A Division of
Massachusetts
Institute
of Technology

The Charles Stark Draper Laboratory

68 Albany Street, Cambridge, Massachusetts 02139 Telephone (617) 258-1475
Mail Station #35

SHUTTLE AVIONICS
COMPUTER SYSTEM STUDIES

TASK 28-S, VOLUME IV

DECEMBER 1972

PART I CONFIGURATION AND REDUNDANCY CONCEPTS

H. Blair-Smith
F. Gauntt
S. Rosenberg
G. Schwartz
W. Weinstein

PART II LIGHTNING PROTECTION STUDY

E. Hall
J. Allen
W. Weinstein

PART III COMPUTER SYSTEM RELIABILITY

R. Filene

Some material in this report is proprietary to IBM Corp. and Singer-General Precision, Inc. and is subject to the restriction stated on the following page.

RESTRICTIONS

Material on pages 44-46 is proprietary to IBM Corp. and is subject to the following restriction:

"These data shall not be disclosed outside MIT or the Government, or be duplicated, used, or disclosed in whole or in part for any purpose other than for the use in Space Shuttle Computer evaluation in "fly-by-wire" applications as specifically authorized by IBM. This restriction does not limit MIT's or the Government's right to use information contained in such data if it is obtained from another source without restriction."

Material on page 45 is proprietary to Singer-General Precision, Inc. and is subject to the following restriction:

"Information disclosed herein is the property of Singer-General Precision, Inc. It is furnished for evaluation purposes only and shall not be disclosed or used for any purposes except as specified by contract between the recipient and Singer-General Precision, Inc. Duplication of any portion of this data shall include this legend."

The data subject to these restrictions are contained in sheets referring to these restrictive clauses by means of footnote.

ACKNOWLEDGEMENTS

The assistance of Eldon C. Hall, Alan I. Green, and Malcolm W. Johnston in the preparation of this volume is gratefully acknowledged.

INTRODUCTION

This fourth volume of reports submitted under Task 28-S consists of three major parts. Part I, Configuration and Redundancy Concepts, addresses questions arising in these areas in the latter half of 1972; Part II, Lightning Protection Study, was released to limited distribution under separate cover on November 24; and Part III, Computer System Reliability, supports a larger avionics reliability modeling study, done for NR, and is also available as Digital Development Memo #709.

We have been guided in a general way by the Knox Committee Report[6], but without feeling strictly bound by it. Specifically, we have written as if the controversy over whether the third GNCS computer should be similar or dissimilar to the others had been settled in favor of similarity (this issue is addressed separately in reference 12).

It may be helpful here to comment on some of the terminology employed. "Digital Computation and Distribution System (DCDS)" was coined to denote all computers, data buses, mass memory units, and other digital logic supporting all the avionics in the Orbiter vehicle. It replaces "DMS" (Data Management System), used in and prior to Volumes I and II of this task, because many people now use "DMS" as a synonym for "PMS" (Performance Monitor System). The terms "rail" and "string" will cause no confusion if regarded as strictly synonymous and meaning a non-redundant set of equipment, especially a computer, a Bus Control Unit, and a data bus. A useful distinction between these terms was made in reference 2, but it does not apply here. Finally, the term "bus", which we continue to use for "time dimension multiplexed data bus", is being supplanted among the contractor team personnel by "multiplex".

The possibility for confusion arises from the fact that, in several of the referenced documents, we have used "multiplexer" or "MUX", not for a bus, but for the interface between a bus and a bus control unit or a subsystem interface unit.

A summary of this volume appears on the following pages, in the form of an annotated table of contents.

SUMMARY: AN ANNOTATED TABLE OF CONTENTS

A Division of
Massachusetts
Institute
of Technology

The Charles Stark Draper Laboratory

68 Albany Street, Cambridge, Massachusetts 02139 Telephone (617) 258-

PART I

CONFIGURATION AND REDUNDANCY CONCEPTS

Hugh Blair-Smith
Frank E. Gauntt
Sumner C. Rosenberg
Gary Schwartz
William W. Weinstein

1.0 SYSTEM CONFIGURATION

The DCDS configuration assumed in these reports and presented in the following section is intended as an amalgam of current ideas about Shuttle avionics from a number of sources, rather than an MIT invention. Early in the reporting period, these ideas came from MSC personnel; then increasingly from NR and contracting team people. Later, some results from the November meetings at NR influenced the work. The sources generally agreed on such points as:

(a) Avionics must satisfy a Fail-Operation, Fail-Safe (FO-FS) criterion from life-critical functions.

(b) Guidance, Navigation and Control (GN&C) is life-critical and should be performed by a triply redundant computer.

(c) The Performance Monitor System (PMS) is mission-critical but not life-critical, and should occupy a dual-redundant computer.

(d) The probability of successful mission completion must be at least .9; probability of crew survival at least .999.

One point on which little agreement has been reached is the number and kind of data buses connecting computers to subsystems. Accordingly, we have taken the simplest possible view, and de-emphasized them in this volume.

1.1. ASSUMED DCDS CONFIGURATION AND INTERCOMMUNICATION CONCEPTS

1.1.1 Configuration

As a framework in which to place the issues discussed in this volume, a DCDS configuration has been assumed. The system is composed of three regions: Guidance, Navigation and Control System (GNCS), Performance Monitor System (PMS), and Power Distribution and Control System (PDCS). Flight control is performed by GNCS. The chief function of the PMS will be the status and health monitoring of the various subsystems of the vehicle, and it is possible that payload management, payload manipulation, and CRT display management will be performed by PMS. The Main Engine System is considered a subsystem of GNCS.

The focus of this volume is on GNCS. Although little will be said about the other regions, the concepts developed for GNCS can generally be applied to PMS and PDCS. In this study we wish to emphasize the contractual advantages attributed to multi-region systems,[8] and with this in mind GNCS is assumed to be, as much as possible, self-sufficient. It is our intention that GNCS need not depend on PMS or PDCS operations to perform its duties. At the same time, there has been a suggestion that GNCS computers use fixed programs, possibly in a read-only memory, and that primary preburn targeting routines, which can be expected to be changed for different types of mission, would be located in the PMS computers, which have read-write program memories. (GNCS would contain backup targeting routines.) The DCDS presented here allows for this possibility.

The fault tolerance requirement on this system is fail-operational, fail-safe (FO-FS). The avionics must be fully capable after a single failure, and after a second failure enough of the system must still work to get the crew home safely. The goal is that a mission can be completed despite a failure, and two failures do not lead to loss of the crew or vehicle. Avionics

elements which are mission-critical must be duplicated, while those which are life-critical are triplicated. Since GNCS is life-critical, it must be triplex, and the same seems to be true for PDCS. PMS, which is mission-critical but not life-critical, is assumed to be duplex.

Triplication of GNCS does not imply the three copies are the same, and in fact the opposite may be the case. There is a concern that with identical copies a software or other systematic flaw could defeat the entire GNCS, replications and all; to avoid this hazard, it has been suggested that one of the copies be dissimilar to the other two. We have not included the element of dissimilarity in this system, but the question of a dissimilar backup has been discussed,[12] and there is nothing so rigid here as to prevent the eventual use of dissimilarity. There is a wide variety in the way dissimilarity could be implemented in a triplex GNCS. Some examples:

1. There could be two GNCS rails (as opposed to the three rails shown in Figure 1.1.1), with backup navigation routines in another region, namely PMS.[10]
2. There could be three GNCS rails, the third of which is programmed differently, and could conceivably even use a different computer type. Similarly, a system composed of three identical GNCS rails plus a dissimilar one is conceivable; such a system wouldn't necessarily break the FO-FS groundrule, since the third of the identical units might just be used for error detection. For a discussion of the use of multiple computer types, which is implied if a dissimilar backup is to be employed, see Section 1.2.
3. There could be three identical GNCS rails, each containing backup software.

A triplex GNCS is shown in Figure 1.1.1. Little detail is specified, because the object here is to present a flexible configuration as a background for the intercommunication, redundancy management and computer type discussions to follow. In general terms, the GNCS is composed of three

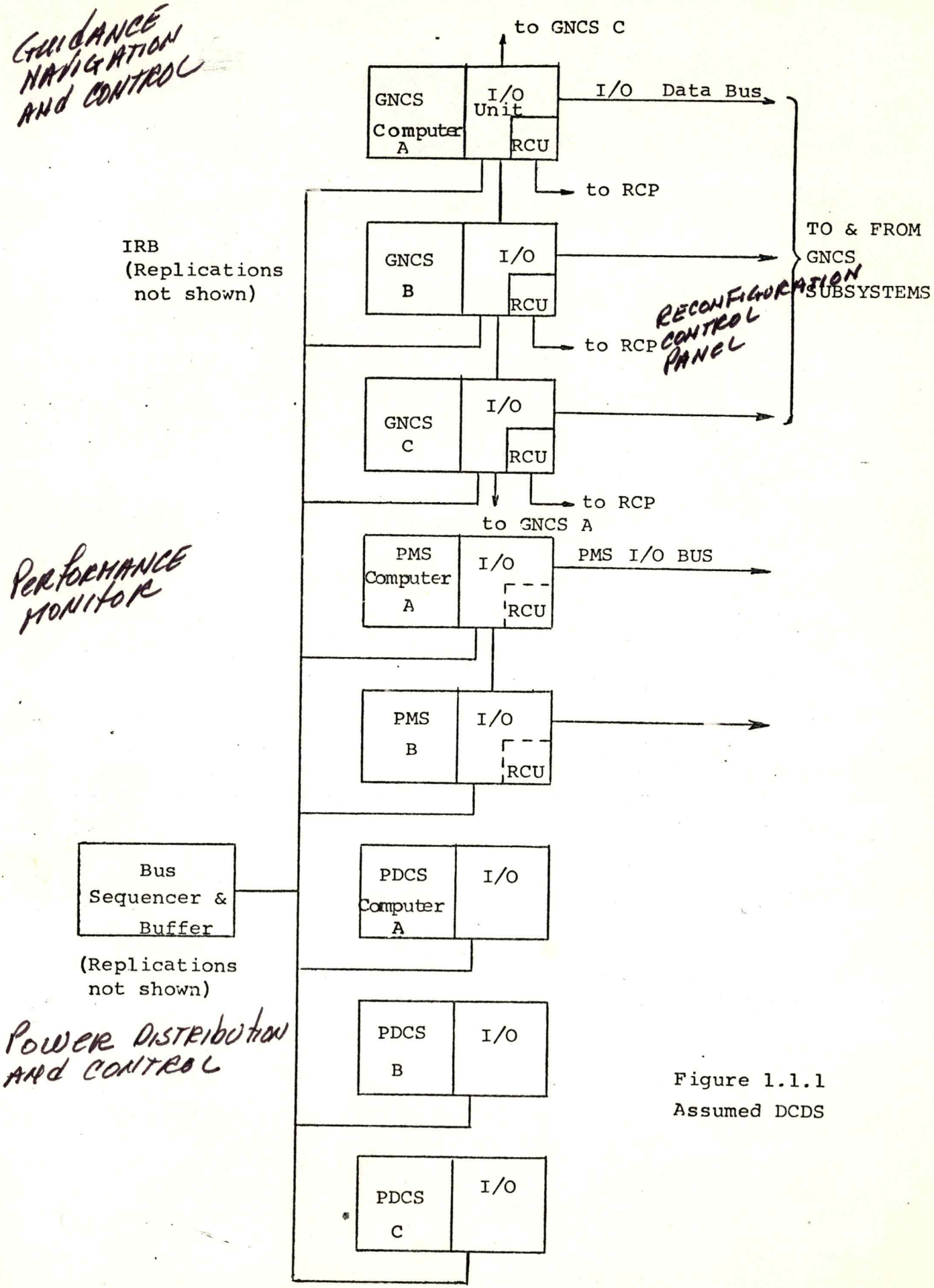


Figure 1.1.1
Assumed DCDS

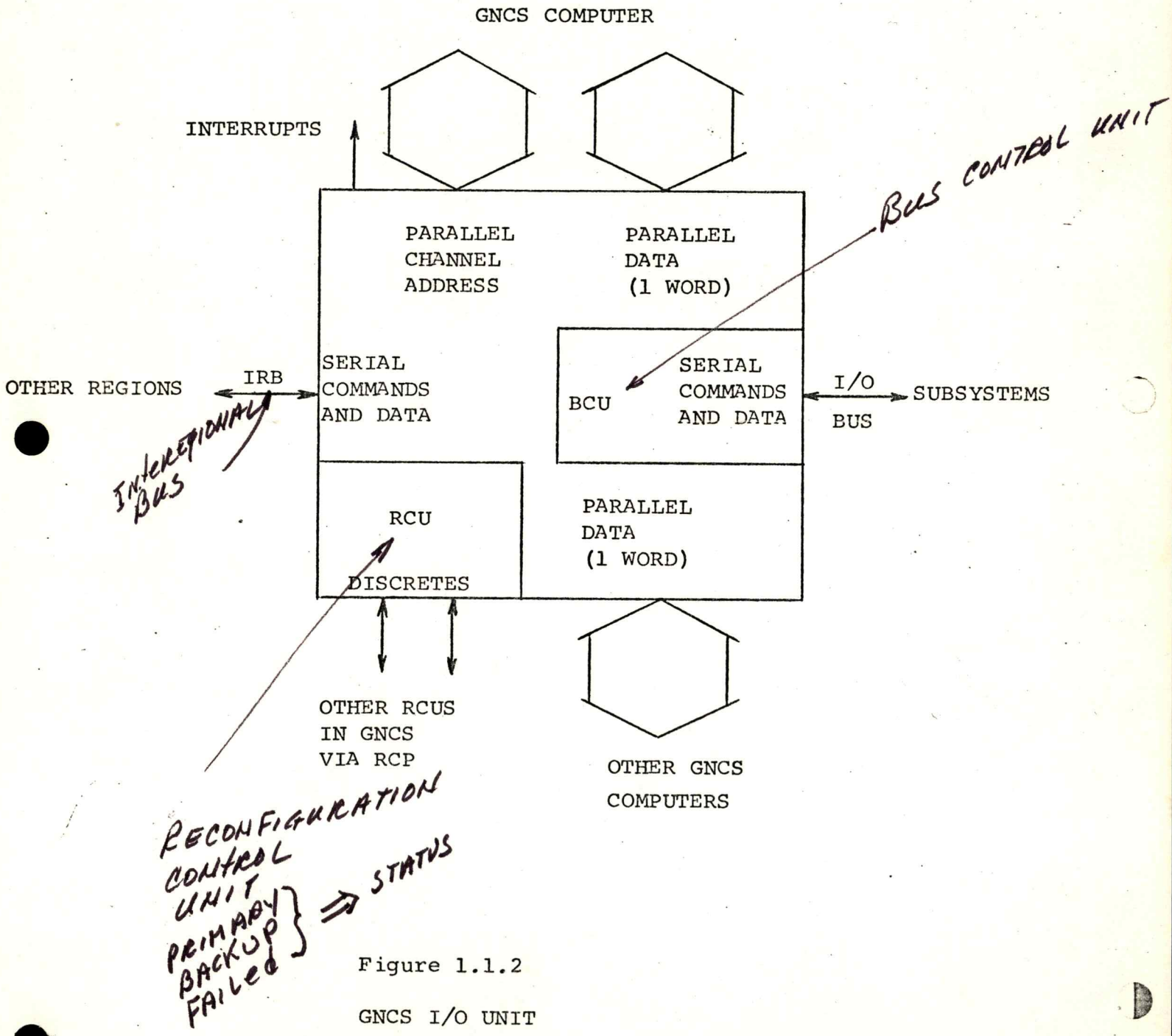


Figure 1.1.2

GNCS I/O UNIT

"loosely coupled" rails, each consisting of a GNCS computer, an I/O unit, and a multiplexed data bus. "Loose coupling" means that the GNCS replications are not hardware synchronized, although it is understood that the physics of the vehicle will cause the replications to be doing about the same thing at about the same time.[1,2,3,8] The I/O unit, some of which might be supplied by the computer vendor and packaged in the computer box, includes the logic for data bus control (BCU), reconfiguration control (RCU), and interfaces to the other GNCS rails and to an Inter-Regional Bus (IRB). An I/O unit is shown in Figure 1.1.2.

Reconfiguration control within a region may be conceived either as a unified "RCU complex" which is at least as fault tolerant as the rest of the system and controls the status (primary, backup, failed, etc.) of the individual computers and other units, or as the interaction of distributed RCUs, one per rail, each of which is sufficiently fault tolerant to assure the validity of status information. The latter conception will be employed in this volume, to achieve visibility into the ways that failures in the reconfiguration control area are associated with, or are indistinguishable from, failures of other units. It is expected that RCUs will be simple devices, requiring little hardware. The natural location for this hardware is in the I/O unit. An RCU issues a "please take over" (PTO) signal if there is a failure in its rail. A Reconfiguration Control Panel (RCP) allows manual control of rail primacy, and provides the means for overriding automatic reconfiguration. RCU and RCP operations are described in Section 2.4.

1.1.2. Intercommunication Concepts

The three levels of intercommunications in the assumed DCDS — intra-rail, inter-rail, and inter-regional — will be discussed below. Implementation details, such as message formats, will not be described. The reader is referred to other studies for detailed treatments.[1,2,4,5]

1.1.2.1 Intra-Rail Intercommunications

We have assumed that each GNCS computer communicates with the appropriate subsystems solely by means of a time-multiplexed serial data bus, and that there are no computer-to-subsystem paths of the dedicated wire or dedicated bus type. This is in line with the thought that messages between a GNCS computer and its subsystems are multiplexed at the computer I/O interface whether or not dedicated wires or buses are used. We are not really ignoring the possibility that some dedicated interfaces might exist; if so, they would have the same form as the dedicated inter-rail and inter-regional interfaces shown in Figure 1.1.2.

There are several design criteria which must be met, as well as decisions to be made in the near future.

1. Communications with certain subsystems, such as the Main Engine System, are critical, and it is necessary that these messages have priority over non-critical communications. This is a problem which must be addressed whether or not dedicated information paths are used; there are many straightforward solutions, one of which is described below.
2. It is also necessary to ensure that no subsystem malfunction causes all copies of the buses to be unusable, if the decision is made to cross-strap subsystems to buses; this too can be done.
3. The transmission code to be used on the bus must be determined. Manchester code, which is used in telemetry systems, seems popular with industry. Manchester code has limitations which have made it unsuitable for some schemes we have described in the past,[1,4] but it is certainly adequate for the system assumed here.
4. Another decision is bus bandwidth, although no problems are expected in obtaining adequate bandwidth. At least one Manchester coded bus exists which operates at 5 Mb/s,[7] five times the rate we have considered adequate in other studies.[1,4]

SEPARATE CHANNELS

Bus traffic is managed by a Bus Control Unit (BCU). In addition to performing reads and writes as requested by the computer, the BCU polls subsystems which do not require service on a scheduled basis, such as manual controllers. There probably are not enough of these in GNCS to justify a demand actuated polling scheme, and in any event Manchester[11] code does not lend itself to demand actuated polling. Instead, a round robin polling algorithm seems appropriate. If there are subsystems — the Main Engine System, for example — which require a quicker response time than can be guaranteed with a round robin, the algorithm can be modified to give them high priority. Subsystems would be polled in order of decreasing priority, and if any station replies "yes" the BCU starts polling again with the highest priority subsystem. The round robin is completed only when all stations respond "no." This is the algorithm that was used by the DCA to satisy a 200 microsecond response time requirement at the SIRU interface.[5]

In the remainder of this section, features which can be used to enhance the fault tolerance of a data bus are listed. Detailed treatments of these items are given in the referenced documents.

A single parity bit per bus message would provide good protection against an error occurring while the message is in parallel form, just as simple parity is used to protect data in a memory. But a failure during serial transmission could cause a "burst" of errors; a single failure could cause the bus to go to all zeros or all ones, half the time resulting in an even number of errors. Many codes have been developed which protect data against burst errors. One simple scheme uses two parity bits, one even and the other odd, at the end of each message.[4] In any event, a Bi-Phase code such as Manchester is considerably less susceptible to this failure mode than codes in which a single level is used to represent a bit. An acknowledgement scheme should be used to verify the receipt of an output message (computer to subsystem);[3] acknowledgement of input messages is not necessary. Another error detection technique is to simply count

the bits of a message; if variable length messages are used, bytes would also be counted.

Failure isolation could be aided by terminating the bus with a dummy subsystem which, by responding correctly to a read command, verifies the continuity of the signal path. Subsystems can be shielded from an anomalous command due to delayed detection of a computer failure by using "arm and fire" sequences for critical commands.[2,3]

1.1.2.2 Inter-Rail Intercommunications

The degree of data exchange among the GNCS rails has not been established. Minimal information exchange would be necessary if error detection hardware and software in a rail is relied upon to "tell on itself," and backup computers can read sensors to obtain knowledge of the condition of the vehicle. A higher rate of information exchange is needed if data must be passed from rail to rail to perform consistency checks for error detection, or if backups cannot interrogate sensors and must be told the state of the vehicle by the prime computer. Error detection schemes predicated on passing data between rails are discussed in Section 2.3. In any event, the rate of data exchange is not an issue which must be settled immediately. With the computer types contemplated, it would be easy to design the inter-rail interfaces for high-speed parallel transfers, as shown in Figure 1.1.2, and if this is done the decision as to how much of the capability is to be used can be postponed until the software design stage.

1.1.2.3 Inter-Regional Intercommunications

It is also necessary for GNCS, PMS, and PDCSto communicate with each other. Since it is understood that different contractors will implement these three regions, the communication path among regions whould be specified for the easiest possible interface control among contractors. Routing all such communications over an Inter-Regional Data Bus (IRB) serves this end, as shown in the DCDS block diagram, Figure 1.1.1.

There needs to be a mechanism for GNCS to report its health to PMS, and further exchanges between these regions are needed if keyboards and CRT displays are managed by PMS. Also, it has been suggested that primary preburn targeting routines be located in the PMS computers, and a data link between PMS and GNCS would be needed to support this partitioning. Traffic between GNCS and PDCS consists mainly of requests by GNCS to turn a GNCS subsystem on or off, routinely or because of a failure, and there would be similar traffic between PMS and PDCS.

It would be possible for PMS to determine the health of GNCS if the PMS buses ran to all GNCS subsystems, which could then be interrogated by PMS. But a better arrangement would be for GNCS to interrogate its own subsystems, perform some interpretation, and pass summary information as to its health to PMS over an inter-regional data path. GNCS is inherently better equipped for this task than is PMS, and if PMS development is postponed in the interests of smoothing the cost profile, GNCS would originally have the responsibility for determining its own health. Also, we are uncomfortable with the thought of cross-strapping GNCS subsystems to PMS buses. This would open the door to the kind of complex inter-contractor relationships we are trying to avoid. We want to avoid the situation where less critical PMS communication with a GNCS subsystem causes more critical GNCS communication with that subsystem to be delayed. It is our feeling that the design of a data bus system which is a secure medium for critical messages is a straightforward proposition, but more difficult if the bus must compete with a second, as yet undefined system for access to its own subsystems.

