate quality control sampling plans based on lot total percent defective (LTPD) acceptance criteria shall be employed for marking and damage.

Production Acceptance Testing

Acceptance Tests

Production acceptance tests shall be conducted under the supervision of quality control using approved test procedures, equipment and software. Each SPM shall be accepted with the SCOMP unit for which it is intended. Spare SPMs may be acceptance tested in any SCOMP of compatible configuration provided that all functional elements used in the test have been inspected for assembly workmanship.

Production Test Software

Software used for acceptance testing of production SPMs shall be derived from the prototype software (see paragraph 3.3.1) or other suitable source which insures that each SPM mediation function is exercised.

Production test software shall be formally issued and controlled.

SECTION IV

CONCLUSIONS AND RECOMMENDATIONS

The hardware verification methodologies investigation has resulted in recommendations in three areas:

1. Probabilistic measures analysis techniques
2. Hardware design certification technique
3. Physical product test and certification criteria

A manual analysis probabilistic measures analysis technique was selected. A SCOMP functional level of analysis was determined to be more suitable than a detail electronic circuit analysis of every component.

A CPU-SPM Instruction simulation is recommended to accomplish the SCOMP hardware design certification. The simulation would encompass the SPM and the portions of the CPU dedicated to support the SPM interface. The technique may be extended for the Series 6000/60 Interface Unit.

Test and inspection criteria were developed and specified for SCOMP hardware new design elements. These criteria include reference monitor functional exercising to be developed on an instruction simulator, electronic parts logical tests for production units and configuration inspections to insure integrity of production product.

BIBLIOGRAPHY OF REFERENCES

1. SCOMP Architecture Specification for Secure Multics, prepared under Contract F19628-74-C-0205, draft August 1975.

2. A Formal Methodology for the Design of Operating System Software; Robinson, Levitt, Neumann, Saxena, September 1975.

APPENDIX A

A FORMALISM FOR DESCRIPTION OF SCOMP SECURITY COMPROMISE

SEPTEMBER, 1975

1.0 INTRODUCTION TO APPENDIX A

This Appendix contains an example of a rigorous logical notation which could be used to establish functional failure categories in a minicomputer system. These failure categories can be evaluated by inspection for security breach characteristics and be directly translated to an English equivalent table of possible security compromises which could be induced by hardware failure. This formal notation was developed in an effort to improve upon earlier attempts to create tables of possible security compromises in a minicomputer using only computer system architecture data and functional diagrams.

2.0 DESCRIPTION OF THE FORMALISM

This formalism has four major elements which should be reviewed:

A. Each secure minicomputer operating function involving the CPU or the SPM must be identified. A few are listed in Table I. These functions involving the system of both hardware and software are analyzed as defined in the three following elements of the formalism.

B. The system result of a hardware failure defined in terms of a change in value of some system parameter. These must be accounted for all system parameters pertinent to the operating functions defined above.

C. The sets of consequences for each possible change in value of each system parameter. The sets must be iterated to successively smaller subsets such that all possibilities of interest are described in detail.

D. The system view of the consequences defined above at the lowest level of consequence subset. These system views of failure consequences are stated in terms which have meaning to the user, user files, the normal hardware fault circuits and the computer control panel. The major system views of failure consequences are listed in Table II.

TABLE I

SCOMP FUNCTIONAL BREAKDOWN INVOLVING CPU & SPM

- Request for Level 6 CPU Bus action
- Request for Internal Bus action
  (several - within specific hardware module as in Figure 2.1.2-1)
- Request for firmware action
  (Control Processor, Security Protection Module, Multi-time Communications Processor)
- SPM, fast access store action
- SPM descriptor cache action
- Standard Bus I/O interface

TABLE II

SYSTEM VIEWS OF FAILURE

PERR	Program error, incorrect execution, etc., but not security related
SERR	Security error, fault mask, etc. . . .
FAULT	Normal fault
Normal INTT	Normal interrupt to processor
NO ACTION	Halt
NORMAL ACTION	Normal control action on system bus
ABNORMAL ACTION	induced by failure: unexpected illegitimate control action.