The IRB configuration discussed above furnishes an attractive mechanism to deal with a subsystem which has failed in such a way as to be unable to respond to a shut down command over normal communication channels. The GNCS or PMS can, as a last resort, use the IRB to direct the PDCS to power down the offending subsystem.

The IRB is envisioned as having a very low bandwidth requirement. The GNCS-to-PMS data transfers that indicate the health of GNCS are expected to involve just a few words, and the repetition rate should be low. Instructions to PDCS to turn subsystems on or off should be infrequent. If targeting is done by PMS, it would involve the transfer of only a few words back and forth between GNCS and PMS, even though the computation performed by PMS for this function may be extensive.

There is, however, a case for making the IRB a high capacity channel that would be expected to be underused. If mass memory devices shared among all regions are used, as indicated by proposed baseline changes, the IRB could serve as the communication path. If payload management and payload manipulation are to be performed by separate computers rather than by PMS, the IRB could be used to integrate these computers with the rest of the system. Also, if the IRB has adequate bandwidth, it is conceivable that an additional computer could someday be added to the system, connected via the IRB, to offload an overloaded region.

The use of the IRB for inter-rail (within a given region) traffic has also been considered, but we do not consider this a good plan. It is expected that the regions will be supplied by different contractors, with the idea that the total DCDS can be divided into several nearly independent efforts. If intra-GNCS communications are dependent on intra-PMS formats, for example, the efforts would no longer be independent, changes in one region would affect others, and inter-contractor disputes would be likely.

1.2 IMPLICATIONS OF SINGLE COMPUTER TYPE IN THE ASSUMED DCDS

1.2.1 Introduction

The use of a single computer type, as opposed to multiple computer types, for the different regions in the assumed DCDS will be examined in this section. Single computer type means that all computers in all regions are identical; that is, even the part numbers are the same. First, general advantages and disadvantages of a single computer type will be presented. Second, the problems of enforcing a single computer type for separate regions (and contractors) will be discussed, citing experiences from Apollo Guidance Computer enforcement history. Finally, implementation of single computer type and the effect that it will have on the DCDS will be considered.

1.2.2 Advantages and Disadvantages of Single Computer Type

The advantages and disadvantages of a single computer type can be discussed in terms of management, hardware, and software differences.

1.2.2.1 Advantages of a Single Computer Type

The advantages of a single computer type, or equivalently, disadvantages of multiple computer types, are largely based on the savings that result from elimination of replication of effort. In the case of management, this translates as less management overhead. Multiple computer types would necessitate a duplication of the complete management structure at the production facilities. Multiple computer types would imply the replication of teams of experts who know the "ins and outs" of each computer, and maintaining those teams for the life of the project.

Replication costs also figure in hardware advantages of single computer type. Single computer type eliminates duplication in the following hardware areas:

a. Documentation and training.
b. Qualification tests and procedures.
c. Radiation hardening and lightning protection.
d. Design of mechanical and electrical interfaces with the vehicle.
e. GSE and special test equipment.
f. Test procedures.
g. Packaging.
h. Thermal design.
i. Mechanical configuration.
j. EMI shielding.

(The last four areas have less impact if the computers are truly off the shelf.)

Another hardware advantage of a single computer type is that inter-region communication should be much simpler. Mass memory interfaces may also be simpler with a single computer type. Also, when considering system reliability, a single computer type implies dealing with a smaller total number of hardware design and production flaws.

Software advantages of a single computer type are similar in nature to those of hardware. Single computer type would eliminate replication of teams of software experts for the life of the project, as well as replication of generation, verification, and configuration control of the following areas:

a. Executive.
b. I/O routines.
c. Assemblers.
d. Compilers.
e. Simulators.
f. Documentation.

Finally, concentration of effort on a single set of software implies a smaller total number of software development flaws.

In the operational phase of the Shuttle the problems of hardware logistics are simpler for single computer type in the following areas:

a. Technology availability or obsolescence.
b. Spare inventory.
c. A documented history of retrofits to all computers.
d. Training and field service.

Operational advantages of a single computer type include more functional flexibility of the system, such as allowing the possibility of re-allocation of computer function both during development and in flight. Multiple computer types might well have different computer/crew interfaces. Unless these interfaces can be forced to be identical, a single computer type would have the advantage of avoiding the necessary additional crew training. Furthermore, the operational experience with more computers of the same type will provide better confidence in their reliability.
PLUS SOFTWARE

1.2.2.2 Advantages of Multiple Computer Types

The disadvantages of a single computer type, or equivalently, the advantages of multiple computer types for different regions, are based on the notions of flexibility and limiting the possibility of a single error type bringing down the whole system.

In the area of management, multiple computer types allow more managerial freedom, both in the choice of computers and in making changes to one regional computer without having to consider the other regions.
LOCAL OPTIMIZATION

With multiple computer types there is less possibility of a single software bug (in common software) bringing down all software systems. Similarly, for hardware, a single type of development flaw is less likely to bring down all computers.

The most influential argument for multiple computer types seems to be the ability to tailor the computers to the particular regions. That is, the hardware can be more efficiently utilized by designing or choosing the features of each computer to meet the needs of the particular region. Such features that may be considered in the computer design include the speed of operation, the memory size, word length, floating point, microprogram capability, and the instruction set.

To allow only one computer type would make it necessary that each computer include all of the capabilities needed for each region. The memory size would have to be large enough for the most demanding region, even though that may make the memory much larger than other regions require. Each computer will contain the special features of every region, even though each region does not make use of all special features. Finally, if microprogram is included, a single computer type probably implies identical microcode, which means region-specific instructions must be carried in other regions. Thus, it takes a combination of overlap and overkill to make a single computer be all things to all regions.

NOT SO! MODULAR

NOT SO!

Finally, the use of a single computer rules out selective upgrading in particular regions.

NOT SO! COMPATIBILITY YES!

1.2.2.3 Evaluation of Advantages and Disadvantages

In weighing the advantages and disadvantages of a single computer type, it is likely that the duplication of costs with multiple computer types will be the greatest argument in favor of a single computer type, while the inability to tailor the computers to each region seems to be the important argument against a single computer type. The duplication costs are real and unavoidable in use of multiple computer types. Tailoring to different requirements by using multiple computer types is subject to a number of considerations.

WHAT DOES GENERAL PURPOSE MEAN?

If different regions demand similar memory size, similar operations and features, and similar speed, then a single computer type may easily satisfy all regions. Obviously, the closer that the separate regional requirements are, the less influence the tailoring argument carries. If the only significant difference is in memory size requirements, it may be possible to tailor to this need without violating the single computer type concept, provided that the definition of a single computer type allows an external add-on memory. Other types of differences in requirements may tempt the system designer to use multiple computer types for the sake of tailoring such features as speed, interfaces, and instruction set. However, this is likely to be not worth the trouble. Selecting a single computer type that meets all such needs in all regions will probably incur negligible penalties of cost, size, power consumption, and reliability.

On the other hand, if regional requirements are very different, it may be undesirable or even impossible to have a single computer type. The decision of whether or not to have a single computer type is then based on a detailed analysis of these special requirements to determine if they can be compromised.

Subjectively, we feel that a single computer type for all regions is quite desirable, based on the factors previously discussed. Aside from the cost factors discussed, the uniformity imposed on the system by a single computer type seems technically desirable (of course, this also translates into lower cost). By the same reasoning, if it is decided that multiple computer types are necessary, minimization of the number of computer types is equally desirable.

1.2.3 Enforcement of Single Computer Type: Apollo Experience

Given that a single computer type is desirable for two or three regions, the contractors and designers of the individual regions must each live with

that computer. The problems and methods of enforcing adherence to an established computer design must be considered at the outset of the program, or it may become difficult or impossible to adhere to a single computer type. The Apollo Guidance Computer (AGC) history is a useful source of experience on the subject of enforcement.

The AGC is the one computer which was imposed on both the Apollo Command Module (CM) and Lunar Module (LM) guidance systems. In this case, a single computer type was more or less acceptable since both modules were to have the same guidance system. Here, the systems were similar enough so that the tailoring factor was minimized and the elimination of duplication was a strong desire. A brief summary of the Apollo history illustrates the difficulty of making and enforcing the decision for common computers, even in this rather ideal case.

NASA, of course, managed the Apollo project, and initially MIT and North American were the contractors for the guidance system and the CM respectively. Later Grumman came in as the LM contractor. At this point it was necessary for NASA to impose a common guidance system and computer on the two spacecraft designers if uniformity of design was to be acheived. NASA initiated and chaired a series of implementation meetings where the three contractors worked out the requirements of the system. A Block I AGC had already been designed for the Block I CM, but its functional capacity was not adequate for the LM guidance and navigation system. In the absence of strong NASA management it would have been very easy to suggest a different design for the LM, leaving the Block I system in the CM. NASA continued to press towards a common design even though it required a new design. All changes that led to the Block II AGC required concurrence by all four parties. During this period of functional definition, the NASA management decision to use common computers was continually questioned and had to be re-enforced. During the development phase, the instrument of control was the Interface Control Document (ICD); it had to be the same (by NASA decree) for each system.

Once most of the computer design was frozen there was not much more enforcement difficulty because things were then accepted and lived with. Most of the problems, after the initial decisions, were related to the interface specification since that was the last thing frozen, but these were not allowed to impact the computer hardware design.

It was necessary that the Apollo computer be designed from scratch, so MIT had the flexibility to try to accomodate the contractors' requirements. In going from Block I to Block II, changes to the AGC included memory size, speed, op codes, interfaces, and design margins to accommodate the addition of the DAP. Consequently the design of the AGC was considered flexible and this added to enforcement difficulties. However, changes to the computer or changes to a spacecraft subsystem to bring it in line with the computer were expensive and would result in changes to both spacecrafts. Therefore, because of the strong NASA management desire for a common system, enforcement was necessary on a continuing basis.

The Apollo experience provides some lessons in enforcement of a single computer type. First, it is necessary that there be one top level group willing and able to make the decision and enforce it. Second, the computer should be chosen early in the project, and RFPs (request for proposals) for subsystem contractors should be constrained to conform to the computer decision. Third, if possible, all system requirements should be considered at the outset, such that a Block I-Block II situation is not forced by an unplanned change in requirements. Fourth, the choice of an off the shelf computer should aid enforcement, since most of its characteristics will already be fixed, thus eliminating attempts to modify the computer.

RECALL S-IB STRUCTURAL INTERFACE FIXED.

One obvious difficulty is choosing the computer before the subsystems are completely determined and their contractors selected, since the choice of computer must be made without knowledge of the details of the system.

In the case of the shuttle DCDS, where an off the shelf computer will be used, the choices of computers seem limited enough so that an intelligent choice can be made without detailed knowledge of the subsystems. Such considerations as reliability and space qualification should further limit the candidates. Finally, based on past experience (Apollo), reasonable decisions of computer size, speed, and features can be made and used to select the computer.

In order for a single computer type to be enforced for the shuttle, NR, as prime contractor, must do the enforcing. To make enforcement possible, NR must dictate at the time that the subsystem contracts are negotiated that a single computer type will be used, and preferably what that computer will be. This is especially important since it seems that many bidders for the subcontracts will be in the computer business and naturally will want to use their own computers. After the contracts are let, and the project progresses through the actual design and manufacture stages, NR must not allow any changes which will defeat the concept of single computer type.

1.2.4 Effect of a Single Computer Type on the Assumed DCDS

The assumed DCDS is more centralized than the various configuration which have recently served as baselines. Flight Control and Guidance, Navigation and Control have been collapsed into one region; Payload Management, Payload Manipulation, and Display Management may be functions of the PMS (Performance Monitor System). The resulting computers are required to perform all the tasks formerly done by several machines. This tends to neutralize one advantage of different computer types – that the computers could be tailored to their specific tasks.

For example, if one task uses 16-bit data exclusively while another requires the precision of 32-bit data, it makes sense to use different

HALF word addressing

computer types when these tasks are done by separate computers. To impose a 16-bit machine on the 32-bit task, or vice versa, would lead to programming difficulties and inefficient storage utilization. But if both these tasks are to be done by one computer, the machine chosen should be able to process data in either 16-bit or 32-bit units. Clearly, the more tasks performed by an individual computer, the closer the computer must be to filling a comprehensive set of requirements.

In the assumed DCDS, the requirements on the computers in the GNCS and PMS regions are similar. It is true that GNCS needs floating point operations, while if there are floating point instructions in PMS they might never be executed; and PMS probably requires a higher I/O bandwidth. If it were necessary to trade floating point against I/O capability, the case for using different computers would be clear. But computers are available which meet both requirements, without sacrificing one for the other. Another consideration is that GNCS targeting routines may be located in the PMS machine; if this plan is followed, floating point would become an PMS requirement also.

If a single computer type is used for the GNCS and PMS, it is possible and desirable to have single types of all common system components, such as the RCU and BCU. This will simplify the inter-region communication and simplify the subsystem communication. As a result of a single type BCU and Data Bus Subsystem I/O, all interface problems will be constrained to the subsystem SIUs. That is, the individual interface problems of a subsystem are removed from the main computer to the SIU associated with the subsystem. This is reasonable, since it is at the subsystems that the regions will differ.

1.2.5 Dissimilar GNCS Backup

This report has addressed the implications of a single computer type for all regions. The advantages and disadvantages of a dissimilar backup

within the GNCS region are similar to arguments presented so far, except that the desire to eliminate common mode failures carries a greater weight. As a result, it needs to be pointed out that there is a very limited set of hardware failure modes which would be common to all redundant copies of GNCS computers and would be eliminated by the introduction of a dissimilar copy. Failures due to design errors are a very important class that would be eliminated. However, these should be detected and corrected early in development. On the other hand, some common mode failures not eliminated by the introduction of a dissimilar backup are erroneous software, erroneous specification of system parameters, erroneous crew procedures, and transient disturbances. The latter may be either induced by external interference (lightning, power supply transients, radiation burst, or other environmental excesses), or may originate within subsystems so as to produce erroneous data at the computer interface. (Part II deals with methods to minimize disturbances caused by lightning.)

In general, common mode failure problems must be solved for all computers. Solving them just for a dissimilar backup is advantageous only if the backup is much simpler than the primary. Since this seems unattainable for the GNCS, the effort to eliminate these classes of failures would seem better directed toward the primary computers.

1.2.6 Candidate Computers

The application of single or minimal computer types will now be considered with respect to a population of candidate computers. These computers are the AP-101, SKC-2000, HDC-601, and the HDC-701.

If a single computer type is to be implemented throughout the DCDS, exclusive of the engine controller system, either the AP-101 or the SKC-2000 must be chosen, because neither the HDC-601 nor the HDC-701 is large enough to contain the entire GNCS. Furthermore, the Honeywell computers do not include floating point, which is highly desirable for GNCS. If the

AP-101 or SKC-2000, either of which is capable of performing the GNCS task, is chosen, the selected computer would also satisfy the PMS requirements. This is especially true if computers which consist of a common CPU but different memories are considered a single computer type. For example, the AP-101 can be packaged with 8K, 16K, 24K, or 32K words in the ATR box; add-on memory units at 32K words per ATR box, up to 128K of memory, would be compatible with the architecture.

If it is decided that the AP-101 or SKC-2000 is so powerful as to be wasteful for less demanding regions, a computer such as the HDC-601 might then be used for such regions. The system is then a two computer type system, and it seems that this is the greatest number of computer types that is desirable or necessary. Each region could then choose among the two computer types for the one most suited to that region.

It may be argued that since the HDC-601 is already included in the system as an engine controller, use of the HDC-601 elsewhere in the DCDS would not be adding another computer type. In the context of this section, that is not valid. The engine controller consists of the HDC-601 components packaged with the engine control electronics, rather than the standard HDC-601 package, and thus would not be considered the same type as an HDC-601 computer.

2.0 REDUNDANCY MANAGEMENT CONCEPTS FOR THE ASSUMED DCDS

Redundancy management is a term that refers to the techniques employed in a redundantly configured system to make the redundancy work -- to use the redundancy to increase reliability and to gain fault-tolerance. Redundancy management attempts to optimize various parameters related to the operational integrity of the system. Among these parameters are coverage, speed of detection, speed of recovery, smoothness of recovery, reconfiguration flexibility, and overall system reliability. Some of the techniques for redundancy management which are discussed in the following sections are: built-in-test-equipment (BITE), built-in-tests (BITs), consistency tests, and reconfiguration control units (RCUs).

The techniques that will be discussed are concerned with improving system coverage (the ability to detect and recover from an error). Coverage is the single most important parameter next to simplex reliability in determining the overall reliability of a redundant system. In order to obtain the prescribed mission success and crew safety reliabilities for the DCDS, within the general framework of the assumed configuration, judicious use of redundancy management techniques is required.

At present, the NR requirement for detection and recovery from errors in the orbiter avionics is 0.4 seconds. We feel that the detection aspect should be accomplished in no more than 0.1 seconds. This number is chosen for several reasons: a critical failure indication analysis by NASA (MSC Internal Note EG-71-38) mentions this number several times as a goal for detection and recovery; after detection has been accomplished some time must be allocated to recovery -- if re-initializations are necessary they could consume most of the allotted 0.4 seconds; the faster the detection time, the better the coverage parameter -- this is critical to overall system reliability; we feel that 0.1 seconds for detection

is not an unreasonable goal.

Even if reconfiguration is done manually and takes seconds to accomplish, certain safing procedures must be accomplished rapidly (hence automatically) to maintain vehicle integrity. Rapid error detection is necessary if such safing is to be done properly.

The question arises as to whether or not the present NR avionics configuration can achieve the required mission success and crew safety reliabilities. An important part of the answer centers around the coverage that can be achieved by this configuration.

2.1 IMPROVING COVERAGE (ERROR DETECTION AND ISOLATION) OVER THAT AFFORDED BY BITE

The GNCS portion of the assumed configuration is composed of three "loosely coupled" computers, each of which has its own bus for communication with subsystems. Only the outputs of the prime computer are actually sent onto a bus; the outputs of the backup computers are inhibited by the Reconfiguration Control Unit. In the event that an error is detected in the prime computer, the Reconfiguration Control Unit designates one of the backup computers as prime. Designation of the prime computer can also be done manually. Knowledge of primacy is not contained within the computers themselves -- this allows them to operate in the same manner. Error detection is accomplished by BITE in the respective computers.

BITE can be fairly effective for detecting errors. Industry figures indicate that .95 coverage (over the appropriate time frame) is not an unreasonable expectation.[2] As good as this may sound, however, our studies indicate that a coverage of .99+ is needed even to approach the shuttle reliability requirement (see Part III). The recommended approach to "beefing-up" existing computer BITE in a loosely coupled system is to include opinion-of-performance testing in the software. Opinions-of-performance fall into two categories: reasonableness tests and consistency tests.

A reasonableness test compares the result of an operation or computation with a value or range of values which have been determined a priori. For example, determining whether or not an indexed operation falls within valid address limits is a reasonableness test. Comparing orbital parameters to an expected mission profile is also a reasonableness test. Reasonableness tests suffer

from the basic weaknesses of BITE in that they operate entirely within a single computer and require the prediction of error modes. They are unlike BITE in that, generally, they do not require a commitment to hardware, so they can be applied selectively.

A consistency test compares the result of a computation with the results of the same computation done by another computer. When three or more computers take part, the result is a "software vote." A tolerance (an amount by which the results from each computer can legally differ from one another) is required, but the need for an a priori prediction of an absolute range for these results is eliminated. This approach to error detection provides the benefits of voting, but can be applied selectively where needed without a committal to hardware. Some of the conceptual aspects of implementing consistency tests are outlined below.

The most urgent need for arbitrarily high coverage (and thus for consistency tests) is in detecting erroneous critical output commands. Since such commands are often collections of discretes rather than numerical values, the concept of a tolerance unfortunately does not apply. Accordingly, a consistency test must either assume bit-by-bit equality of the outputs of each computer, or be able to compare the "maneuvers" specified by each of these outputs. The latter alternative is, clearly, very complicated, but the fact that the computers are not phase-locked to the bit level means that commands generated by flight control algorithms will not always be exactly the same, in order or in value.

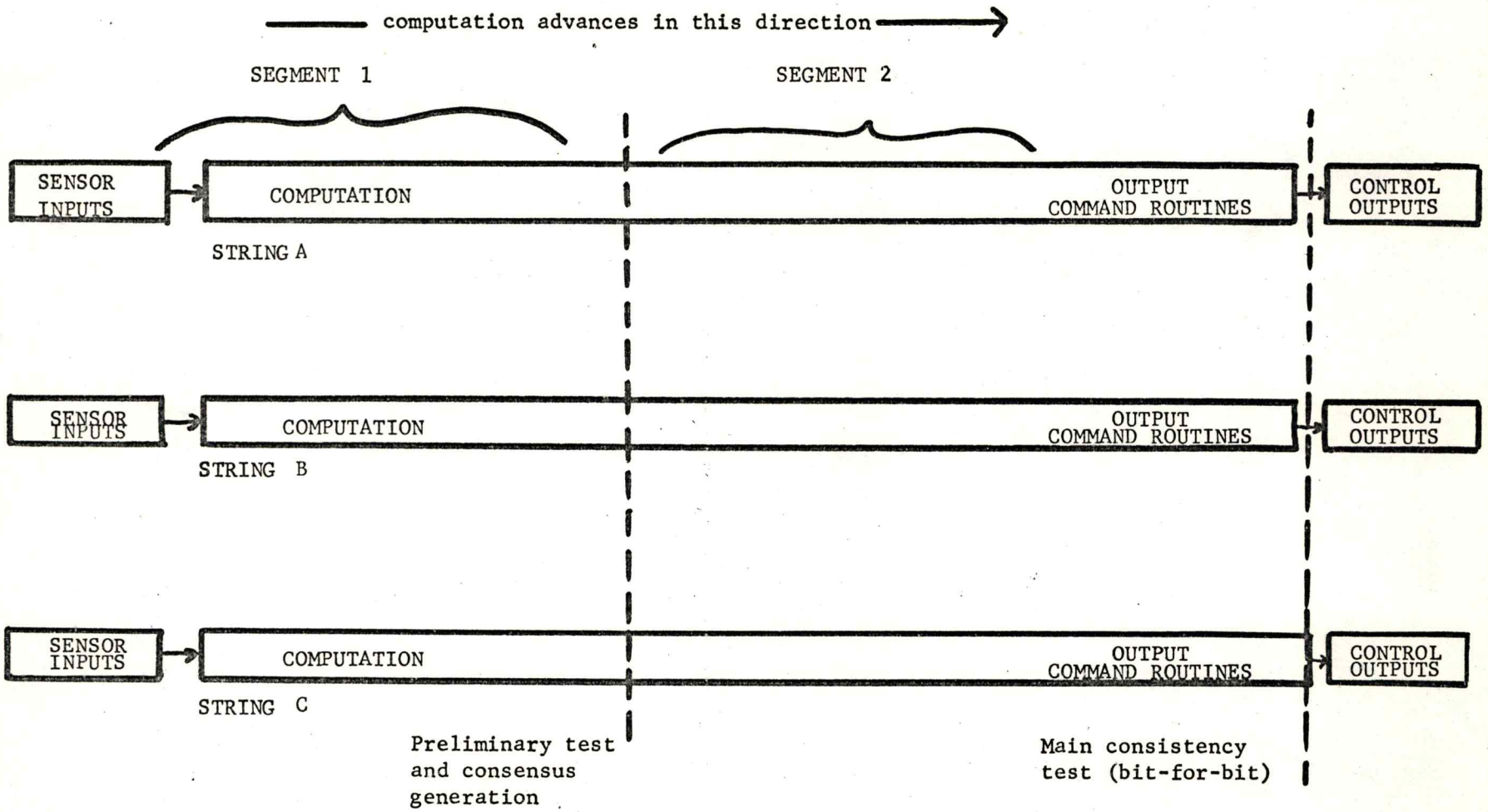
The easiest solution to this problem is to provide a common set of input values to the output command routines in each computer. Identical input values will result in identical output commands, as long as the computers are functioning properly. The common value is generated by employing a preliminary consistency test at a point in the computation cycle where numerical variables can be readily compared. This preliminary "test" does two things: first it checks that the values from each computer fall within a certain tolerance of one another; second, it computes a "consensus" of

those values which meet the tolerance criterion. There are various algorithms for determining consensus, e.g., averaging and mid-value selection. None of them are very complex, and the selection of one or another will probably depend upon the nature of the sensors and computations that provide inputs to the control loop. Since the consensus value is used by each computer as input to the next step of compuation, the outputs of the command routines should all be bit-by-bit equal. The main consistency test can now be applied to the output values with a zero tolerance criterion (bit-level equality). This is effectively a software voting scheme. The details of testing and consensus generation are described in Section 2.3.

Figure 2.1.1 depicts the relation between the two consistency tests. In Segment 1 the computer operations exhibit the characteristics of "loose coupling": sensor data used by the computers is slightly different and the order of some operations may be different in each computer. Because of this, the operations in Segment 1 have a resistance to many common mode failure mechanisms -- particularly those mechanisms involving pattern sensitive data and subtle software timing characteristics. However, the fact that inputs to the computations are not equal allows for the possibility that the computations will diverge.

In Segment 2 each computer operates on identical data (the result of a "consensus") so the computations in each computer will be identical. This solves divergence problems, but does not provide any resistance to common mode failures.

The placement of the preliminary consistency test (and consensus generator) determines the relative sizes of Segment 1 and Segment 2. Appropriate placement strategies are yet to be determined. One suggestion has been that the preliminary test be made on sensor input information (thus eliminating Segment 1). The rationale given for this approach is that it allows the detection



ONE ITERATION OF A COMPUTATION LOOP
SHOWING CONSISTENCY TEST PLACEMENT

Fig. 2.1.1

and elimination of sensor biases, and that the tolerance for the test is easier to specify before possible computational divergence takes place. Another approach is to place the preliminary test as late as possible in the computation loop -- just prior to the output command routine. This provides resistance to common mode failures for most of the computation cycle.

In summary, it can be seen that consistency testing is a very valuable tool for increasing the coverage provided by BITE so as to meet the necessary reliability requirements. Consistency testing has the following advantages over other possible approaches to support BITE:

1. There is no commitment to hardware synchronization and voting.
2. Consistency tests can be employed selectively -- the decision to apply them to any particular process can be put off.
3. There is no impact on the basic hardware configuration, other than the addition of an intercommunication channel among the redundant computers.
4. Each computer executes the same program so it is not necessary to verify dual mode software.

2.2 BUILT-IN TESTS

2.2.1 Introduction

In order for the redundant strings of the GNCS to achieve fault tolerance, it is necessary that some mechanism exist for the purpose of detecting faults or errors throughout each string in order that recovery may be initiated. With one exception, the error detection methods proposed for the DCDS and described in this section are known as Built-In Tests (BITs) or, in reference to additional hardware added for error detection, as Built-In Test Equipment (BITE).

2.2.1.1 Hardware BITE

One important means of error detection is hardware BITE; that is, detection by equipment additional to normal computing hardware. Some basic important examples of hardware BITE will be discussed later in this section. Intuitively, the more BITE employed to detect errors, the better the probability of detecting any random error. On the other hand, for each piece of hardware added to BITE, the problem of verifying the correct operation of the BITE itself can be a difficult one. Furthermore, each addition to the total part count due to additional BITE must lessen the reliability of a string.

2.2.1.2 Software BITs

Software BITs involve including in the programs tests or exercises which help to verify the status of the string. Software BITs are used in addition to, and in conjunction with, hardware BITE to increase the likelihood of error detection. For example, self check routines may test each memory location, or may test arithmetic units by making calculations and comparing to predetermined results. Another example of a software BIT is reasonableness tests, which will be discussed later in Section 2.2.5.7. All tests mentioned so far are "built-in," meaning that they are complete in one

computer string. On the other hand, the individual strings may cooperate with each other to see if their views of the system agree. Such techniques are called consistency tests and are discussed in Section 2.2.6. Software tests do not decrease the reliability of a string by requiring special additional equipment (as BITE does), but they may have this effect if memory must be re-sized to accomodate them, since memory size contributes strongly to failure rates. Furthermore, extensive software tests involve coding and verification costs and risks, and the associated computing overhead may interfere with primary programs.

2.2.2 Coverage

The reliability of any multi-string system is a function of the reliability of the individual strings, the reliability of switching primacy from one string to another or restarting the same string, and the reliability of detecting and signaling an error within a string. The latter parameter will be referred to as detection coverage or simply coverage (coverage usually refers to detection and recovery, but in this section we will use the term to refer to detection only), and is used to characterize the effectiveness of hardware and software Built-In Tests. Coverage is measured as a percentage of errors detected within a specified time. In this definition there are three variables that must be determined before the coverage number is meaningful.

First, the percentage of errors may be calculated as a ratio of the error types detected to the total possible error types, or each error type may be weighted by its relative frequency. Error types are distinguished by means of detection, regardless of failure types involved. For instance, it is felt that errors in memory will occur more than other errors, so in calculating coverage such errors may be given relatively more importance (and therefore better detection techniques generally are applied to memory errors). The relative weight method of calculating percentage of coverage seems more suitable for use in computing system reliability.

Second, the general class of errors being measured by coverage must be defined. In this section, coverage will refer to single-point random failures; that is, we will be concerned with only one failure at a time. Double (or multiple) failures which occur at the same point in time, or so close in time as to be effectively simultaneous, and combine diabolically to defeat the BITE, are so unlikely that they will not be considered in coverage. Catastrophic errors, where the system is massively interfered with, as by radiation, or physically damaged by external forces, are not addressed by BITs; however, such events are expected to produce some errors that are detectable by the BITs provided. If there is an undetected failure in the BITE hardware such that the failed detection device will never signal an error, a second error which would have been detected by the failed detection device can occur at some future time and go undetected as a result. This is known as a "Pollyanna" failure. An example of this is a failure in the parity checker such that a word with bad parity will not be detected or signalled. We will assume that a Pollyanna failure will be considered the same as a normal undetected error with respect to coverage, even though it may never affect the system adversely. This assumption is not major, since if little hardware is used for BITE, it is considerably more reliable than what it is testing, so it does not affect overall reliability calculations very much.

Finally, the time within which an error must be detected is an important parameter which must be defined in order for coverage to be meaningful. Obviously, the longer the allowed detection delay, the higher the error coverage figure will be. This is because as an error propagates it will tend to cause other errors (wrong values); and if a failure persists, it will cause many errors, thereby increasing the likelihood of detection. In the synchronized voting systems we have previously discussed[2,4,8] we claimed coverage was 100% for single point failures because discrepancies are detected within the instruction step or before causing erroneous outputs. This is the most demanding coverage detection time, and any system based on BIT can only approach it.

Our criterion for coverage for the GNCS will be detection, within 0.1 second, of errors caused by single point failures. This figure is based on the assumption that in a time-critical mission phase an error must be detected within 0.1 second to insure a safe recovery.

2.2.3 Hard Failures

Computer failures are either hard or transient. A permanent or persistent failure is such that under the same conditions the same errors would occur in the future, and the same failure will cause many other errors in most related situations. Hard failures may be caught immediately by some error detection logic (such as parity), or they may be discovered after a short time delay by another BIT (such as a timing or activity alarm), or they may be exposed by a test routine which exercises a failed piece of hardware (such as self check). It is felt that BITE can do quite a good job against hard failures, given a liberal detection allowance such as 0.1 second in which the system can detect that it has failed.

The exception to detecting anticipated errors may be the test equipment itself. The basic problem to testing the test equipment is to see that an error presented to the BITE will cause detection and request switchover to a new string. One suggestion is to induce an error while in some way inhibiting switchover, but then the inhibitor must also be tested for failure, and so on. Another suggestion is to actually force a switchover, and observe that it occurs. As far as we know, this may be the only way to check out the test equipment. But this would be totally impractical if the 0.1 second detection criterion must be met, since continuous testing would cause continual switching, and in addition, the original prime must be made to reappear healthy to the other strings. Therefore it is felt that BITE must be made simple and reliable enough so that it does not detract materially from coverage and system reliability.

By way of contrast, it should be noted that a synchronized and voted system minimizes in-flight testing problems by use of redundant voters. The correct operation of voters still needs to be verified; nevertheless, the ad hoc and distributed nature of BITE makes its correct operation much harder to verify than that of voters.

2.2.4 Transient Failures

Errors can be caused by an external influence or by an intermittent failure in the hardware. By the nature of these errors they are more difficult to detect and may even be indistinguishable from good data. For instance, an instruction may be altered, or the location counter affected in such a way that the correct sequence is not followed. Data may also be altered and eventually be used incorrectly. Since transients are by definition not repeatable, it is difficult to test for them.

Decisions must be made as to how much BITE can or should be added to protect the system from transients, what the relative importance of transients is, and how good the protection is. It is difficult to study the impact of transients because, by their insidious nature, they defy detection and identification of origin.

Parity may catch one half (or more) of all memory transients (assuming that multiple bits may be affected), and transients will more often than not be so gross that they may be easily detected. Thus, much of BITE intended to catch hard failures may also be useful in detecting transients. Features like reasonableness and consistency testing may do much to protect the system from transients, especially in critical areas.

An important and difficult question is whether BITE offers good enough coverage for transients as well as hard failures. The effective prevention of transients (i.e., reducing their probability of occurrence to a negligible level) may be the answer to the problem. It is felt in some quarters that

electromagnetic interference (EMI) is the major cause of transients (e.g., the Apollo 12 lightning experience). If so, effective shielding, grounding, and electrical isolation, as described in Part II, should eliminate this class of transients. However, others feel that intermittent hardware failures may be a major cause of transients, perhaps of equal importance with hard failures. That is, a hardware component may fail intermittently before it fails hard. Elimination of intermittent failures is not possible and the BIT methods must be used to detect these errors. On the other hand, the coverage provided by BITs may be unacceptable.

The remainder of this section will be devoted to presenting methods of Built-In Tests and their effectiveness.

2.2.5 Types of Built-In Tests

Built-In Tests can be generally classified as software or hardware implemented, although cooperation between the two is often required for a specific test. Tests also have the characteristic of either immediate detection or delayed detection of errors. Following is a description of the basic important types of Built-In Tests. Hardware tests are in effect at all times, or concurrent, and can cover many situations, while software tests are program specific and periodic. Tests with immediate detection assure that the error will be detected within any critical time, instead of depending on the error to propagate in a way that eventually will be caught by a test with delayed detection. These characteristics will be noted in the descriptions below.

2.2.5.1 Parity

Parity is the most commonly used type of Built-In Test. By appending one or more bits to memory locations and I/O messages, and including the appropriate parity checking hardware, errors in individual words may be detected. Parity may also be included in the microprogram memory.

One parity bit will detect any single bit error within a word, including an incorrect parity bit. Thus parity generation hardware is also covered for failure.

Memory is probably the part of a computer most susceptible to failures. The fact that parity alone can completely cover single point memory errors shows its importance with respect to the coverage of Built-In Tests. In addition, parity is generated and checked completely in hardware and is effective at all times. Finally, since the parity checker is an instantaneous detection test, it detects all odd bit transient errors, as well as hard errors, in the areas it covers. This is also important, since active memory and transmission lines, if not properly designed, are especially susceptible to transients. There are also higher order codes, such as Hamming codes, which cover more failures, but involve more complex coding and decoding hardware. Parity checking does not cover address selection, which can be critical, especially in interference situations (see Part II).
IF CAN, INCLUDE P(ADDR+DATA)

2.2.5.2 Watchdog Timers and Activity Alarms

Activity timers are useful for detecting errors in normal program procedures. By use of counters or timers, the activity of specific events may be constrained to occur within specific bounds. For instance, correct program activity may require that a counter be reloaded before it counts down to zero. If it ever reaches zero, it will activate the program activity alarm. Cooperation by the software is required so that in normal operation the counter will always be reloaded. Thus, if any failure occurs that causes errors in the instruction sequences or causes an infinite loop, the program activity test should detect the error within the 0.1 second criterion. The same kind of activity alarms may be devised for other areas, such as interrupt activity or program transfers, so that they can detect an event occurring too often or not often enough.

Observe that in 0.1 second, a machine whose throughput is 500 K op/sec (i.e., 2 microseconds per average instruction) will perform 50,000

instructions. It is felt that in 50,000 instruction executions, most hard CPU failures will eventually cause an error which can be detected by strategically designed timers and activity alarms, since nearly all CPU components are widely shared among many instruction types. For example, if an adder failed, the first error caused by that failure may be a wrong sum which may never be detected. However, if the same adder is used for address modification, it will shortly upset the instruction sequence so grossly that some activity alarm will detect an error.

This class of Built-In Tests will be a significant method of detecting errors in the CPU. The tests are hardware implemented, but, as previously mentioned, may depend on software cooperation. The activity checks are always in effect, but typically provide delayed detection, since the abnormal activity or inactivity is usually a kind of infinite loop which must persist for a considerable time to be recognized as illegitimate. The time of occurrence of the error that caused the loop is necessarily prior to the looping, possibly by a sizable interval. Consequently, it is impossible to say that activity or timing tests detect errors within any specified time after their occurrence, especially in the case of transients. The power of these tests derives from the fact that the looping phenomenon they do detect is the result of a great variety of error modes, foreseen and unforeseen.

2.2.5.3 Output Feedback

Output feedback is a Built-In Test by the sender of an output to see if in fact a message was sent, and if it was sent correctly. This is actually a limited form of message verification.[2,3] The output is read back in by the sender and tested by checking parity or direct comparison. Direct comparison can be most easily and efficiently implemented in the microprogram, and may be used in conjunction with or in lieu of message parity.

Output feedback is either software or hardware implemented (preferably hardware) and offers immediate detection, thereby detecting transient

errors as well as hard failures. When hardware (or microprogram) implemented, output feedback provides automatic coverage for all output data paths, and it provides effective coverage at the computer and subsystem interfaces.

2.2.5.4 Other BITE

There are many more hardware tests for specific but important error modes. Some examples are: power supply monitor, oscillator monitor, frequency divider alarm, and event failure timing. Many of these features are commonly available in aerospace computers.

2.2.5.5 Self Check

Self check is a set of software routines designed to periodically exercise and test areas of the computer, such as memory, control pulses, or arithmetic logic units. Self check can be divided into two classes of tests: those which exercise the computer in such a way that any failure would cause an error that will be detected by one of the hardware Built-In Tests, such as memory parity, and those which actually perform operations and check their results against prestored values in the self check program, such as software testing of the arithmetic logic unit.

By their nature, the more extensive the self check routines are, the more memory they use, and the more time they take to run. In order for self check to add to the coverage of Built-In Tests, it is suggested that it be run at least once every 0.1 of a second to fulfill the coverage time criterion. However, since the coverage discussed was defined based on time-critical mission phases, a dilemma appears: when self check is most needed it can be afforded the least.

There are two different ways to respond to this dilemma. First, although in the past, time-critical periods have usually been associated

with a fully loaded computer, this is not necessarily so any more. Present computers should be fast enough, and large enough, to allow self check to run periodically, even in time-critical phases. Now, time-critical refers to the necessity of the computer to respond quickly to any error to avoid disaster, and self check may be an integral part of this response. Present estimates of the impact of self check when executed every 0.1 second with microprogram assist (see Section 2.2.7) is less than 2% of computing time. The argument that the critical-time programs will always expand to fill any void in computing time must be countered by the argument that such situations are now unnecessarily dangerous and should be avoided by careful program control.

The second response to the dilemma is to observe that not all of self check is useful during critical periods. Specifically, the exercise portions, which depend on BITE, add very little to critical-phase protection. The same BITE would protect the mission programs nearly as well with no help from self check, while the exercise routine might excite an error mode that would not have come up in normal operation; this would cause a string switchover during the critical phase that is an unnecessary and dangerous nuisance from the mission point of view. The testing portions of self check, however, are just as essential during critical phases because they augment the BITE.

Self check is in effect only periodically, and is probably of little use in detecting transients, since it would only detect a transient that occurs within the appropriate self check test.

In summary, self check is a software BIT which can be made quite efficient by the aid of special microprograms, as explained in Section 2.2.7. It consists of exercise routines, which undertake to excite error modes that will be caught by BITE, and testing routines, which excite other types of error modes and catch them by comparing results with predicted answers. Self check should be considered a form of delayed detection since it does not directly detect errors made in normal operation. Instead it immediately quickly

detects errors occurring in its own operation and signals a failure on the assumption that similar errors are occurring undetected, or will occur, in normal operation. Thus, the exercise routines are most valuable when most easily afforded, that is, to check out a string before entering a time-critical phase and get any necessary switching done before the critical phase. The testing routines add to the total coverage and are therefore equally valuable at all times.

2.2.5.6 Software Error Tests

Software error tests include such tests as addressing out-of-bounds, improper underflow or overflow, writing into protected memory, dividing by zero, illegitimately executing privileged-mode instructions, using instructions as data or vice versa, and branching to improper locations (see Ref. 3, pp. 196-8). The stated purpose of such tests is to detect software errors, but in case of transients that look like software errors, these tests are also effective. Microprogramming facilitates implementation of these tests.

2.2.5.7 Reasonableness Tests

Reasonableness tests are software tests of computer internal results and outputs which test to see if a parameter is consistent with a predicted range. The philosophy behind reasonableness tests is that a failure-induced error in any parameter is more likely to be gross than small. Usually, a parameter is known to have to be within certain bounds, either by nature of the parameter alone, or as a result of the mission phase or program being used. For example, the angle of a gimbal must fall between 0 and 360 degrees, and any other value is obviously an error. Including within the software a direct test for this range will detect any error which is not within bounds. The more that the reasonable range of any parameter can be reduced, the more likely that the test will catch any error. However, if a correctly computed value falls outside its redlines because a redline is too restrictive, it will take down all similar strings; that is, a wrong

redline is a type of common mode specification error. In fact, this occurred in the first Lunar Module flight.

Reasonableness tests suffer from some of the same problems that self check does. They involve additional software, implying generation and verification costs as well as memory use, and additional computation time. As they are more extensively used, they take up larger portions of time. Since reasonableness tests are specific to mission programs, they must be run in time-critical phases to be useful. It is possible that reasonableness tests may represent a few percent of the mission program time, if extensively used.

Reasonableness tests are specifically designed for individual parameters, and therefore the important parameters must be chosen and the ranges predicted. One advantage of this specificity is that the tests are only run when useful. Reasonableness tests are completely software implemented, and can often be placed to achieve immediate detection. This last fact implies that reasonableness tests are very useful for detecting transient errors.

2.2.6 Consistency Testing

Consistency tests, which will be described in detail in Section 2.3, compare different versions of some quantity computed on different (though similar) equipment, rather than comparing computed data with pre-specified values in the same equipment that produces the quantity (reasonableness tests). Consistency tests require hardware-software cooperation, and depend on two or more active strings. Detection is often delayed and coverage is selective, but can cover any or all outputs. Consistency tests may be particularly useful for transients (except massive interference affecting all strings). Of course, they also serve as a backstop for hard failures not caught by Built-In Tests.

2.2.7 Microprogramming Aids to Built-In Tests

Microprogramming aids Built-In Tests in the sense that it offers a faster way to execute some tests as well as compacting the storage these tests require, but does not seem to offer much in the way of new methods of Built-In Tests without additional hardware. It has already been mentioned that microprogramming may be used to implement some of self check, and that software error tests and output feedback may be automatically carried out by the microcode. These capabilities are useful, and may be imperative for an effective self check.

A computer with microprogram capability should be beneficial to the efficiency of self check programs in two ways. It should provide faster execution of the self check and reduce its storage requirements in main program store. If portions of the self check are coded in microprogram, the overhead for decoding machine level instructions is reduced. Also, it is often necessary in program level self checks to spend several instructions to set up the proper conditions to test a specific case. Microprogram should be able to set up the desired conditions much more directly and save these instructions completely. This latter point should also reduce storage requirements in the main program for self check.

One possible use of microprogramming for Built-In Tests may be to test arithmetic and logic operations. Within the microcode of each arithmetic instruction may be included an inverse operation and test. For example, an addition instruction could subtract one operand from the result and compare that result to the other operand. Carrying out this microprogrammed procedure for each instruction will certainly reduce throughput, though by much less than a factor of two. The main overhead in an instruction is the fetch and setup cycle. The additional test operation will not require any additional fetch cycles, so the complete instruction time is increased only by the time needed to run the additional microcode. However, the test operation might require additional temporary registers that are not available.

The incorporation of arithmetic checking in the microcode should reduce the need for self check program testing of those arithmetic units. With regard to the earlier discussion of self check during time critical phases, arithmetic checking in microcode should be faster and more effective, and may partially replace self check programs needed during time-critical phases.

2.2.8 Built-In Test Characteristics of Sample Computers

Table 2.2 presents a summary of the Built-In Test characteristics of the following computers: AGC, DCA, IBM AP-101, SKC 2000, and HDC 601. The latter three systems are candidates for use in the Shuttle avionics; the AGC and DCA are included as a basis of comparison of Built-In Test effectiveness. In the table, Type I coverage is the probability that any failure is detected immediately, so that all errors caused by that failure can be localized or quarantined, usually to the one data word or register affected. Type II coverage is the probability that any error is detected within 0.1 second.