TERMINOLOGY

$=:$	defined equality
$< \quad >$	defined variable
K	Kernel
S	System (operating supervisor)
A	Application (user program)
|	logical or
&	and
Δ	change due to hardware fault
INTT	normal interrupt
PERR	program error (incorrect execution)
FAULT	hardware trap (aborts at memory cycle) - calls a routine
SERR	serious (security?) error (incorrect mediation)
(V)	virtual (subscript for PERR, SERR, FAULT)
(A)	absolute (subscripts for PERR, SERR, FAULT)
Module	any functional unit interfacing the SCOMP bus
Support	timing and/or power inputs necessary to facilitate bus cycle

Examples of Formalization of Hardware Fault Definitions Which Result in Security Compromise

```
REQUEST FOR BUS ACTION - - - - - - - - - - - - - "bus" = any data or control array of bits
                                                         internal to a module, or on system
CONTEXT OF REQUEST = :: |<K code>                        bus.

   |<S code>|<A code>|<K data>                   "bus" = :: <(V) address>|<(A) address>

   |<S data>|<A data>|<K I/O >                      |<(V) data>|<(A) data>

   |<S I/O >|<A I/O >|<Input >                      |<(V) address master control>

   control><|<output control>|                      |<(A) address master control>

   <input data>|<output data>                       |<(V) address slave control>

   <fault response>|<interrupt response>|           |<(A) address slave control>

   <K action>                                       |<(V) data master control>

"code" = :: procedure = :: <op code>|               |<(A) data master control>

   <memory address> |<register                      |<(A) data slave control>

    address>|< index register                       |<(V) data slave control>

    address>|< I/O address>                         |<(V) bus support>

                                                    |<(A) bus support>
```

PERFORMANCE RELATIONSHIPS IN THE PRESENCE OF HARDWARE FAILURE

```
GENERAL FORM:  Change in value due to }  = :: <Consequence of value change>
               Hardware Failure       }                └→ SYSTEM VIEW OF FAILURE

   I.   Δ(OP code) = :: <Legal Operation>|<detected illegal operation>|
                              └→ PERR                 └→ FAULT

                        <undetected illegal operation>
                              └→ PERR (?)

  II.   Δ(Memory Address) = :: <address too high>|<address too low>

          <address too high> = :: <next higher increment>|
                                        └→FAULT (V)
                                          SERR  (A)

                         <same segment, higher address>|
                               └→PERR

                         <non-existant segment>
                               └→FAULT (V)
                                 SERR  (A)
```

```
        <address too low> = :: <next lower segment>|
                                  └──►FAULT (V)
                                      SERR  (A)
                     <same segment, lower address>|
                              └──►PERR
                     <non-existant segment>
                              └──►FAULT (V)
                                  SERR  (A)

III.   Δ(Register address) = :: wrong operand selection
        = :: <read & wrong operand selected>|<write & wrong operand
                       └──►PERR
           altered & proper operand unchanged>
                       └──►PERR

IV.    Δ(Index Register address) = ::  wrong index selected = ::
           improper address generated = :: Δ(Memory address)
           = ::  II above
```

<I/O Address> = :: <controller add>|<device add>|<channel add>|<I/O function>

V. Δ(I/O Address) = :: <controller too high>|<controller unchanged>|<controller too low> & [<channel too high>|<channel unchanged>|<channel too low>] & [<device too high>|<device too low>|<device unchanged>]

<I/O Function> = :: <read data>|<write data>|<read control>|<write control>

VI. Δ(I/O function> = :: <improper controller selected>|<improper controller address>|<improper device address>|<proper device address>|<improper channel>|<proper channel>| & [(<read> & <no action>|<write>) & <read request>)|(<read> & <no action>|<write>) & <write request>)] & [(<control request> & <control>|<data>)[<control request> & <control>|<data>)]