2.2.8.1 Coverage Study of the IBM AP-101

From Table 2.2 it is apparent that the AP-101 has been designed to provide good coverage by means of BITE and flexibility through microprogramming. For this reason, as well as the fact that we have detailed information on the AP-101, we have undertaken to examine in more depth the coverage obtainable with this machine.

Table 2.2 suggests that the coverage of the AP-101 falls between the estimated coverage of the AGC and the DCA, or between 80 and 99%. The

TABLE 2.2

BUILT-IN TEST CHARACTERISTICS OF FIVE COMPUTERS

	AGC	DCA	IBM AP-101	SKC 2000	HDC 601
MEMORY PARITY	X	X	X		
MICROPROGRAMMABLE		X	X		
MICROPROGRAM PARITY		X	X		
COMPREHENSIVE CPU CHECKS		X			
SOFTWARE ERROR CHECKS		X	X	X	
PROGRAM ACTIVITY ALARM	X	X	X	X	
INTERRUPT ACTIVITY ALARM	X	X			
LOOP ACTIVITY ALARM	X	X			
OUTPUT PARITY		X	X		
OUTPUT FEEDBACK TESTING			X		
INPUT PARITY		X	X		
INPUT TESTS	X		X		
VOLTAGE	X		X	X	X
OSCILLATOR & FREQUENCY DIVIDER ALARMS	X				
HARDWARE DEVOTED TO BITE	5-10%(E)	30%(M)			
TYPE I COVERAGE (See Section 2.2.8 for explanation)	30-40% (E)	70-80% (M)			
TYPE II COVERAGE (See Section 2.2.8 for explanation)	80-90% (E)	95-99% (E)			

(E) = Estimated

(M) = Measured

AP-101 has at least the Built-In Tests that the AGC has, with the exception of two activity alarms. This may be more than compensated by the great increase in speed, which effectively allows an error more instruction steps in which to be detected, or may allow an extensive self check to be run. In addition, memory protect bits and privileged mode instructions are included in the AP-101 software error checks, and these additional features may be quite useful.

At this point it would be appropriate to give some numerical estimate of the coverage furnished by the schemes described above. However, the uncertainty that would have to be associated with any such estimates precludes their incorporation, lest they be misinterpreted as having more validity than is warranted. Detailed examination of the computer design and a rigorous program of testing would be necessary before coverage estimates can be meaningfully made. Even when that is done, it will be no easy matter to demonstrate convincingly that a quoted coverage has been attained.

Some preliminary reliability analysis has indicated that very high detection coverages are likely to be required for the computer, perhaps in the range of 0.99 to 0.999. It is our feeling that such coverages are unlikely to be obtained with the BITE of the AP-101 itself. However, if the basic BITE is augmented with self check programs (hopefully the microprogramming capability should help here), liberal use of reasonableness tests in the software, and software implemented consistency tests involving the redundant copies of the computer, our feeling is that the error detection capabilities of the system can possibly be made acceptable. The inclusion of consistency tests in this list of BITE augmentation techniques is considered essential to provide the required error detection quality, since these tests provide the same protection as synchronization and voting for

the parameters tested. As more and more parameters are filtered through these tests, the system approaches a hardware-voted system. Nevertheless, employment of these BITE augmentation techniques will require a great deal of ingenuity, specification, and enforcement to keep the goals of coverage foremost in the minds of the designers of both hardware and software.

2.2.9 Conclusions

Although we have discussed very stringent requirements for error detection techniques, using realistic estimates of detection time requirements, available computer MTBFs, and mission success probability requirements, it is useful at this point to review how these elements interact as variables. Coverage, first of all, is a function of required detection time, which in turn is a function of the switchover time required by vehicle dynamics – generally speaking, longer switchover times allow simpler switching procedures, perhaps even by manual means, and therefore make for surer error detection. Coverage requirements can be relieved to some extent by improvements in component reliability (MTBFs) for a given success probability, which is typically set by policy. MTBFs are not subject to much control by system designers, though vendors can be prevailed upon to increase reliability (for a price) by various means such as long burn-in periods, buying high-reliability parts, or running a comprehensive high-rel program – each of these steps is rumored to double MTBF. A more tenuous way to increase MTBF figures is to predict an improvement in reliability over the program development period, extrapolating from the histories of past programs.

The conclusion to be drawn from these observations is that, although we have concentrated on Built-In Tests and detection coverage in this section, we must keep in mind their complex interrelationship with the other elements that contribute to the total probability of success.

2.3 CONSISTENCY TESTS

2.3.1 General Description

In Section 2.2, it was observed that built-in hardware and software tests may not provide adequate coverage to meet the reliability requirement on the Guidance, Navigation, and Control System (GNCS). To improve the coverage in a loosely-coupled system, the error detection and fault isolation provided by built-in tests can be augmented by opinion-of-performance testing.

There are two basic types of opinion-of-performance tests: reasonableness tests and consistency tests. A reasonableness test compares a computed data word to a predetermined range of values. This test does not require intercomputer communications and is therefore applicable to a single computer system. A consistency test compares equivalent data words from two or more redundant computers. The reasonableness test and the consistency test will succeed if the data compares to within a specified tolerance. If the tolerance is specified as zero, the test is a bit-by-bit comparison. Consistency testing has an advantage over reasonableness testing, with respect to preflight preparation, in that there is no need to predetermine nominal values for each critical variable for each step of a mission. It may also be true that the non-zero tolerance value for a given data word, necessary for consistency testing, can be determined with greater precision than the equivalent value for the reasonableness test. The remainder of this discussion deals with consistency tests for the assumed GNCS for this volume.

One goal of consistency testing is fast (approximately 0.1 second) error detection and fault isolation. This is necessary if automatic reconfiguration is required for time-critical periods of a shuttle mission. Another goal of consistency testing is the certification of the output commands which the GNCS computers send to the subsystems. The test on the output

commands must be a bit-by-bit comparison, for commands are not always whole number data words, but are quite frequently a collection of discretes for which a non-zero tolerance has no meaning. A bit-by-bit comparison cannot be done on the output of several computers unless the design insures that the outputs are equal in the absence of an error. Inequality in the absence of an error can occur because a maneuver may be executed by any of several sequences of commands, where the sequence chosen by a computer may be affected by software slivering. This can result in two computers doing the same job, each with a different sequence of output commands but both with the same end product. One example is a one-axis maneuver from a heads-up to a heads-down attitude. One computer determines that the vehicle attitude is +0.01 degrees from the vertical and commands:

"Roll +179.99 degrees"

while the second computer determines that the attitude is -0.01 degrees from the vertical and commands:

"Roll -179.99 degrees".

Another example is a two-axis maneuver from a heads-up, face-forward to a heads-right, face-down attitude. With small discrepancies as above, one computer may command:

"Pitch -89.99 degrees, Roll +90.01 degrees"

while the second commands:

"Yaw -89.99 degrees, Pitch -90.00 degrees"

both of which result in the same end attitude. In both examples, neither computer is wrong; the resulting final attitude will be correct for whichever computer is in control. A comparison of algorithms could verify the equivalence, but this is more complicated and does not guarantee bit-by-bit agreement of the commands on the data bus.

A consistency test on a given data word is executed by each of the redundant computers at essentially the same time. In general, if the data passes a test, the test procedure may compute a consensus of the data elements for possible use in further computation by all the computers. If

the test does not pass, an alarm or error condition is signalled to the Reconfiguration Control Units (RCUs) of the system and the crew, and the consensus value of the data may or may not be computed. Appropriate use of the consensus procedure will guarantee the bit-by-bit agreement of outputs of an algorithm, simply by forcing all computers to use the same inputs (including real time) to the algorithm. Referring to the example above (heads-up to a heads-down maneuver), if the consistency test is applied to the determined vehicle attitude, it would accept +0.01 degrees as being consistent with -0.01 degrees, and might pick 0.00 degrees as a consensus value for that attitude. Then both computers, using the same inputs and the same algorithm, would command a roll of +180.00 degrees. If a third computer had computed an initial attitude of (say) +5.37 degrees, it would be accused of a failure, but it too could be given the consensus value with which to continue, pending a decision to shut it down permanently.

The way in which consistency tests aid fault detection may be seen from the following discussion. Where two or more computers are performing identical computations on equivalent input data of known precision, the results of the computations will be within a predetermined tolerance. A significant fault in one of the computers will cause a result to disagree with the corresponding result of the other computers. The degree of detection obtainable by such tolerance checks approaches that of a synchronized system, with bit-by-bit voting. In fact, a consistency test is equivalent to bit-by-bit voting when the tolerance is specified as zero. This provides a means to vote on commands which the computers send to the subsystems.

2.3.2 Application to GNCS System

Execution of GNCS computer software for a generalized control loop follows this outline:

1) Subsystem sensors provide inputs to
2) the vehicle state determination/update routines that generate
3) the state variables that input to

4) the guidance/targeting routines that output
5) the delta variables (to change the state of the system) that are inputs to
6) the command generation routines that output
7) the control commands to the actuators.

Subsystems can be protected from erroneous control commands by comparing the subsystem commands of two or more computers that are all doing the same job. If all computers use identical inputs to identical command generation routines, where time is an input if needed, a consistency test with a zero tolerance may be applied to the output of the routines. If the computers are fault free, the outputs agree bit-by-bit and the test will pass successfully. If a computer has a fault that affects the output, the test will detect the bit discrepancy and signal an alarm so that corrective action may be initiated via the RCU and the good computers, while the correct output (if there are three or more computers) goes to the actuators.

This discussion generally assumes that there are three active rails (one prime and two backups). The inputs to the command generation routines are the appropriate outputs (redundantly produced) of the guidance routine, after they have been processed by the consistency test and a consensus generator. The test involves each computer comparing a data word from its guidance routine with equivalent data words from the other two computers. Differences are computed and compared to a predetermined tolerance for the given data word. An output of the consensus generator, if at least two data words are good, could be, for example, the arithmetic mean of the good data words. This mean value is used as the input to the copies of the command generation routine.

There are two possible responses of the consistency test to a detected failure in computers of the GNCS. One response occurs when agreement is found between only two of three active rails (one prime, two backups). This response alarms and identifies the disagreeing computer to the crew

and the RCUs. If the problem is with the prime computer, that computer is inhibited from putting out commands. Primacy is passed on to one of the other computers by simply allowing it to output commands.

The other response of a consistency test to a detected failure occurs when none of the rails agree (for either two or three active rails.) This response alarms, but does not isolate the failure to a rail since there is insufficient information to make this judgement. One situation in which this response can occur is when the system had been degraded to two rails. The consistency test continues to work after a failure had caused the system to degrade from three to two rails (one prime, one backup). With only two rails, if the test inputs do not agree within the specified tolerance, the consistency test response is to trigger an alarm to the crew and to the RCUs. In the absence of an error signal from BITs both computers could be inhibited from outputting commands on the bus until the crew can indicate a corrective procedure. An optional procedure could permit the prime to continue while the crew may override and change primacy if the results prove to be unsatisfactory. With this procedure, there is at least a 50 percent chance of being right. This procedure is preferable whenever it may be more dangerous to do nothing than to send the wrong command. With an alarm from the BITs, the faulty computer can be isolated automatically.

The worst case is when no agreement is found between three active rails by the consistency tests of all three rails. The response for this case is identical to that for the degraded mode case of two active rails, that is, signal the crew and inhibit the computers from transmitting commands on the bus. This case, which is not generally being considered in the system design, amounts to simultaneous failure of at least two rails. A specific subset of this worst case that is being considered is an event that can cause a massive noise pulse that has a general destructive effect on data in all rails (e.g., a lightning strike). In this case, response of the consistency test can serve to minimize the number of bad commands received

by the subsystem effectors by providing the signal that causes the RCUs to inhibit all the computers from transmitting commands on the guidance data bus.

2.3.3 Test Method

The first step of a consistency test is the collection of the data to be tested. For example, some routine produces a result that is tagged for consistency testing. Before passing it on to the next routine, it must pause, transmit the result to the other computers, and wait for the other computers to transmit the corresponding data back to this computer. The transfer of this data assumes the existence of a fast intercommunication channel among the computers (see Section 1.1). The data collection process may involve an interrupt or may operate by polling an intercomputer data buffer. Once the data is collected, the appropriate value of tolerance is loaded, along with the data, into the input registers of the consistency test routine.

Figure 2.3.1 is a flow chart that illustrates an example of a consistency test. The three input variables to be compared are A, B and C and the value of tolerance is D. The difference between A and B is BA; B and C is CB; C and A is AC. The variable A is the result of the computation in the host computer, that is, the computer executing the test illustrated. The variables B and C are the equivalent results from the other two computers. These two computers will run the same test with the same variables appropriately permuted. The routine provides entry points for two or three rails in the active mode. The test compares the magnitude of each difference to the tolerance. The rules of the test algorithm for three input variables are:

(1) If two of three differences are within tolerance, use the average of the three variables as the consensus output of the test for use in further computation, do not alarm.

(2) If only one difference is within tolerance, use the average of the two producing that difference as the consensus output and

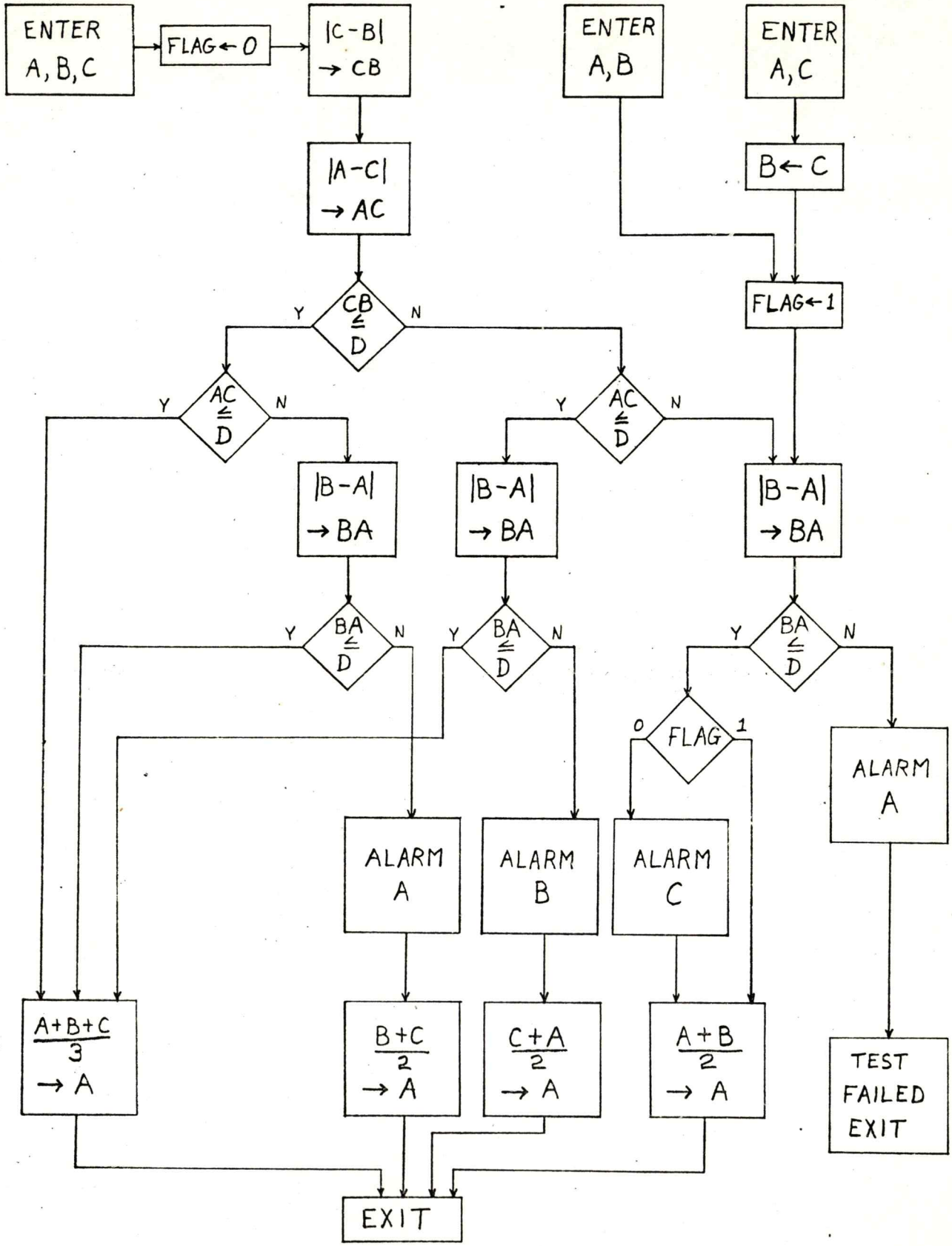


Figure 2.3.1 Consistency Test Performed by Computer A

alarm, indicating the computer that provided the third variable as the one being in error.

(3) If there are no differences that are within tolerance, alarm indicating that computer A has gotten an unreasonable result, and terminate the test by way of the "Test Failed Exit". A consensus output is not provided.

The consistency test routine stores the consensus output, if provided, into the register A. If a consensus output is not provided, the original input variable in register A is left unchanged.

The two-input case is also represented on the flow chart. The inputs are A and B or A and C. Only rules (2) and (3) apply, with the modification that reference to the third computer is meaningless. For single string operation, the computer bypasses the consistency test.

2.3.4 Test Implementation

Except for the mechanism for the intercommunication channel and a time-out trap, there are no special requirements for implementation of the consistency test. The time-out trap is required because the redundant computers are not running in tight synchronization. As a result, even though the computers start together and execute identical routines, the first computer to reach a consistency test will have to wait for the data from the other computers before it can proceed. If after a specified time, a computer fails to deliver the appropriate data, the time-out trap alarms and identifies the computer that has failed to report. The consistency test can be implemented by software, by firmware, or most likely by a combination of both, if microprogram is available. The microprogram capability is desirable to minimize the execution time of the consistency test. The test could be implemented by external hardware, but this is not cost effective since such a test is complex enough to be considered a hardware development risk.

Good

The test illustrated in Figure 2.3.1, if implemented entirely by software, will have a nominal execution time of 40 microseconds if all three variables are good (assume two microsecond add time). For any other case that passes the test (stores a consensus in register A) the maximum execution time will be approximately 50 microseconds. These times assume that the data has been collected and stored in the consistency test input registers. The time delay due to the data collection is the phase difference in the execution of instructions between the fastest and slowest computers and the time required to transfer data between computers. With all the computers starting together, and their individual oscillators having drift rates of less than one part per million, the difference in time will be less than one microsecond for one second of running time (i.e., one half of an add time). The data transfer time is about ten microseconds so that the total data collection time is no more than 15 microseconds. The collection of data for the execution of the consistency test tends to phase lock the computers. Assuming that the period of consistency tests is on the order of one-tenth of a second (or less), the software phase difference at the time of any consistency test should be, on the average, no more than a few microseconds (about ten percent of the time required to execute the test). The sum of the test execution time and the test data collection time results in a time of about 65 microseconds. If the consistency test is performed once every ten milliseconds, the resultant increase in software overhead will be less than one percent. This overhead can be reduced by partially or fully encoding the consistency test in microcode. Full microcoding of the test would result in a reduction of about 15 to 20 microseconds from the total test time of 65 microseconds. This savings is cost effective if time is critical and there is room in the microprogram memory.

2.3.5 Summary and Conclusions

The purpose of the consistency test, with the consensus generator, is to reinforce BITs in the detection and isolation of faults within computers of a multicomputer complex. The procedure is usable as a programmer's

option. It may be used selectively and does not have to be used for every computational result or every command transmitted on the data bus. The consistency test gives maximum coverage where it can be applied to commands from the computers to the subsystem actuators. A two level consistency test provides this feature.

The first level is executed with non-zero tolerances at some intermediate point in the software between inputs to the computer and the outputs from the control routine. Outputs from these first level tests provide equal input data for the balance of the computation that produce the control outputs. This provision insures that the command generation routines will produce equal outputs. The command generation routines, without equal inputs, will not necessarily produce equal outputs even when there is no error.

The second level test is applied to the control outputs with a zero tolerance requirement. The choice of the point for the first level of consistency test is flexible, it may vary from one software package to the next as circumstances require. It may even be desirable to locate the first level test at several points, for example, at the input to the computers and at the guidance-output/control-input software interface.

The consistency test is fast enough to provide error detection in time-critical periods. The impact of the test on software overhead is on the order of one percent for a representative computer (two microsecond add). The most significant design risk is not with the test itself, but with a fast intercommunication channel that is necessary for timely and rapid execution of the consistency test. Where microprogram is available, the test time and overhead can be reduced, though optimization of the test may not require full microcoding of the test. It appears more likely that the test will be a mix of software and several special purpose microcoded instructions.

2.4 RECONFIGURATION CONTROL

2.4.1 General Description

This section describes reconfiguration control in the Guidance, Navigation and Control System (GNCS) portion of the assumed digital computation and distribution system. The general objective of reconfiguration control is redundancy management of the GNCS. The primary goal of reconfiguration control is fast automatic reconfiguration of the GNCS in the presence of an error in the prime rail, especially during time critical periods of a shuttle mission. (Switching time, having detected an error, is negligible compared to the allowed detection time of 0.1 second.) A time critical period is one in which the crew might not be able to respond to a detected error fast enough to prevent a hazardous response of the system to a bad command from the prime rail. Another goal is to provide the crew with indication of failed backup rails, to help them decide whether to abort, and to prevent primacy from being assigned to a bad rail. A third goal is to provide the crew with a manual override capability for the GNCS configuration. This allows for single string operation to conserve power during non-time-critical periods.

Figure 1.1.1 shows the GNCS computer complex, consisting of three loosely coupled rails with interfaces to a Reconfiguration Control Panel, an inter-regional data bus and an intercommunication channel among the computers. Each rail (Figure 2.4.1) consists of a computer capable of performing all GNCS processing, a Reconfiguration Control Unit (RCU) to perform redundancy management of the three rails, a GNCS I/O controller and a data bus to the subsystems. In many places in this section, "computer" is written informally instead of "rail", because although reconfiguration involves switching the whole rail, the emphasis in this report is on the computers.

A fault tolerant computer complex consists of two or more computers, arranged to be functionally equivalent to one ultra-reliable computer of

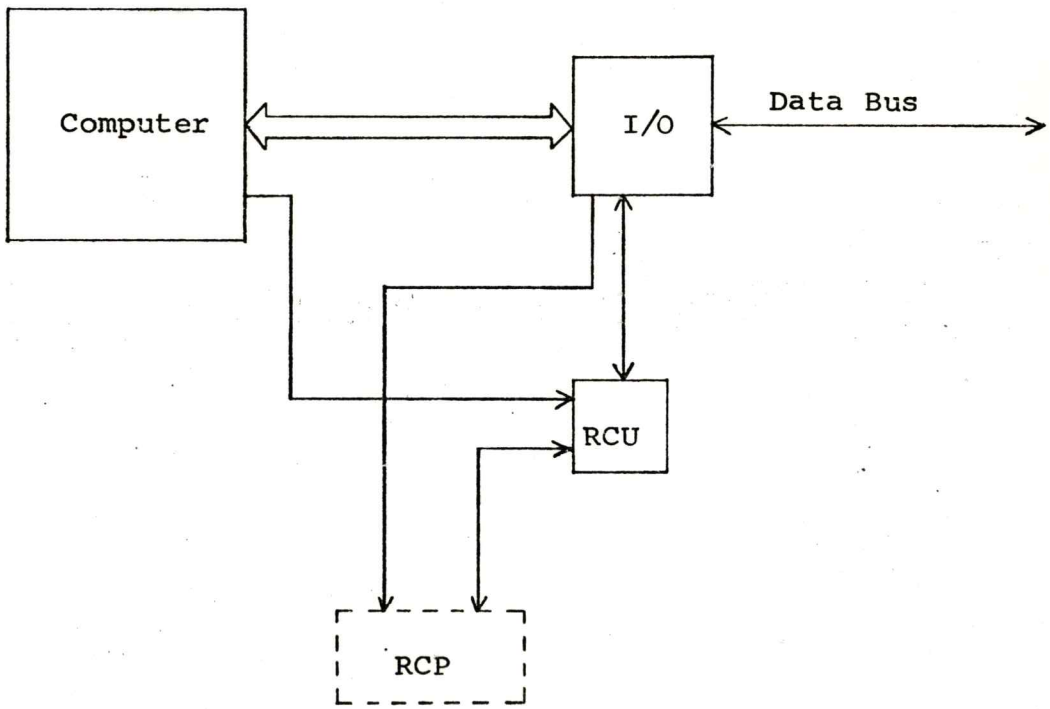


Figure 2.4.1 One Rail of the GNCS

the same performance characteristics. The GNCS computer complex achieves this equivalence, under the ground rule of loose coupling, by making one computer Prime until it fails, at which time another computer is made Prime. The Prime computer is the only one that can issue commands to control the GNCS; the others may have the status Backup Not-Ready (unpowered), Backup Not-Ready (powered), or Backup Ready (which implies having a complete and up-to-date conception of the state of the GNCS for minimum transition time at takeover). In fact, the computers are ignorant of which computer is prime and which are backup ready. This knowledge is contained in the RCU and I/O controller. The backup ready computers execute the same routines as the prime and even try to transmit control commands to the subsystems. The I/O controllers prevent this from happening, but the computers do not know this. Loose coupling implies that no bit-by-bit hardware voting can be used for error detection or fault isolation, and therefore that the Prime computer must detect its own errors and consequently give up control to another computer. This rule places a heavy responsibility on the Built-In Tests (BITs) provided with each computer. However, a three-rail configuration can perform active surveillance with "software voting" if the two non-Prime computers are in Backup Ready status, thus augmenting the coverage of BITs. This idea of software voting is discussed in further detail in the section on consistency tests (see Section 2.3).

Reconfiguration control may be accomplished either by a centralized RCU that is responsible for the three rails, or by an RCU dedicated to each rail. The design of the centralized RCU can be visualized as either one RCU having overlapping responsibilities or subdivided into three parts or modules, each module dedicated to one rail. Fault tolerance of the centralized RCU would be achieved by replication of the whole RCU for the first case, or of each module in the second case. The main concern here is that a single point failure must not cause the RCU to shut down more than one rail, or allow two to be prime. For the case of the modular centralized RCU, the design, including replication, is virtually identical to

that of three RCUs, one per rail. The only difference is topological; that is, there is no functional difference whether the replicated modules reside in a central location, or are distributed throughout the system. For purposes of this study, the one-RCU-per-rail concept achieves greater visibility into how the failures in the reconfiguration control area are related to failures of other units.

The Reconfiguration Control Units (RCUs) located in the GNCS computer complex of this system are dedicated to their respective computers and associated I/O controllers as represented in Figure 2.4.1. Inter-RCU signal paths are over dedicated wires routed by the Reconfiguration Control Panel (RCP). The RCP provides the flight crew with a manual reconfiguration override capability and consists of a display and control switch panel. If two computers are in a Backup Ready status, a priority rule, set manually at the RCP, is used to pass control automatically in the event that the prime computer fails. The rule is set by controls that are functionally equivalent to routing the reconfiguration signal by way of a patchboard.

A general concept applicable to the problem of a backup checking the health of the prime is that an automatic response to an error message indicating that the backup thinks the prime is erroneous must be inhibited unless the message is independently confirmed by a third computer. The corrective action is initiated by the RCU of the offending computer. If the error message is not confirmed (suggesting that the plaintiff is erroneous), the RCP display will indicate this discrepancy. The crew incorporates this information with any other information that is available and takes whatever action is appropriate. In the absence of other information, that action might be to do nothing except to change the priority rule used to pass primacy between rails, but this depends on who is prime.

2.4.2 Function of the RCU

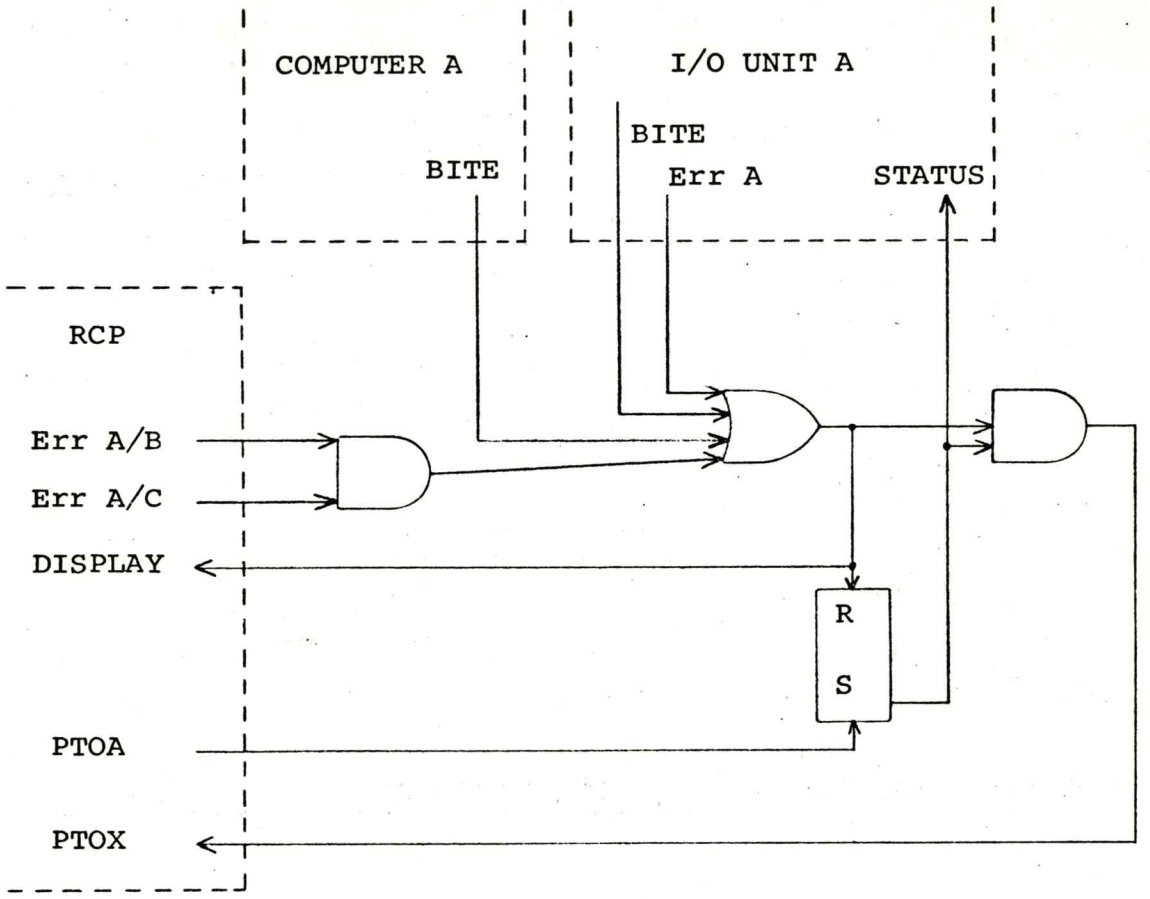
The RCUs perform essentially the same functions as their counterparts in the Task 28-S Unsynchronized Federated system (UF).[2] The primary

function here is to pass Prime status from a faulty computer (actually a rail) to another computer, which may or may not be ready to effect a smooth takeover. Secondary functions are the indication of a failed computer and the reconfiguration to single-string operation to conserve power. The most significant difference between the two systems is that each RCU of the UF system had to service four computers, each doing a different job, while the RCU of this system services only one computer, doing the GNCS job. Another difference is that the RCU of this system has a lesser role in the reinstatement of a computer that has had a transient failure.

The RCU is envisioned as a simple collector of error signals that issues a please-take-over signal (PTO) when an error is detected in the prime rail. The inter-RCU traffic consists of PTO signals and rail status signals over the Reconfiguration Control Cable, a dedicated-wire signal path. The PTO signal is preferred to an I'm-taking-over signal from a backup rail. If the prime rail issues a PTO signal erroneously, primacy is passed to a good backup (the same response as for any other failure in the prime rail); but if I'm-taking-over signals were used, the potential would exist for a bad backup to issue a takeover signal erroneously, thereby assuming system primacy.

2.4.3 Operation of the RCU

Figure 2.4.2 is a simplified logic diagram of an RCU. The diagram illustrates functional relationships between various signals but does not show detail timing or replication for fault tolerance. An RCU can receive three different types of error signals. The first type indicates a failure of its computer and originates from the computer's hardware built-in-tests (BITE) or software checks (i.e., self-check, reasonableness tests, and consistency test routines). The second type indicates a failure of its computer but is generated by the consistency tests of the other computers. The third error signal type originates in the BITE in the I/O unit, indicating that the I/O unit, the RCU, or the data bus has failed.



INPUT SIGNALS

BITE: A is in error

Err A: A is in error

Err A/B: A is in error (from rail B)

Err A/C: A is in error (from rail C)

PTOA: A, please take over

OUTPUT SIGNALS

PTOX: A has failed, B or C, please take over

DISPLAY: A has failed

STATUS: A is Prime

Figure 2.4.2. Logic Diagram of the RCU for Rail A.

In the event that one computer decides that another computer is in error, the error signal (type 2) is transmitted by way of the rail status signal paths. The PTO signal is sent by the prime computer's RCU to the appropriate backup when the prime has had a hard or transient failure. The RCU/RCP interface includes signals to and from the crew. Information displayed to the crew includes the results of consistency tests so that in the event that one backup is inactive and the other disagrees with the prime computer, the crew can cast the deciding vote as to which rail is good. Control signals from the crew to the RCUs alsc provide the means for effecting single string operation.

An error detected by BITs, either in the computer or I/O unit, is sensed by the RCU, which sends a signal to the crew and, if the computer is prime, issues a please-take-over signal to one of the backup computers. In addition, the PTO signal causes the current prime computer to change status to Backup Ready. This is accomplished by simply not allowing this computer to output control commands on the data bus. In addition, the crew can disconnect this computer from the priority chain for system control, thereby making it Backup Not-Ready, so that control cannot be returned before reinstatement of this rail has been accomplished.

An RCU reacts to errors detected by consistency tests in two ways. If a computer determines that it, rather than one of the other computers, is faulty, the response of the RCU is identical to that for an error detected by BITs: signal the crew and if prime, switch primacy to a backup computer. If a faulty computer does not discover that it is faulty, but this conclusion is reached by consistency tests in two backup computers, the RCU responds as if the faulty one had discovered an error. The latter only occurs if the opinions of the backup computers are in agreement; no change in primacy takes place if only one of the backups thinks the prime computer has failed.

2.4.4 Reinstatement

Once a computer has failed, its RCU switches it to the Backup Ready state. At this point the crew may use the facilities of the RCP to set the

state to Not-Ready and might even power down the failed unit. If the power is left on, an attempt can be made to reinstate this computer. The various possibilites to start and execute a reinstatement procedure run a full spectrum from manual through semiautomatic to fully automatic. This volume does not consider the means to start and execute the procedure but the feeling is that manual initiation is adequate and cost effective.

A reinstatement procedure would start with the execution of the computer self-check program. If this passes, some type of initialization program (e.g. Fresh Start) would then be run. This computer proceeds by requesting, on the intercommunications channel, the GNCS system state information from the other computers. After reconstructing its record of the state of the system, the computer continues by performing its own system state update computations with data fetched from the subsystems. After some number of iterations of the system update computations, this computer compares (via the consistency test) the state information received from the other computers to its own computed data. After at least one complete cycle of data from the system without an error indication, the status of the complex is manually changed from Backup Not-Ready to Backup Ready, providing that no further error signals have been received from the RCU. While the reinstatement process is being executed, the other rails do not use data from this rail for consistency tests.

2.4.5 Fault Tolerance within an RCU

The design of an RCU must consider the fault tolerance requirement for that RCU. The fault tolerance requirement for an RCU of the Unsynchronized Federated system is that it must endure one single point failure without degrading functional capability, and be able to switch rails after detecting a second single point failure within itself. The requirement for a distributed RCU of this system is that it must be able to switch rails after experiencing a single point failure within itself.

There are some unanswered questions in this area. For instance, it is possible to minimize the ways in which RCU failures are indistinguishable

from computer failures, by cross-strapping a redundant RCU complex to the computers, thus making it satisfy a fault tolerance criterion independent of the one applied to the computers. Furthermore, one might argue that the RCU complex should be required to survive more failures than the rest of the computer system because of its central importance and authority over the other elements; in fact, the preceding paragraph defines the minimum increased requirement, which is equivalent to perfect error detection coverage. The degree of "redundancy overkill" that is appropriate will be determined by a number of factors: the small size and simplicity of function of an RCU, the reliability program followed in its development, the interface problems encountered in mating a specially developed RCU with off-the-shelf computers, etc. Perhaps these questions are best left open for the time being, but must be addressed before beginning the detailed design of the RCU.

APPENDIX A

A DATA BUS INTER-COMPUTER COMMUNICATION SCHEME

A.1. Introduction

An inter-computer communication scheme is described for a loosely coupled multi-region system in which all the computers of all the regions are connected to a single data bus. The configuration discussed here is not intended to represent any avionics baseline being considered for the shuttle, except insofar as the partitioning into regions is similar to previous studies. This intercommunication scheme was developed before the current configuration, and is included here for background. The intent is to demonstrate the feasibility of intercommunication among a large number of computers, thereby establishing the feasibility of intercommunication on a smaller scale. In particular, it is expected that a subset of the scheme described here could be used for the IRB of the DCDS assumed in this volume. The difference between this configuration and the assumed DCDS is that in this appendix, the members of a region communicate with each other over the bus used for inter-regional intercommunications. In the assumed DCDS (Section 1.1.1), the IRB is not used for communications among computers of the same region.

The system, shown in Figure A.1, consists of up to four computational regions, all of which may be triplex. Bus access is allocated by an external Bus Controller. Since computers receiving a data transfer will generally not be ready to receive when the sending computer is ready to transmit the data, the Bus Controller also performs a "store and forward" function, buffering the data being transferred until the receiving computers are ready. The Bus Controller and data bus are triplicated for fault tolerance. Computers are partially cross-strapped to the bus system; each computer is connected to two of the buses, rather than to just one or all three.

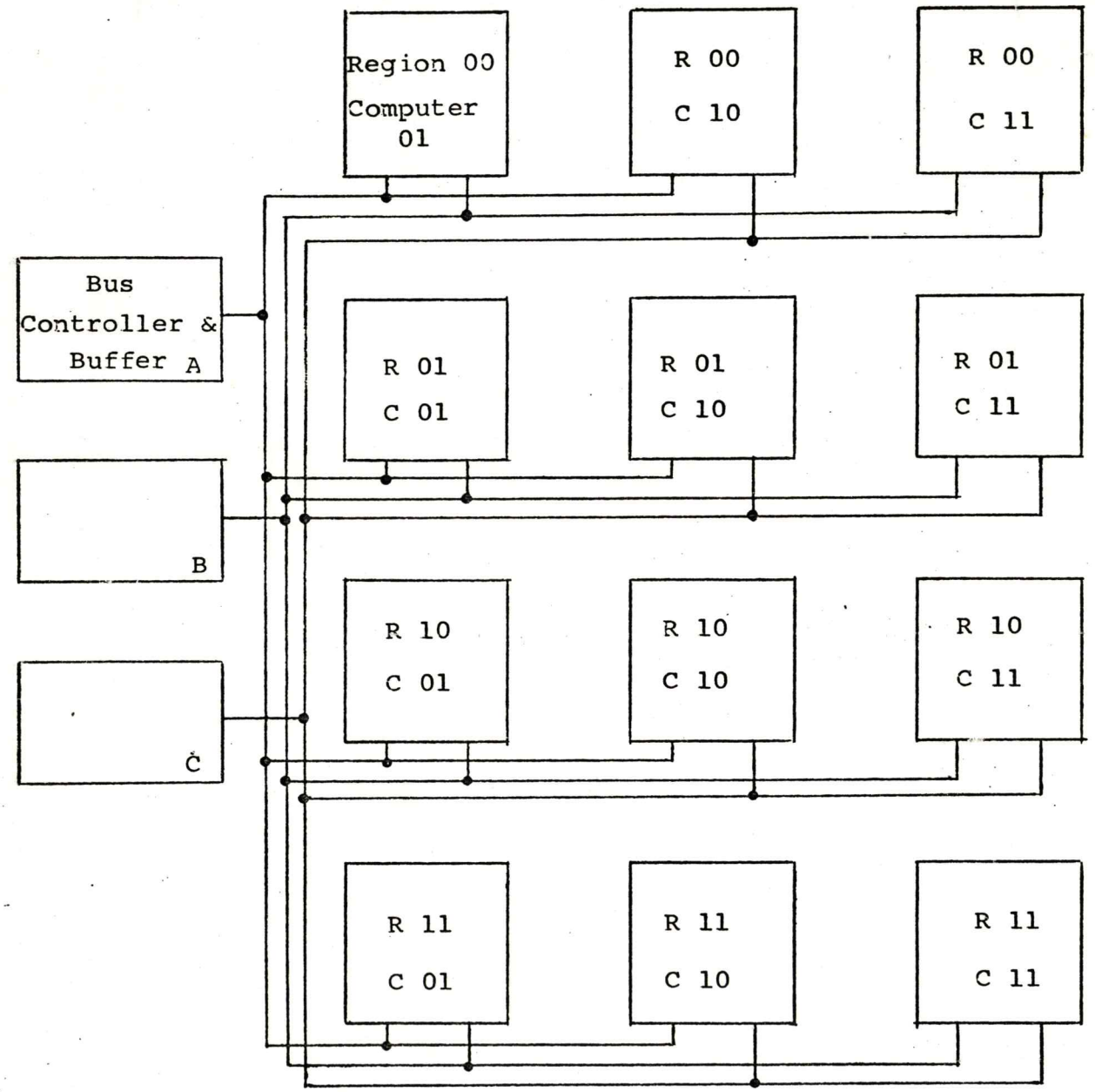


Figure A.1. A Generalized Loosely-Coupled Multi-Region System

A.2. System Operation

An information transfer involves some number of conversations between the Bus Controller and the computers sending and receiving the information. There are four types of conversation, as tabulated in Figure A.2; an information transfer requires at least one conversation of each type. Every conversation begins with a message from the Bus Controller to a computer, and the format of the first byte of this message indicates the type of conversation (see Figure A.3). Most conversations also require a reply message from the computer back to the Bus Controller.

All messages contain an integral number of bytes, a byte consisting of 8 information bits plus parity. It is assumed here that the bus system is serial, so two parity bits, one even and one odd, are used with each byte. A data message is a variable-length block whose size is specified by the computer originating the transfer. One byte is used to specify the block size, so the limit for a particular transfer is 256 bytes.

Manchester coding[9] is used to represent data. Messages from the Bus Controller to a computer are prefaced by a SYNC character. For the conversations which require a reply message, the reply is not prefaced with a SYNC. However, the first bit of the reply is always a zero so the receiver in the Bus Controller can lock itself to the data.

Data being transferred is sent to all the computers of the destination region, so the same mechanism is used for transfers among computers of a particular region as for inter-regional transfers. In all cases the information is first sent to the Bus Controller, which stores it in a 256 byte memory. The Bus Controller then asks each of the destination computers if it is ready to receive the data (an interrupt may be needed here to get the computer's attention), and forwards the information when they are all known to be ready.

TYPE	BUS SEQUENCER TO COMPUTER	COMPUTER TO BUS SEQUENCER
POLL	SYNC, Do you have anything to send?	NO/YES, data
READY	SYNC, Are you ready to receive?	NO/YES
DATA	SYNC, Here comes the data, Data	
ACK	SYNC, Did you receive it OK?	NO/YES

Figure A.2. Conversation Types.

TYPE	FIRST BYTE (BUS SEQUENCER TO COMPUTER)
POLL	xxyy_1000
READY	xxxx_xxyy
DATA	xxxx_1100
ACK	xxyy_0000

x: 0 or 1.

y: 0 or 1, but yy $\neq$ 00.

SYNC and parity not shown.

Underscore (_) is used for clarity only.

Figure A.3. How Conversation Types are Identified.

1. POLL

Query		Srrqq_1000P$\bar{P}$
Reply	NO	0011_rrqqP$\bar{P}$
	YES	0100_dd00P$\bar{P}$_hhhh_ℓℓℓℓP$\bar{P}$, followed by the information, (hhhhℓℓℓℓ+1) bytes of the form iiii_iiiiP$\bar{P}$.

2. READY

Query		Shhhh_ddccP$\bar{P}$
Reply	NO	0011_ddccP$\bar{P}$
	YES	0100_ddccP$\bar{P}$

3. DATA

Sℓℓℓℓ_1100P$\bar{P}$, followed by the information, (hhhhℓℓℓℓ+1) bytes of the form iiii_iiiiP$\bar{P}$.

4. ACK

Query		Sddcc_0000P$\bar{P}$
Reply	NO	0011_ddccP$\bar{P}$
	YES	0100_ddccP$\bar{P}$

cc	Computer in destination region being addressed (cc ≠ 00).
dd	Destination region.
hhhh	High four bits of the byte count. (hhhh+1) is the number of 64-byte blocks which must be reserved for the data.
hhhhℓℓℓℓ	(Count of information bytes)-1.
iiii_iiii	Information byte.
ℓℓℓℓ	Low four bits of the byte count.
P$\bar{P}$	Parity bit and its complement.
qq	Computer in region being polled.
rr	Region being polled.
S	Sync character.
	Underscore (_) does not represent a bus character. It is used for clarity only.

Figure A.4 Bus Formats.

Since the computers are partially cross-strapped to the buses, a Bus Controller to computer message must be sent on at least two buses; otherwise, one computer in each region would be guaranteed not to receive the message. It is not necessary to synchronize the buses, but the Bus Controllers must be coordinated in some manner so that one doesn't get so far out of phase that it is doing an operation completely different from the others. It would seem desirable to transmit on all three buses to minimize the need to retransmit a message if there is a failure. Similarly, it would be sufficient to transmit a computer to Bus Controller message on only one of the buses connected to the computer but it seems better to send the message on both. On the other hand, the best approach may be to fully cross-strap the computers to the buses. Then the Bus Controller/data bus units could be used sequentially; that is, one Bus Controller and bus would be used until there is a failure, then another unit would be switched in. Parity and time-out traps should provide good enough error detection capability to allow this strategy.

A.2.1. Details of Operation

The exact message formats are listed in Figure A.4. The Bus Controller polls the computers sequentially. Polling is not expected to require an interrupt; a computer with something to transmit should be able to set a bit in its I/O unit causing it to reply "YES" the next time it is polled. The response to a poll, "YES" or "NO," indicates whether or not the computer has data to send to another region, or to the other computers in its own region. If the response is "NO" (one byte), the computer being polled does not wish to transmit data, so the Bus Controller polls the next computer. If the response is unintelligible, "NO" is assumed; it should be noted that when "YES" is intended an unintelligible response could be up to 258 bytes long.

A "YES" response (one byte which includes the ID of the destination region) is immediately followed by a byte which indicates the number of

bytes of data, followed by the data. The data is stored in the Bus Controller Memory until it can be forwarded to the destination region. If the Bus Controller detects a parity or count error in this message, it asks for a retransmission by immediately repolling the computer. Once the information being transferred is stored in the Bus Controller Memory, the Bus Controller suspends polling and completes the transfer. Each computer in the destination region is asked individually if it is ready to receive the transmission (except that if the transmission is to the other computers of the same region, the computer which originated the transmission is not asked). This operation might involve interrupting the computers. A computer which replies "NO" (or replies unintelligibly) is asked again after an interval, and this is repeated until it replies "YES" or the Bus Controller gives up. (The number of tries before giving up has not been determined). When all the computers in the destination region have replied "YES," or at least have been given a decent opportunity, the Bus Sequencer completes the information transfer by placing the contents of its memory on the bus while the destination computers "listen." The actual mechanism for getting the data into the computers is probably a DMA facility, although it is conceivable that each computer would be required to continually read its I/O channel while the data is being transmitted. In any event, interrupts would not be used for this operation.

After the message has been transmitted to the destination computers, the Bus Sequencer asks each recipient for an acknowledgement. The reply is again either "YES" or "NO." If any computer replies "NO," indicating that it has detected an error, the whole message will be retransmitted. Once all recipient computers have acknowledged receipt of the message, or the message has been retransmitted, normal sequential polling is resumed. It is expected that it is not necessary to ask for acknowledgements after a retransmission. If a computer replies unintelligibly to an acknowledgement query it is asked again; if the reply is still unintelligible, it is assumed that there has been a failure, and the message is not retransmitted unless one of the other computers replies "NO."

A.2.2. Events in a Normal Data Transfer

A normal data transfer requires at least one of each of the four types of conversation shown in Figure A.2. The first message of a conversation originates with the Bus Controller. A SYNC character is used to indicate that a Bus Controller to computer message follows; the format of the first byte of this message identifies the conversation type, as illustrated in Figure A.3.

The operation of the Bus Controller and the computers involved in a normal data transfer is summarized below. Operations are divided by conversation types, which occur in the order given. The formats of Figure A.4 are referred to.

1. POLL. A polling query asks the question, "Do you wish to send data to another computer?" It is a one-byte message of which four bits (rrqq) identify the computer being polled, and the other four (1000) indicate the conversation type. A polling query is sent on enough buses that all computers hear it, but only the addressed computer replies. The reply may be "NO," in which case the next computer is polled, or "YES," which leads to the sequence of events described in the remainder of this section. A "YES" reply may be three to 258 bytes long. Two bits (dd) of the first byte indicate the region to which the data bytes are to be forwarded, and four bits distinguish "YES" (0100) from "NO" (0011). The second byte indicates the number (one to 256) of data bytes which follow, and the remainder are data.
2. READY. A ready query asks, "Are you prepared to receive data being sent to you?" It is sent individually to the computers in the destination region, which reply "YES" or "NO." Four bits (ddcc) of a ready query identify the computer being asked as well as the type of conversation (since cc $\neq$ 00). the other four bits (hhhh) are the high-order portion of the byte count, which

is used by the computer to allocate space for the data; these bits indicate the number of 16-byte blocks to be allocated. If a computer responds "NO" it will be asked again after an interval. A "NO" response might always occur initially if the computer must be interrupted; in this case the "YES" reply would not be sent until the interrupt has been processed, and the requested number of blocks allocated.

3. DATA. Once all the destination computers have replied "YES" to a ready query, the Bus Controller uses a data message to forward the data. The first byte consists of the low four bits (llll) of the byte count as well as the designation (1100) of the conversation type. The first byte is followed by one to 256 bytes of data, as specified by the count. There is no reply to a data message.
4. ACK. Acknowledgement queries are used to verify that the data message was received coherently by the destination computers. The Bus Controller sends each destination computer a one-byte message, asking whether the data passed the parity and byte count checks, to which the computer responds "YES" or "NO." Four bits (ddcc) of the acknowledgement query specify the computer being addressed, and the other four (0000) indicate the conversation type.

REFERENCES FOR PART I

1. Blair-Smith, H., et al., AGC/Shuttle Study Status Report, MIT/CSDL, December 1970.

2. Blair-Smith, H., et al., Data Management System Configuration Studies for Off-the-Shelf Computers ("Task 28-S") Vol. I, MIT/CSDL, December 1971.

3. Blair-Smith, H., et al., Shuttle Avionics Computer System Studies, ("Task 28-S") Vol. III, MIT/CSDL, June 1972.

4. Green, A.I., et al., STS Data Management System Design (Task 2), MIT/CSDL Report E-2529, June 1970.

5. Griggs, K.M., and Schwartz, G., The DCA Computer, MIT/CSDL Report E-2590, December 1970.

6. Knox, R., et al., Ad Hoc Review Team Technical Observations (Knox Committee Report), NR, October 1972.

7. Levy, C.D., A Description of the Data Bus System for Integrated Breadboards (Revision A), MSC/ISD Internal Note, January 1971.

8. Multi-Region Shuttle DMS-Interim Report, Digital Development Memo #603, MIT/CSDL, April 1971.

9. Schwartz, G., The Mechanics of Bi-Phase Coding on a Data Bus, Digital Development Memo #696, MIT/CSDL, September 1972.

10. Tindall, H.W., "Some Really Outstanding Thoughts about the Performance Monitor System (PMS)", Memo, 9 December 1971.

REFERENCES FOR PART I (con.)

11. Weinstein, W.W., An Efficient Intercommunications Scheme for the Elements of a Real-Time Data Management System, MIT/CSDL Report E-2588, June 1971.

12. Johnston, M., Contrasts between Several Shuttle DMS Architectural Concepts, ("Task 28-S") Vol. II Addendum, MIT/CSDL, December 1971.
(also available as Digital Development Memo #715)

A Division of
Massachusetts
Institute
of Technology

The Charles Stark Draper Laboratory

68 Albany Street, Cambridge, Massachusetts 02139 Telephone (617) 258-

PART II

LIGHTNING PROTECTION STUDY

Eldon C. Hall

James G. Allen

William W. Weinstein

TABLE OF CONTENTS

1.0 INTRODUCTION AND CONCLUSIONS

1.1 Introduction

This report addresses the problem of lightning disturbances in the Space Shuttle avionics system, with particular emphasis on the computer system. This type of disturbance is of interest since it can cause massive transient failures which are common to all the redundant copies (whether similar or dissimilar) of the Shuttle avionics, thus defeating the design intent of the redundancy. The EMP component which results from a line of sight burst is also discussed as its effects are similar to those associated with lightning strikes.

The material is divided into three main sections. First, documented lightning strikes are reviewed to categorize the nature of the lightning and its effects on electronics in avionics systems. Second, techniques such as shielding and grounding for preventing the lightning from disturbing the electronics are discussed. Third, methods are developed for computer system recovery after lightning induced errors.

1.2 Conclusions

This section summarizes the major conclusions of the body of this report. The conclusions section itself is divided into the following topics:

Lightning Properties and Effects
Shielding and Grounding
Testing
Design
Architectural Considerations.

1.2.1 Lightning Properties and Effects

1. Worldwide Air Force lightning strike data gathered from 1965 through 1971 reported over 400 lightning incidents. This information shows lightning to be a very real atmospheric flight hazard. However, it does not provide detailed information on lightning characteristics and its effects on aircraft electronics. Of the material reviewed, the Apollo 12 lightning strike incident proved to be the most thoroughly documented.

2. Lightning can be expected to induce both hard and transient errors in unprotected electronic equipment.

3. Lightning can be expected to induce transient power loss, as in the Apollo 12 incident.

4. Peak currents of 100,000 amperes with rise times of 50,000 amps/microsecond are possible under the right conditions. The discharge may occur in as little as a few microseconds or last as long as hundreds of milliseconds.

5. EMP is characterized by a pulse whose width is measured in tens of nanoseconds with significant field strengths. The effects are similar to lightning although the incident radiation would be spatially distributed over the entire Shuttle and the incident energy bandwidth is three orders of magnitude wider.

6. Apollo accepts lightning strikes as a real risk and simply avoids launching into atmospheric conditions conducive to lightning strikes. However, thunderstorm activity data indicates the restrictiveness of such a policy. The Eastern Test Range experiences an average of 70 days with thunderstorms per year. In comparison, Vandenberg and Edwards Air Force Bases, respectively, experience averages of 2 and 4.3 days with thunderstorms per year.

1.2.2 Shielding and Grounding

1. Prevention of lightning-caused permanent damage to the computer hardware, through the use of proper shielding and grounding practices, is feasible and must be incorporated in the design.

2. Normal design practice based upon MIL-STD-461A is not sufficient for a lightning or EMP operational environment.

3. The shielding necessary for absolute information protection for all orbiter electronic systems is probably not feasible due to Shuttle weight constraints.

4. Accepting transmission errors between modules, but shielding and grounding the modules and incorporating the appropriate software recovery capability, seems feasible and cost effective.

5. No evidence can be cited that the vehicle structure must not be used for power return. However, use of dedicated twisted pair power cabling is recommended, even in recognition of a weight penalty, to reduce the risk associated with lightning related failure modes.

6. The design for protection from lightning is <u>not</u> sufficient for EMP, unless the range reduces the magnitude of EMP significantly. Absolute numbers specifying this range must

be worked out. The Air Force survivability specification does not provide sufficient information to say that lightning is equivalent to EMP at or beyond the specification range.

1.2.3 Testing

1. Testing defined in MIL-STD-461A is not sufficient for evaluating a system designed to operate in a lightning or EMP environment. Techniques developed and applied to hardened missile systems provide a basis for developing a proper set of tests for evaluating the shuttle design.

2. Where special software has been incorporated to permit the computer to operate properly in the lightning environment, appropriate software verification tests must also be developed.

1.2.4 Design

1. A lightning detector is required to alert the computer to enter the appropriate operational mode for this environment. In particular, the computer must be alerted to the possibility that interface and subsystem data may be contaminated. This implies that the lightning detector mechanism has sensors distributed throughout the structure of the vehicle.

2. All memory in critical subsystems must be re-establishable or must be protected from loss.

3. The computer must possess sufficient power reserves to operate through a period of transient external power loss. For longer periods, this would require the capability to shut down in an orderly fashion so as not to lose data or machine state information.

4. If shielding of the computer is not sufficient to protect the contents of logic, then the lightning detector must shut down the memory in order to prevent more than a single memory word loss. Software techniques for controlled roll-back recovery must be implemented.

5. Several computer manufacturers claim that lightning is not a problem if they design to MIL-STD-461A criteria. This claim should not be assumed valid until further supporting evidence is developed. Present visibility into design details is not sufficient to accept the claim. Subsystems external to the computer will, of course, also impact the survivability of the shuttle system. The Apollo 12 incident is an example.

1.2.5 Architectural Considerations

1. Computer hardware/software must operate error-free during power transients.

2. For the implementation of controlled rollback recovery techniques, we recommend a lightning detector which is sufficiently fast to alert the computer, to close the door to the memory, and to limit memory errors to the word currently being operated upon.

3. The controlled rollback recovery techniques for DRO memory are decidely more complex and inefficient of time and storage than those techniques required for NDRO memory. This is primarily due to the fact that, in a DRO memory, data can be destroyed during a read operation as well as during a write operation.

4. Given (2), NDRO memory can guarantee controlled recovery in the face of a second strike which may occur during the recovery attempt from an earlier strike. DRO memory cannot guarantee recovery under this condition.

5. Given that a high probability of protection can be obtained for the computer by proper shielding and grounding techniques, serious consideration should be given to contrasting the additional protection gained by implementing a controlled rollback recovery scheme (to cover the remaining possibility of data loss), with the potential error modes and inefficiency introduced by the implementation of such a complex and costly scheme.

6. There are other software approaches to recovery from lightning induced errors which are simpler to implement than controlled rollback. Specifically, a study might reveal that in flight regimes subject to lightning, the vehicle state can be recovered sufficiently well from onboard sensors or the ground without dependence on the contents of computer memory. This procedure may be cheaper and easier to verify than controlled rollback. These other approaches need more detailed analysis.

2.0 LIGHTNING PROPERTIES AND EFFECTS ON AEROSPACE VEHICLE AVIONICS

2.0 LIGHTNING PROPERTIES AND EFFECTS ON AEROSPACE VEHICLE AVIONICS

Frequently, electrostatic discharges are considered along with lightning. Here, the discussion is constrained simply to the consideration of lightning. In an electrostatic discharge, the aircraft itself serves as a charge center. This charge may be picked up while flying through dusts, clouds or precipitation. In rocketry, the plume will serve as a low current generator. In either case, the charge is generally dissipated via static discharge wicks and corona discharge at sharply curved and pointed parts of the airframe before charge buildups comparable to that involved in lightning strike currents are experienced. Lightning strikes require electric fields of several thousand volts per centimeter to breakdown the air and the resulting current is correspondingly high, involving much more energy than that associated with electrostatic discharges. Passing reference should be given, however, to the loss of two early Minuteman flights attributed to static discharge problems due to improper grounding between the boost and upper stages.[19]

2.1 Review of Recent Air Force Lightning Strike Data

Worldwide Air Force lightning strike data from 1965 to December 1971 has been stripped out from the files maintained at the Headquarters Air Force Inspection and Safety Center, Norton Air Force Base. This information is organized principally on the basis of the degree of aircraft damage. The categories are delineated as major accidents, minor accidents and incidents. One lone event is reported involving pilot injury, but no aircraft damage.

Five cases, where lightning was a factor, are described under the category of major accidents. In four of the five cases the aircraft was destroyed. The remaining case involved extensive structural damage. Two of the four cases where the aircraft was destroyed resulted in fatalities. Generally lightning was the initiating cause, but not necessarily the single cause of the accident. The pilot also sometimes played a part in the events which led eventually to the accident. In one case the aircraft commander, confused by lightning, failed to switch the fuel selector from empty inboard tanks to full outboard tanks. In another case the pilot was temporarily blinded during a critical flight phase and ejected when the airspeed moved rapidly from 230 knots toward 700 knots. On the other hand blade failure, induced by a lightning strike, in the main rotor system of a helicopter rendered it uncontrollable by the pilot and resulted in a crash and the death of all eight persons onboard.

Four minor accidents, where lightning was a factor, are detailed for this 1965-1971 time period. Generally, these accidents all involved structural damage, although in all cases the aircraft were able to land safely. The structural damage included wing damage, loss and explosion of external fuel tanks, radome damage, pitot boom loss and radar antenna loss. Effects pertinent to aircraft electrical and avionics system included:

1. Fuel ignition in the wing subsequent to a lightning strike in one aircraft was attributed to the arcing of a shorted navigation light power line.

2. Lightning strike initiation of jettison squibs.

3. Both power generators were knocked off the line by a lightning strike and subsequently successfully reset.

Some four hundred inflight incidents are itemized in which lightning was a factor. The aircraft damage associated with these incidents is characterized as either slight or none. Frequently

the description merely indicates that lightning struck and will give the location of the strike; e.g. a wing tip, a radome, an antenna or the vertical stablizer. Where more information is given, it is likely to be terse. Generally, the documented reports are incisive and likely not to describe electrical malfunctions to the detailed level desirable for the purposes of this study. In fact, it is likely that many electrical malfunctions occurred which were not even mentioned. However, in roughly 5% of these incidents, the description indicated electrical malfunctions which resulted from a lightning strike.

These malfunctions were manifested in a number of different fashions as is indicated by the following summary observations.

*Five incidents involved the accidental deployment of crash position indicators.

*Twenty-five percent of the strikes struck radomes. From these 100 strikes only four reports indicated difficulty with radar operation, one of which indicated only a temporary failure of one minute duration.

*Five strikes resulted in generator failure. In these cases, three reports indicated that the generators were recycled and put back on the air.

*Erroneous malfunction indications - door open and fire warning - were experienced, each in a separate aircraft lightning strike incident.

*Power supply damage was indicated for subsystem electronics in one strike report.

*Tacan and ILS malfunctions were indicated in one strike report.

One case was reported where a lightning strike injured the pilot, but no aircraft damage was incurred. The pilot felt his

scalp tingle and suffered second degree hand burns.

Assessment of this data, with an eye toward lightning effects on the shuttle digital equipment, especially the computer hardware, indicates that lightning is a very real hazard. Lightning strikes do occur and they have induced not only transient signals in aircraft electrical system, but have also inflicted permanent damage on electrical equipment. For a more detailed understanding of the likelihood of lightning strikes and its effects on aircraft electronics; however, the review of other information sources was found desirable.

2.2 Some Properties of Lightning

2.2.1 Physical Properties

A number of sources exist in the literature which describe the properties of lightning.[1,8,12,14] The following, which is derived from Reference 12, summarizes lightning characteristics as generally described throughout the literature.

"1. Intracloud Lightning. Lightning which does not connect to ground, although dissipating amounts of electric charge and energy similar to those that do, does not generally involve currents greater than 1,000 to 2,000 amperes with maximum rates of rise probably not exceeding 100-500 amperes per microsecond. The average total duration of these currents does not exceed 3 milliseconds.

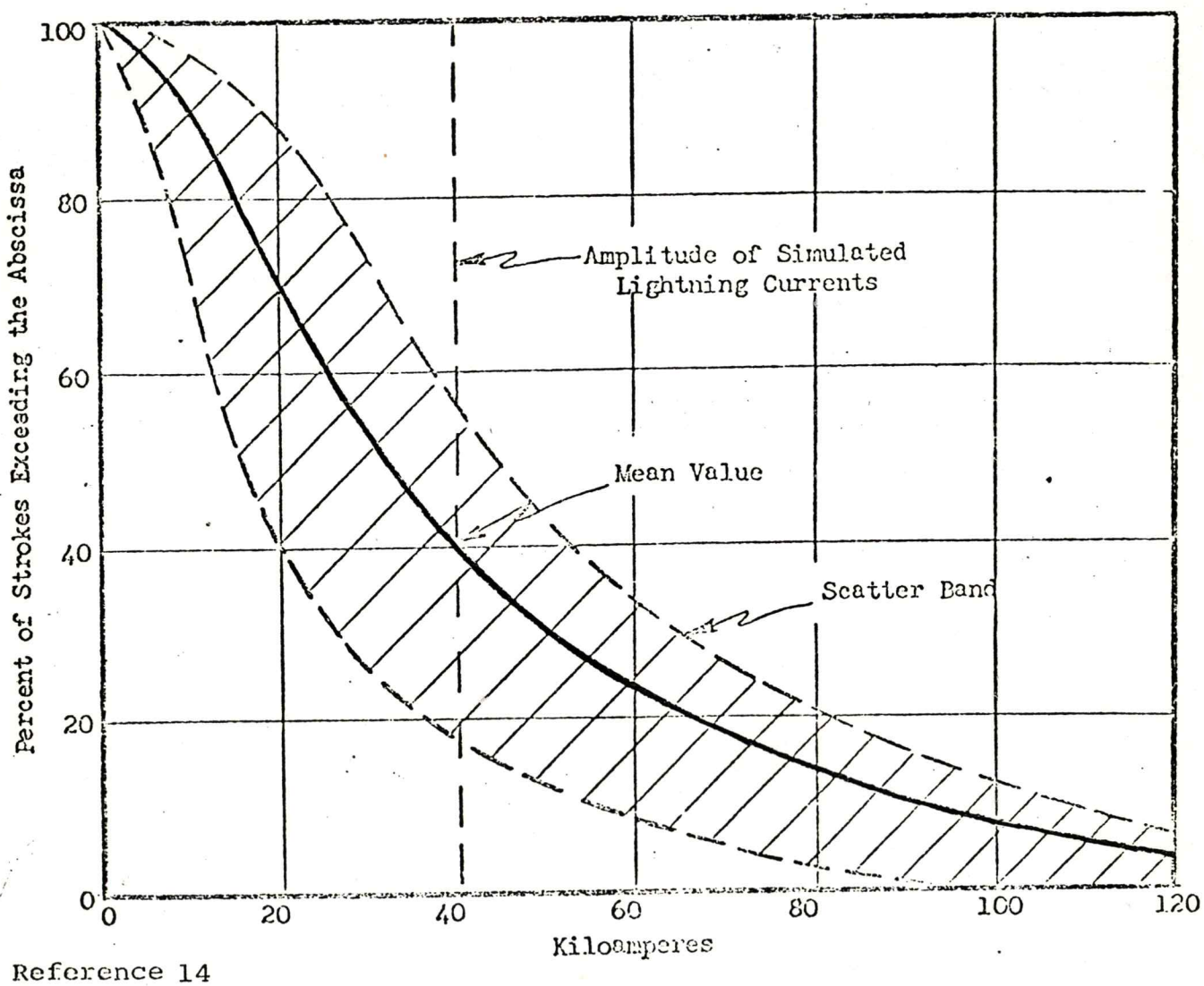
2. Discrete Lightning Strokes to Ground. Lightning which reaches ground involves a low current leader followed by a return stroke with an average peak current value of 20,000 amperes, and with a rate of rise of about 10,000 amps/microsecond. The current falls to half value in about 40 microseconds and is essentially at

zero value after several hundred microseconds. On the average there are about 3 or 4 strokes to each discharge, with a time between strokes of about 40 milliseconds. The first stroke in a discharge usually carries the largest current.

3. Long Continuing-Current Lightning Strokes to Ground. About one out of 5 or 6 strokes to ground is initiated by a leader followed by a discrete return stroke in which the current does not fall to zero value after a few hundred microseconds, but which continues at an average current value of about 185 amperes for an average duration of about 175 milliseconds. Continuing currents of 250 amperes lasting for about 0.25 seconds are not uncommon.

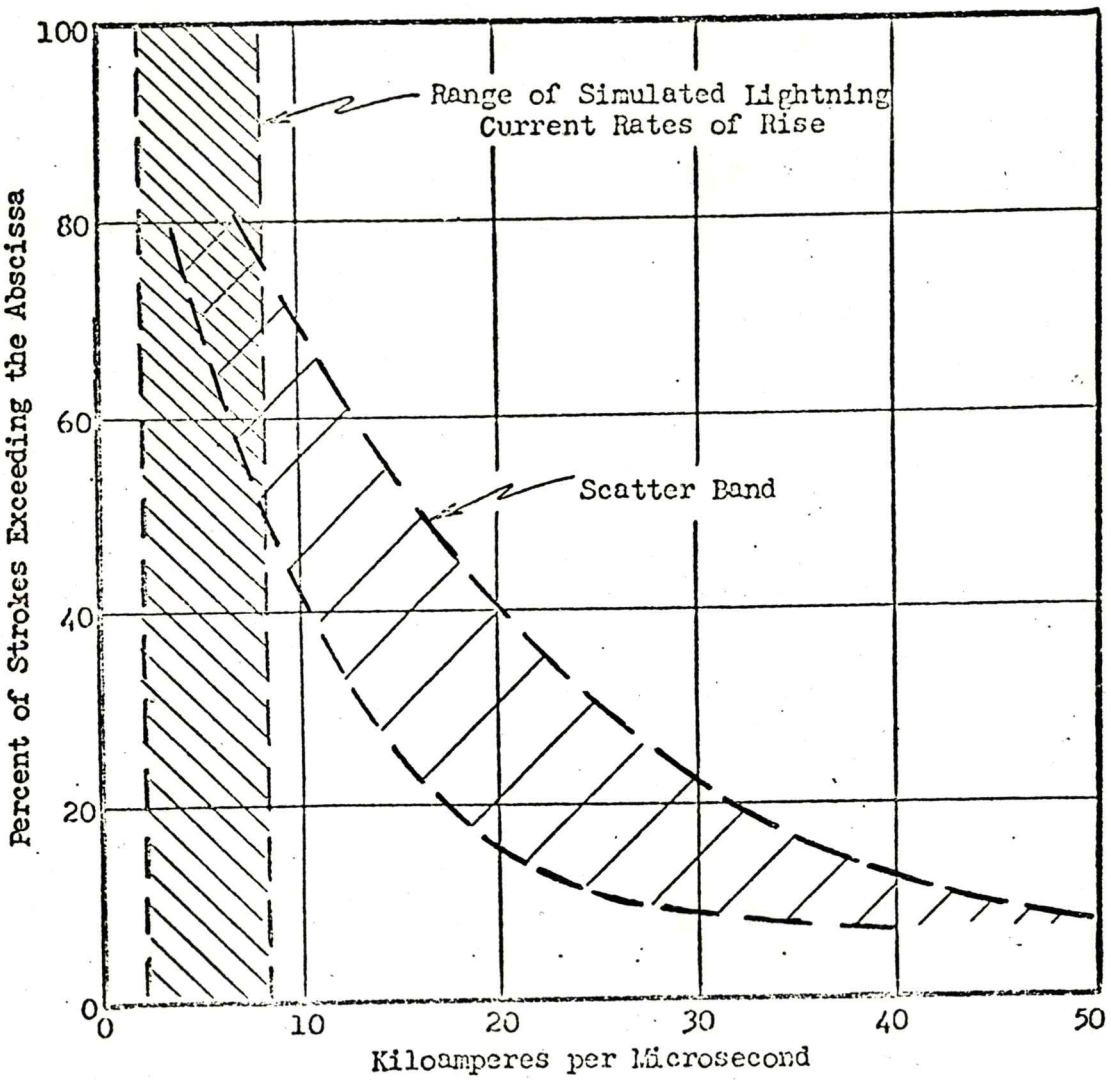
Summarizing, high currents and high rates of rise of current are not expected from intracloud strokes; rates of rise of the order of 10,000 amps/microsecond are to be expected from discrete return strokes, each involving from 1 to 5 coulombs of charge; long continuing-current strokes involve high rates of rise as well as persistent currents of about 185 amperes for periods of about 0.2 seconds bringing from 12 to 40 coulombs of charge to earth. In terms of energy, the continuing currents involve at least an order of magnitude greater energy release than do ordinary discrete return strokes."[12]

Reference 12 goes on to imply that lightning triggered by a shuttle-like vehicle is likely to be of the long continuing-type described previously. Figures 1 and 2 depict the likelihood of lightning current amplitudes and current rise rates for lightning strokes to ground. NASA guidelines for space vehicle development are largely consistent with the characteristics discussed thus far. Table 1 summarizes these NASA guidelines.



Reference 14

FIGURE 1. - PROBABILITY DISTRIBUTION OF LIGHTNING STROKE PEAK VALUES



Reference 14

FIGURE 2. - PROBABILITY DISTRIBUTION OF LIGHTNING STROKE RATES OF RISE

TABLE 1. CHARACTERISTICS OF LIGHTNING DISCHARGES

Type of Lightning	Average Peak Current per Stroke (A)	Maximum Rate of Rise of Current (A/μsec)	Average Amount of Charge Transferred		Average Total Duration of Stroke (msec)	Average Number of Strokes (unitless)	Average Time Between Strokes (msec)	Remarks
			Per Stroke (C)	Total (C)				
Intercloud lightning	100 - 2000	100-500	1-5	1-5	300	1		
•Discrete lightning strokes to ground								
Leader	100		1-5	5	20	1		
Return stroke	20 000	10 000	5	4-20	0.3	3 to 4	40	Peak current exceeding 100,000 A have been measured about 2 percent of the time.
Long continuing current lightning strokes to ground								
Leader	100		1-5	5	20	1		
Return stroke	20 000	10 000	12-40	12-40	200	1		Average current value of 185 A for long periods (175 msec).

Reference 1

Assessment of the radiation which results from lightning currents is not a simple matter. The geometry of the vehicle, location of the strike, bonding practices, skin thickness and continuity, and relative location of the electronics to the lightning current all contribute to a highly complex situation. The following simple model provides an indication of the magnetic field intensity by calculating the field associated with current flow on a wire. This model yields a circuital magnetic field whose magnitude is given by equation 1. At a

$$H = \frac{i}{2\pi R} \text{ amp-turns/meter} \qquad (1)$$

distance of $1/2\pi$ meters, a current of 200 KA and a current rise rate of 100 KA/μs, this model yields an H field of 2×10^6 amp-turns/meter and a $\dot{H}$ field of 10^{11} amp-turns/meter-second. These field strengths represent a worse case figure of merit from the point of view of the values shown for i, di/dt and R as well as from the geometry associated with the model. The values chosen for i and di/dt are consistent with those used in military specifications.[4] The value of R is considered to be representative as roughly a worse case value for equipment location in the shuttle.

2.2.2 Likelihood of Lightning Strikes

The likelihood of aircraft strikes has been assessed from a number of points of view. Studies have analyzed the frequency of lightning strikes as a function of altitude, temperature, geographic location, and time of year. Data is available which correlates real lightning strikes with the aircraft altitude and atmospheric temperature. An early study[15], 1935-1944, of lightning strikes based on 170 reports showed 75% of the strikes to occur between 3000 and 9000 feet and that over 80% of the strikes occurred with outside temperatures in the $\pm$ 5°C temperature interval. This early data was biased in the sense that the majority of the aircraft in this time period were nonpressurized

and constrained to fly at altitudes less than 15,000 feet. Subsequent strikes however have tended to confirm these observations. A study of Air Force[7] incurred lightning strikes in the 1961-1964 time period showed the majority of the strikes to occur below 15,000 feet even though such altitudes are far below the normal jet cruising altitudes. Data accummulated by the British on commercial flights during the 1960's provides a third source of information. This information shows 80% of the documented lightning strikes to have occurred from 2 to 12 thousand feet and 55% occurred where the outside air temperature was between $\pm$ 5°C (75% between $\pm$ 10°C). (10)

With respect to the higher altitudes, the Air Force data showed 2 strikes above 30,000 and none above 35,000. The British commercial experience showed only 2% above 20,000 feet. This data, as was the 1946 data, however, is also biased by the operational practices. Frequently, IFR flights encounter low altitude holding patterns in bad terminal weather. Conversely, high altitude flights are less likely to incur thunderclouds and adverse weather conditions condusive to lightning, and are freer to modify their flight plans so as to avoid such weather conditions. Nevertheless the existing data does suggest that lightning precautions for the shuttle need only consider the lower altitudes of atmospheric flight. Exactly where this line might be drawn is considered beyond the scope of this study although the limited data reviewed here suggests somewhere in the 20,000 to 35,000 foot region.

The number of thunderstorm days per year (isoceraunic level) provides another indication of the likelihood of lightning striking the shuttle. Table 2 lists this information for locations of particular interest. This table clearly indicates the relatively high incidence of thunderstorms at Cape Kennedy, particularly in the summer months. A simplistic interpretation of this information infers that the likelihood of lightning strikes in WTR Shuttle operations is more than an order of magnitude less than at ETR.

TABLE 2

FREQUENCY OF OCCURRENCE OF THUNDERSTORM DAYS (ISOCERAUNIC LEVEL) (1)

Location	Mean Number of Days Per Year of Thunderstorms
Eastern Test Range	70.09
Vandenberg AFB, California	2
Edwards AFB, California	4.3

Obviously, such an observation is subject to the objection that it does not take into consideration operational constraints such as the Apollo practice of not launching into adverse weather conditions.

Another indication of the likelihood of lightning strikes is given by the observation that "world-wide USAF operations encounter such electrical incidents at the rate of about 50 per year. Domestic air carriers experience about 750 incidents per year, or about 1 per 2,000 hours of flight".[11]

Probably the most dramatic lightning strike incident is the Apollo 12 strike.[13] The subsequent investigation to this incident found that the Apollo design had an inherent degree of protection from the effects of lightning, but that launch restrictions must be introduced to avoid potentially hazardous electric fields.

2.3 Some Experimental Results

2.3.1 Flight Test Data

In 1964-1966, a multi-aircraft lightning research program was conducted using instrumented C-130, C-100F and U-2 aircraft.[9] A ship equipped with line-throwing rockets was used to trigger lightning strikes while the test aircraft flew overhead. Transient currents, RF waveforms, electrostatic fields, precipitation, turbulence, photographs and radar measurements were obtained. Some of the data derived from this test data is listed in Table 3. This data is less severe by an order of magnitude or more from that described previously for cloud to ground strikes. Reference 11 observes that "these strikes are believed representative of incloud or intracloud discharges and not the more extreme values in a cloud ground channel."

TABLE 3

LIGHTNING PARAMETERS

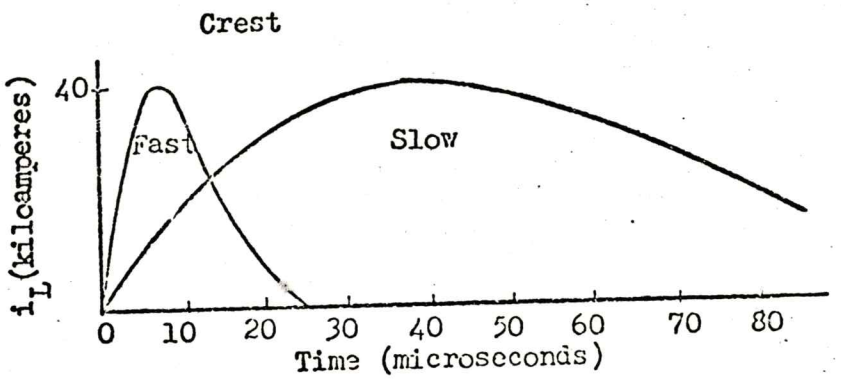
Parameter	90%*	50%*	10%*	Maximum Observed	Number of Observations
Crest Current	150 to 200a	1.6 to 2 Ka	6 to 7 Ka	22 Ka	73
Current Rate of Rise	70 to 100 a/μsec	700 a/μsec	2 to 3 Ka/μsec	5 Ka/μsec	36
Current Rate of Decay	30 to 40 a/μsec	100 a/μsec	550 to 600 a/μsec	7 Ka/μsec	37

*Percent of strokes which will have parametric values exceeding those indicated.

2.3.2 Lightning-Induced Voltages In Aircraft Electrical Circuits

Noise induced by lightning may enter computer electronics in a number of ways. The properties discussed earlier are obviously not directly applicable as design criteria. No reasonable design would permit the thousands of amps associated with a lightning strike to flow directly through the case or on the cabling associated with computer electronics. The real problems which must be dealt with are associated with induced voltages.

An indication of the voltage magnitudes which might be expected from lightning induced voltages are provided by experimental data.[14] The experiment referred to applied simulated 40 Ka lightning strokes (Fig. 3) with 2 Ka/μs and 8 Ka/μs rise rates to an F89J wing; and measured the resultant induced voltages and currents in the wing electrical circuits. Test measurements were



Simulated Lightning Currents

Reference 14.

Figure 3.

performed upon eight existing wing circuits, each having its own unique characteristics. Some had dedicated returns, others used the airframe; some were shielded, others were not, and all had unique wire paths.

A report on this experiment observes that "induced voltages ranged between several millivolts in well-shielded high-impedance circuits, and one hundred volts in poorly-shielded low-impedance circuits, which utilized the airframe as the circuit return path." (14) The data shows that all circuits with airframe returns exhibited induced voltages greater than 1 volt. Such circuits with longer wire runs and less inherent structural shielding exhibited induced voltages greater than 10 volts, in fact nearly 100 volts. Conversely the circuitry with dedicated returns exhibited induced voltages less than 10 volts in magnitude, frequently less than 1 volt and in a circuit with individual shielded conductors no more than .2 volts. In the worst case, the wing tip position lights, the induced voltages reflected both significant resistive drop as well as inductively coupled voltage.

A special series of tests were run to study the effect of lead length, and the effectiveness of parallel conductors, twisted conductors and coaxial shielding. Special purpose conductors were strung through the wing structure for this series of tests. Table 4 depicts the results of this experiment. As was to be expected the longer conductors exhibited higher voltages. The twisted pair exhibited induced voltages roughly a factor of 5 less than the parallel set relative to the airframe and a factor of 10 less conductor to conductor. No substantial improvement is associated with the shielded vs unshielded measurements. This last effect was noted to be related to the inherent shielding afforded by the paths selected within the wing.

2.4 Design Practice Versus Lightning Effects

2.4.1 Conventional Design Practice

It is to be expected that a lightning environment is more severe than that associated with conventional design practice. In fact, some concern is appropriate as to whether off-the-shelf equipment incorporated into the Shuttle with conventional design practice will either survive or operate properly in a lightning environment. The following is intended to illustrate one such conflict where conventional electronics might be expected to fail.

Electronics designed to meet conventional military standards (MIL-STD-461A) regarding electromagnetic susceptibility are required to function with various types of noise imposed through the power leads. This noise consists of relatively low level sinusoidal noise from .03 to 400 MHz and a 10 microsecond pulse with a magnitude of 100 volts or twice the line voltage, whichever is less. It is common aerospace design practice to use the airframe for the power return in the design of conventional electronics. With such a design a lightning strike could result in 100 KA in this power return which could certainly yield a power transient in excess of that stipulated above. The tests[14] discussed early substantiate this possibility. It is not unlikely that such a lightning current would result in a failure in a computer which was designed to comply with such conventional design practices.

2.4.2 EMP Design Practice

Equipment designed to function in an EMP environment is subject to conditions not entirely unlike those associated with lightning. In both cases the equipment is being asked to function in an environment associated with a high energy electrical phenomena.

One textbook on nuclear hardening[22] describes EMP as a pulse whose width is measured in tens of nanoseconds, with field strengths and rates on the order given in Table 5. It should be

TABLE 4 - MAXIMUM INDUCED VOLTAGES AND CURRENTS IN NEW WING CIRCUITS

(Series 2 - All Circuits and Shields Connected to Airframe at Location Between Aileron and Flap on Trailing Edge. Identical Measurements on Circuits Terminating at Leading and Trailing Edges)

Conductor	Open Circuit Voltage (Volts)				Short Circuit Current (Amps)			
	Conductor-to-Airframe		Conductor-to-Conductor		Conductor-to-Airframe		Conductor-to-Conductor	
	Leading Edge*	Trailing Edge*	Leading Edge	Trailing Edge	Leading Edge	Trailing Edge	Leading Edge	Trailing Edge
Unshielded #16 Insulated Conductor	2.0	0.4	--	--	1.6	1.3	--	--
RG 58A/U Coaxial Cable	2.1	0.4	--	--	2.2	0.4	--	--
Twisted Pair of #16 Insulated Conductors	1.0	0.5	0.22	0.04	1.4	0.9	0.1	0.1
Parallel Pair of #16 Insulated Conductors	6.0	1.0	2.3	0.1	6.0	2.0	1.3	0.1

*NOTE: Circuits terminating at leading edge are 38 feet long.
Circuits terminating at trailing edge are 12 feet long.

noted that the magnitudes quoted in Table 5 can only be used as a rough indication of actual EMP requirements imposed on various applications. These requirements are, in general, classified.

A comparison of EMP with lightning shows EMP to be a higher frequency phenomena by roughly three orders of magnitude. EMP also possesses different spatial characteristics in that it is radiation which would impinge essentially uniformly on the Shuttle, whereas lightning is most likely to strike the extremities of the Shuttle and the current will seek a path of its own choosing, which is not likely to be uniform.

The relative severity of these two phenomena relative to the Shuttle application is beyond the scope of this present study. It is possible to observe, however, that the design practice required by these two phenomena is similar. In both cases special and similar shielding, grounding, circuit hardening and architectural design practices are required as discussed in Section 3.0 and 4.0. The differences are largely ones of degree.

The similarity of lightning and EMP permits the use of existing hardened computers as examples of existing designs which should survive and operate through a lightning strike. The testing practices associated with such computers provide a data base which serves to assure the feasibility of designing a computer for a lightning environment in the Shuttle application.

It is not possible, however, to extrapolate directly from the existence of hardened computers to an obvious solution for the Shuttle lightning environment for many reasons. For example, hardened computers generally assume that the semiconductor portion of the computer (or a large majority thereof) cannot be adequately shielded from gamma radiation to prevent erratic operation. Circumvention techniques are incorporated which reinitiate the

computer subsequent to the radiation incidence based upon good information retained in a hardened memory. The transient operation associated with EMP is sufficiently similar to that associated with gamma radiation such that the circumvention required for gamma radiation will also circumvent EMP effects, thereby negating to a large extent the need for shielding the machine from EMP transient effects. Thus, it is not clear that a hardened computer does or does not operate, error free, through an EMP environment or that it would operate, error free, through a lightning strike. If there were a lightning detector, the circumvention capability in such a computer would provide protection from the errors which might be induced by lightning.

TABLE 5

TEXTBOOK ESTIMATE OF NOMINAL EMP RADIATION LEVELS[17]

E	10^5 volts/meter
$\dot{E}$	10^{18} volts/meter-second
H	10^2 amp turns/meter
$\dot{H}$	10^{15} amp turns/meter-second

2.5 Failure Modes Associated with Lightning Strikes

A number of failure modes are associated with avionics electronics due to a lightning strike. In general, such failure modes may be avoided through appropriate design practice.

2.5.1 Cable Failure

Cables which run to the extremities of a flight vehicle are susceptible to becoming a direct current path for lighting currents. In such cases, the cable is likely to melt and neighboring cables within a bundle are also likely to be damaged. Cabling

to navigation lights is typical of wiring susceptible to such failures.

2.5.2 Semiconductor Failure

Lightning current and induced voltages have more than sufficient magnitudes to break down and burn out semiconductor electronics. Semiconductors which are located directly in a lightning current path, such as thermocouple sensors on the skin of a vehicle,[13] will inevitably be burnt out by such high currents. Induced voltages in signal paths which use the vehicle airframe as a ground return are also highly susceptible to voltage break down as well as burn out conditions. Induced voltage break down is also a likely failure mechanism where dedicated returns are used if appropriate grounding, shielding and twisted pair design techniques are not incorporated in the design. Appropriate shielding and grounding techniques are also required for subsystem electronics to avoid induced voltage breakdown of semiconductor components.

2.5.3 Other Component Failures

Other components, such as coils, transformers, resistors and capacitors are also susceptible to lightning current and induced voltages. Components incorporated in receiving and transmitting circuits are particularly susceptible to lightning currents which find their way directly onto the associated transmission lines. Coupling transformers are also susceptible to high voltages in typical ground isolation applications. Generally, the induced voltages internal to a shielded case are unlikely to exceed the break down voltages of these components although this failure mechanism cannot be dismissed entirely.

2.5.4 Transmission Noise

Signals transmitted from sensors, to effectors and between subsystems are highly susceptible to lightning induced noise. Long cable runs are inherently susceptible to noise pickup through both inductive and capacitive effects. Shielding, dedicated returns and twisted pair signal transmission serve to attentuate such noise susceptibility, but in an application which must live with such severe weight constraints as the Shuttle, these techniques may prove to be impractical for all signals. Grounding problems are also a severe problem in such a distributed electronic system. Good practice associated with lightning design requirements may well enhance signal noise sensitivity for transmitted signals.

2.5.5 Electronic Noise

The electronics are susceptible to lightning induced voltages if not properly shielded and grounded. Here the susceptibility is less of a problem than in the case of transmission noise. Shielding can be localized to the space allocated to the electronics with less weight penalty than that associated with a multiplicity of long cable runs. The localized grounding problems also are less severe.

Care must be taken to insure that noise is not introduced into an electronics package via its power and signal cabling. In the Apollo 12 lightning strike, the umbilical cabling was suspected to have provided a path for the noise which initiated the fuel cell disconnect.

2.5.6 Power Loss

Systems which use the airframe as a power return are highly susceptible to lightning induced power transients. They are also likely to incur transient power loss when the power systems are switched off line by surge current protective devices triggered by

lightning induced power transients. Aircraft lightning strike reports document such generator shutdowns. The Apollo 12 incident serves as another data point where the lightning strike induced a transient power loss.

3.0 DESIGN PRACTICES FOR SURVIVAL IN A LIGHTNING ENVIRONMENT

3.0 DESIGN PRACTICES FOR SURVIVAL IN A LIGHTNING ENVIRONMENT

The methods used to prevent lightning or EMP from inducing transient errors into avionics systems are mainly methods of packaging to provide shielding from the environment and proper grounding. In computer design there are also architectural constraints that are desirable or perhaps necessary, as discussed in Section 4.0. The effects of EMP and lightning are similar. Generally EMP is significantly worse such that a design which provides sufficient shielding for the lightning environment does not necessarily provide adequate protection from EMP. A design sufficient for the EMP type environment should be sufficient for the lightning. This study will define desirable design features for protection from lightning and indicate additional features necessary for EMP.

3.1 Lightning Characteristics

A brief summary of the characteristics that are significant to this study is that peak currents of 100,000 amperes with rise times of about 50,000 amp/microsecond are possible under the right conditions. Each lightning discharge can be from a few microseconds up to hundreds of milliseconds in duration. The references (1,12,14) provide more information on the characteristics but this data is sufficient to provide a guide to the design of protective shielding and grounding.

3.2 Shielding

With no limit on weight, it is possible to provide complete protection by shielding. However in an application like the Shuttle the problem is a complicated trade between the weight penalty of complete protection and providing adequate protection within a reasonable weight budget.

The outside skin of the vehicle should provide a high degree of shielding but holes in the skin for lights, antennas, windows,

etc. and a fairly high skin impedance make it necessary to provide additional protection internally. The first criterion, which is more or less normal design practice, is to provide as low an impedance as possible in the skin and supporting structure of the vehicle.

Internal to the vehicle, the ultimate of a Faraday cage for all electronics is not practical since there must be holes for controls and cables. In addition, the size of the vehicle with the electronics distributed at remote locations would prevent the design of an all inclusive shield. With careful application of good shielding technique and the application of filtering at critical breaks in the shields it seems feasible to provide complete protection from lightning with more or less normal shielding and grounding practices. Following is a list of the major constraints on the shielding and grounding.

a. Any conductor path which may originate at a hole in the skin of the vehicle like wing lights or antenna wiring must have special attention.[22] Lightning arrestors and filters are needed to attenuate the disturbance at the point of entry. All wiring and conductor paths are critical even though they are not directly related to the digital avionics since the high currents can enter the vehicle at these points then reradiate into the electronics or set up large voltage drops in the shields and grounds. For example, many aircraft will wire the wing lights with one wire using the skin as return without any provision for filtering the transients in the wire. This method of wiring provides a very easy point of entry for large voltage transients which can reradiate to more critical signal paths.[14]

b. Power wiring must be two wire and should be twisted

pair. It would even be desirable to shield the power wiring and to provide separate power wiring for each critical subsystem. In contrast, many systems use the vehicle structure for power return in order to save weight. In this case lightning strikes will couple directly into the power ground as a result of the large voltage drop in the skin resulting from lightning induced currents flowing. Two wire, not twisted and shielded, systems can still couple magnetically and electrically.

c. Each critical subsystem must have internal power filtering and must have the case isolated from the power ground. The filter must protect the subsystem from noise induced between any combination of the two power leads and case. Many digital equipments do not have sufficient filtering and are sensitive to noise between power leads and case.

d. All signal transmission between systems must be balanced line transmissions (sending and receiving ends both balanced) using shielded twisted pairs. The twisted pairs could be bundled and shielded if the coupling between signals is small enough to be tolerable. The shields must be multiple point grounded using a technique which will insure a very low impedance ground at all frequencies. In contrast, normal practice permits unbalanced drivers and receivers, single wire transmissions with a common signal ground, and shields with single point ground (usually with a ground wire many inches long). Such practice permits noise to electrically couple through ground current induced voltage drops which might be generated by lightning and magnetically couple due to high currents in the vehicle skin.

e. With the long wires in the Shuttle vehicle, special attention must be given to the wavelength of these wires and shields. Multiple point grounding of shields is required due to antenna effects when wire lengths exceed .15 of the impinging radiation wavelength.

f. Wire routing between interface connectors and interface circuits internal to the case of an electronics subsystem must be arranged to minimize coupling with other internal signal paths.

g. Electronic subsystems which contain digital electronics should be provided with good solid ground planes for internal signals and the case of the subsystems should be designed to provide shielding from electric fields. At this level, the Faraday cage approach seems practical.

h. Ground zones should be established at the subsystem level. Isolated signal grounds should be incorporated on a module by module basis where each module is encased in its own Faraday shield.

3.3 Bonding

Any two points on a metallic structure whether electrically connected or not will develop a potential difference at some frequency. For example, when the structure dimensions are on the order of magnitude of a wavelength, the potential difference will exist in the presence of electric and magnetic fields. At lower frequencies, the potential difference between two points in the structure will be proportional to the impedance between the points. Reducing the impedance will reduce the potential difference. Good bonding between structural elements provides low impedance paths thus limiting the electric potential between various points at the cost of higher currents. Normally this is considered to be the optimum approach since the resulting magnetic field is preferable to the high potential difference between points of high impedance.

The important impedance when considering shielding and grounding for lightning is the impedance at radio frequencies and there is little correlation between the direct current resistance of a bond and its radio frequency impedance. To make the shielding and grounding effective for RF, special precaution must be taken with the grounding straps and bonding.[4,20]

3.4 EMP

There are two characteristics of the EMP environment which are significantly different from those associated with lightning. First, the EMP rise times are two or three orders of magnitude faster than lightning. Second, the environment is distributed more or less uniformly around the vehicle thus causing induced current densities at localized points within the vehicle. The magnetic fields will induce currents in loop areas and the electric fields will induce currents in shields.

The most significant additional requirement for protection against EMP is on the physical layout of the electronic equipment both within an enclosure and in the cabling between enclosures. Cabling should be arranged to eliminate all loop areas by placing it in the form of a tree with control trunk and branches fanning out to appropriate system elements.[20] This would mean separate wiring for the power of each subsystem as suggested in 8.2b. Any loops that still remain should be made as small as possible. Within a subsystem the same principle of eliminating loop areas is important but, in addition, the physical dimensions should be minimized. If there are long signal paths they should be balanced transmissions using twisted pairs.

For the Shuttle application the EMP near field environment may be sufficiently reduced at the ranges specified for Shuttle survivability that the additional requirements indicated above may not be necessary.[26] These additional requirements result from the characteristics of the near field environment.

3.5 Testing

Apollo experience offers some background for testing.[18] In addition to conducted and radiated RF susceptibility tests, testing for susceptibility to radiated transients was introduced in response to the development of major hardware problems associated with AGC sensitivity to static discharges and power line transients developed between the power lines and case. Steps were taken to remedy the problem and a spark gap transient radiation test was introduced to test the susceptibility of the Block II design to such static discharges. Here, the point is that the testing should reflect real hardware and environmental problems.

Similarly special testing procedures will be required for the Shuttle avionics to verify that they will operate as specified in a lightning environment. Conventional EMI testing[5] will not properly qualify equipment for such an environment. Some guidance for these tests is offered by testing performed on missile computers although it is likely that such tests will exceed Shuttle requirements. For example, Poseidon testing[24,25] includes tests where current is pulsed directly into the cable sheathing and the computer case. Facilities used for testing also avail themselves. Such facilities as those at Sandia Corp., Martin-Marietta's long wire test facility, and Air Force Weapons Laboratory, Albuquerque provide examples of what can be done. The lightning experiments,[14] discussed previously, performed at G.E. High Voltage Laboratory are another source of background information on a type of test which might be run and the type of information which is to be gleaned from such testing.

Again Apollo experience offers some guidelines. There it was shown that such critical and cost driving specifications as those associated with lightning should be carefully drawn to properly describe realistic hardware requirements and testing procedures. Such testing procedures must realistically reflect

whether the Shuttle computer electronics must merely survive or whether some portion or the entirety is expected to function with no error throughout one or more lightning strikes.

4.0 ARCHITECTURAL CONSIDERATIONS FOR SURVIVAL IN A LIGHTNING ENVIRONMENT

4.0 Architectural Considerations for Survival in a Lightning Environment

4.1 Introduction

In order to determine ways to recover from a lightning strike, it is first necessary to see what the possible effects of lightning are.

1. All equipment is shielded and grounded so that there are no power transients, equipment damage or information loss.
2. Some information is lost, but there is no permanent hardware damage.
3. Some hardware is damaged, but not all usable strings—since a switch to good equipment must be made, (by methods which will not be discussed here), there will be some information loss. Subsequent recovery is effected as in (2).
4. All strings of hardware are irreparably damaged.

Case 1, the most desirable one, requires no special recovery procedures. The disposition of Case 4 must be left to the ship's chaplain. Case 3 reduces to Case 2 for information recovery purposes, so the focus of the subsequent discussion will be on Case 2. Effort must be made to recover vital information and to restore the state of the computer (this is requisite to recovering knowledge of and control over the state of the system). It can be assumed that lightning transients affect all redundant copies of the computers and buses, so switching to a backup alone is not sufficient to get the system operating again. To hope for anything better than a bootstrap freshstart, redundant information must be available within a single string system.

While the main objective of redundancy is to increase overall system reliability—to protect against physical hardware losses—the purpose of recovery in the face of power or EMI transients is to maintain vehicle stability and to avoid generating dangerously erroneous commands. For

purposes of information recovery considerations, an avionics system has three parts: subsystems (including SIUs and digital portions of subsystem electronics), buses (including dedicated digital interfaces), and computers.

1. The question of subsystem protection will not be addressed except to note that if a subsystem has memory that is volatile in the face of EMI transients, then the data in this memory must be reconstructable, by the same methods used for computer data. Or, it must be periodically stored in a protected area such as the computer, in which case the subsystem must enter a safe dormant state between the time of the disturbance and the time that the computer re-initializes it.
2. Buses carry commands which affect the state of the vehicle. Erroneous commands have potential immediate danger, so preventative techniques which avoid dangerous situations are most desirable.
3. Computers contain data pertaining to the perceived vehicle state, the commanded vehicle state, the mission mode, and the computer's own internal state. Recovery of this information is less time-critical than preventing erroneous commands on a bus, but still must be accomplished on a timely basis. Much emphasis has been given to the problems of computer recovery, for overall system recovery is very difficult to accomplish without direction from a properly functioning computer.

4.2 Bus Commands

If an EMI transient above a pre-specified level occurs, it is safe to assume that bus errors occur also. Lightning and other EMI transients are burst phenomena and cannot be reliably detected by parity codes or corrected by simple Hamming codes. More complicated codes such as BCH and Fire codes are effective for correcting data after short noise bursts, but these codes are expensive to implement.

The basic problem with bus traffic is to avoid erroneous commands which could endanger the vehicle. The problem of preventing vehicle danger can be solved by employing temporal redundancy over a time period which is longer than any expected noise bursts. An example of this technique is to employ "arm-and-fire" commands to an engine. Both commands must be received and properly interpreted by the engine subsystem. If one of the commands is erroneously generated then nothing will happen. (This technique presupposes a hard memory at the subsystem level.)

Some commands must be sent out at such a high rate that the "arm-and-fire" technique is not feasible. Commands of this type are usually incremental in nature (such as engine gimbal delta angles) and the loss of just one increment is not a serious matter. This is in contrast to whole angle transfers where an erroneous transfer that is not detected can have devastating results.

The most important part of recovering from erroneous bus commands is to realize that one might have occurred. The computer's reaction should be to safe all subsystems that may have been activated.

4.3 Computer Information Loss

If lightning or other EMI transients get into a computer, vital system state information may be destroyed. Such information losses have been divided into four categories:

1. Central registers and an indeterminate number of main memory locations are erroneous (due to a transient that passes through the memory stack)
2. Central registers and one main memory location are erroneous (due to a transient that affects a memory access)
3. Central registers are erroneous but main memory is intact
4. Power transients

This section explains the recovery techniques for these different categories of computer information loss.

4.3.1 Category 1—All Memory and Register Information is Lost.

Loss of information contained in the CPU registers and in the main memory of the computer is a rather serious problem. Data loss in an indeterminate set of memory locations implies that neither vehicle state information nor computer state information nor any recovery routines stored in electrically alterable portions of main memory can be trusted. Recovery from such an information loss requires a reinitialization of the computer and the vehicle states, in the manner of an AGC FRESHSTART.

In general, a backup store for volatile information (i.e., inputs to the computer or the results of computations based on these inputs) must be, in some sense, volatile itself. Spatial proximity of a backup store to the primary information store will subject both stores to roughly the same external electrical and magnetic phenomena. Given the nature of a lightning strike, it must be assumed that spatial redundancy of volatile information is ineffectual because of common mode failures. That is, redundant copies of main memory are all suspect.

A volatile backup store which exists at a great distance from the primary information store can function as a valid source of the vehicle state. For example, after a massive loss of data in memory, the vehicle state could be recovered from the ground—if there is an adequate communication path. During blackout such a path does not exist.

Recovery from a massive data loss has two objectives: the first is to maintain vehicle stability and to inhibit any action or activity which might be detrimental to crew safety; the second is to determine the present vehicle state and the mission state at the time of the data loss. Only when these steps have been taken is it possible to make a rational decision about how to proceed. • DETECT AN EXISTING HAZARDOUS VEHICLE STATE!

A ground based backup can perform these tasks fairly easily—if a communication path exists. Without such a backup, all information must be recovered from the environment or reconstructed with the help of crew inputs.

1. Program which was in a volatile store must be reloaded from a non-volatile store.
2. The rotational state of the vehicle must be sensed and any excessive rotation counteracted (the appropriate action depends upon whether the vehicle is in a liftoff, an orbital, or a re-entry situation—this can be determined by observation of the vehicle configuration, altitude, etc. or by a crew input.)
3. Engines must be put in a state which does not cause detriment to the vehicle. This may be either on or off and depends upon the particular point in the mission where the memory loss occurs. There will undoubtedly be a heavy reliance on the crew for the mission state information.
4. The vehicle state vectors must be restored—this requires a navigation operation.

Bootstrap programs to perform these operations must reside in non-electrically-alterable program or microprogram memory.

Recovery in the face of massive information loss is an attempt to avoid a catastrophic situation by making use of whatever real world information can be rapidly accumulated. The recovery procedures often must be designed in an ad hoc manner, so a logical recovery cannot be guaranteed 100 percent. However, on the positive side, such recovery procedures may not be complicated or expensive to implement.

4.3.2 Category 2—Loss of a Single Memory Location (during a memory access)

This situation comes about if the computer has a noise detection mechanism which prevents the initiation of a new memory cycle when a potentially dangerous noise event is detected. The rise time of the noise is so fast that the current memory cycle may not be completed properly, causing one word of memory to be lost. It is, however, possible to design a lightning detector which is fast enough to short the memory drivers before a word other than the one being addressed can be affected. After the noise burst is over the computer comes out of "hibernation" and attempts to recover.

When a single memory location is "lost" the object of recovery is to reconstruct the information which was in that location or to nullify the possible detrimental effects of the erroneous information. Two situations arise from the loss of a single memory location. The first case is where the effects of the erroneous information cannot be nullified, because the affected location is "unknown". That is, the affected location cannot be determined by system diagnostics and the data cannot be regenerated by controlled rollback techniques. Since none of memory can be trusted, this case is equivalent to a Category 1 information loss and requires a bootstrap or AGC FRESHSTART type of recovery. The second case is where the affected location can be "patched up" by the system, and the effects of the erroneous information can be nullified. Recovery can be accomplished in a controlled (though complex and costly) manner with minimal system disturbance. Most single memory location losses can be made to fit into the second case by taking the appropriate precautions.

The following rollback schemes consider single noise bursts (e.g., a single lightning strike) which affect only the word being addressed. A second burst that occurs after the computer has come on the air and is attempting recovery is relegated to the realm of diabolical failures. The effects of incorrect addressing and multiple bursts are discussed in Sec. 4.3.2.4 and 4.3.2.5 respectively.

4.3.2.1 Program Losses

Loss of a word of program is a very insidious error, as program cannot be regenerated by rollback operations. Large noise transients can mutilate a program word in a DRO memory during the read or restore cycle of a program word fetch, (e.g., by zapping the memory buffer register (MBR), or the data path from the CPU to the memory). Parity on the affected word may or may not be destroyed. Protection against such a program loss can be accomplished by:

1. Employing a non-electrically-alterable memory for program, such as a rope, a braid, or a semiconductor ROM.
2. Employing an NDRO main memory for program.
3. Providing a program backup in a hard source. This backup is used to reload the program in the event of any indications of excessive noise. WHAT IS A HARD SOURCE?
4. Designing program blocks to contain a vertical parity word. If parity on memory words can detect an error, then the affected word can be reconstructed by a microprogrammed recovery routine.
5. Employing voting techniques to reconstruct the program using information from redundant copies of the computer. If the computers are not bit-level synchronized then it is possible that the same word will not be zapped in each of them.

If a DRO memory is used as the main computer store it is clear that some form of program redundancy is needed. Methods (3), (4) and (5) represent the general approaches to providing the necessary redundancy.

The backup memory for method 3 can be either onboard or on the ground. Interfacing such a backup store to a computer is not very complicated —it can be accomplished via a DMA channel. However, use of this kind of a backup store means either an additional onboard memory requirement or a clear communication path to the ground during critical periods. (A hard onboard store may already exist in the form of a "mass memory", although some of the tape memories being proposed may not meet the hardness criterion.)

Method 4 employs coded redundancy within a single computer. If the expected multiple-bit failure modes are all zeros or all ones, parity on the memory can be designed to catch them (as well as any single bit errors). A microprogram recovery procedure can perform the vertical parity check and reconstruct a bad program word.

In method 5 the computers swap each program word and vote to get the correct value. Operation is predicated on the existence of at least

LOCK STEP SYNCHRONIZATION

three computers, (any fewer does not allow the determination of the correct program word), and a de-synchronization (forced or stochastic) of the computers, so that the same effects do not occur in each one. Communication among the computers can be done via the intercommunication channels which are expected to exist. The entire operation can be conducted by microprogram.

4.3.2.2 Data Losses

Data redundancy cannot be implemented in a static backup. A periodically-updated dynamic backup is necessary. The period of the update may be relatively long (seconds) for vehicle state information, or relatively short (tens of microseconds) for computer state information. There are three feasible approaches to providing the necessary backup:

1. Keep the vehicle state and the mission mode state in a ground-based backup.
2. Rely upon the redundancy afforded by multiple identical computers which operate in a decoupled manner.
3. Multiply store critical variables and rely upon computer phase information to implement an Apollo style controlled rollback.

Method 1 has the same inherent problem as the corresponding method for recovering program information: there must be a valid communication path from the ground. The recovery is not very smooth (relative to other techniques) because only the vehicle and mission states can be restored from the ground. Computer state cannot be salvaged because it changes too rapidly to keep the ground updated. Computations cannot be reasonably rolled back—they have to be reinitiated.

Method 2 works only if there are at least three computers, so that the proper value of data can be determined by a vote. There are serious problems with this approach: loose coupling implies that data may not be updated to the same point in each computer, so a vote becomes difficult if not impossible. The same decoupling problem exists with mission mode variables—one computer may have progressed into a new computation while the others have not. DEPENDS UPON SYNC POINTS CHOSEN

Method 3 implies that sufficient redundant information is stored within a single computer so that a controlled rollback can be effected in the event of the loss of one memory word and the contents of the CPU registers. The mechanization of redundant information storage and the techniques for executing a controlled rollback are described in the following discussion.

4.3.2.3 Controlled Rollback

After a computer error occurs it is desirable to be able to return to a recent point in the program, where data is known to be good, and to resume execution. This approach to error recovery, called controlled rollback, provides a minimum interruption in subsystem servicing, and a minimum impact on subsystem operation.

The first premise of a controlled rollback recovery is that the computer operations can be divided into relatively short segments called phases. The nature of this segmentation must be such that information used as the input to a phase of computation is not destroyed during that phase. The second premise is that phase identification information is always valid. This is necessary if a proper rollback point is to be obtained, i.e., the phase during which the error occurred must be known and the entry point to that phase must be available. This premise implies that an error can be detected during the phase in which it occurs. Such detection appears feasible in the face of large EMI transients, since detection of the transient signals the onset of a possible error.

The first premise will be true if Rule 1 is enforced.

Rule 1: If a value can be destroyed by a memory access, then the value must be stored redundantly in memory. Access to the value during a phase must be limited so that at least one valid and identifiable copy always exists.

SUFFICIENT CONDITION
DISJOINT READ/WRITE SPACE.

The second premise will be true if Rule 2 is enforced.

Rule 2: Phase identification information must be triply stored in memory to account for the situation where an error occurs during the update of the phase pointer.

4.3.2.3.1 Phase Identification

Rule 2 is used by a class of computers (called circumventing computers) to maintain proper phase information. Briefly, the computers update the phase pointer by a special instruction that writes the new pointer value into three consecutive memory locations. The special instruction is a convenience, not a necessity. The update could be performed as effectively by three consecutive write operations. The fact that Rule 2 is a minimum requirement is shown as follows:

1. Consider that the minimum information to identify a phase is a pointer to the beginning of that phase.
2. Consider that the first operation a phase performs is to set the phase pointers in memory to its own entry point.
3. If there are only two copies of the phase pointer and if one of them is destroyed during an update then it is impossible to tell which copy has a proper value (which may be "old" or "new").
4. If there are three copies of the phase pointer and if one of them is destroyed during an update then the proper phase value can be recovered by the following algorithm:
   * If a detected glitch occurs during the update of copy 3 then copy 1 = copy 2 = the new value of the pointer.
   * If a detected glitch occurs during the update of copy 1 then copy 2 = copy 3 = the old value of the pointer (the phase must be repeated).
   * If a detected glitch occurs during the update of copy 2 then none of the copies are equal, but the new value has been properly written into copy 1.

The three copies must be made consistent before program re-execution can begin.

4.3.2.3.2 Data Consistency

Rule 1 implies that a valid and consistent set of data must exist if a controlled program rollback is to work. A glitch must not cause the loss of "too much" data. That is, a consistent set of data must be reconstructable and relatable to a particular phase point. The only types of data which cannot be salvaged are soft subsystem values and incremental inputs to the computer. If hard whole-number subsystem values are not available, incremental transmissions must be limited to "small" increments so that loss of an increment is not critical and results, at worst, in a slight degradation in system accuracy.

Rule 1 can be realized by a multi-phase updating techniques which use shadow, or temporary, variables. Any real time computer with a controlled rollback implementation must rely on this technique in one form or another. Examples of machines which have controlled rollback are the AGC and the DCA.[27]

Recall that Rule 1 says an input to a phase must not be destroyed during that phase. This implies that an input to a phase cannot be over-written during that phase, so updating a variable is necessarily a two-phase procedure. During the first phase the new value of the variable is computed and stored in a temporary, or shadow, variable. During the second phase, the shadow variable is written into the real variable, (at which point the shadow variable can be released for other use). This much is sufficient with an NDRO memory, since a variable cannot be altered by a read operation.

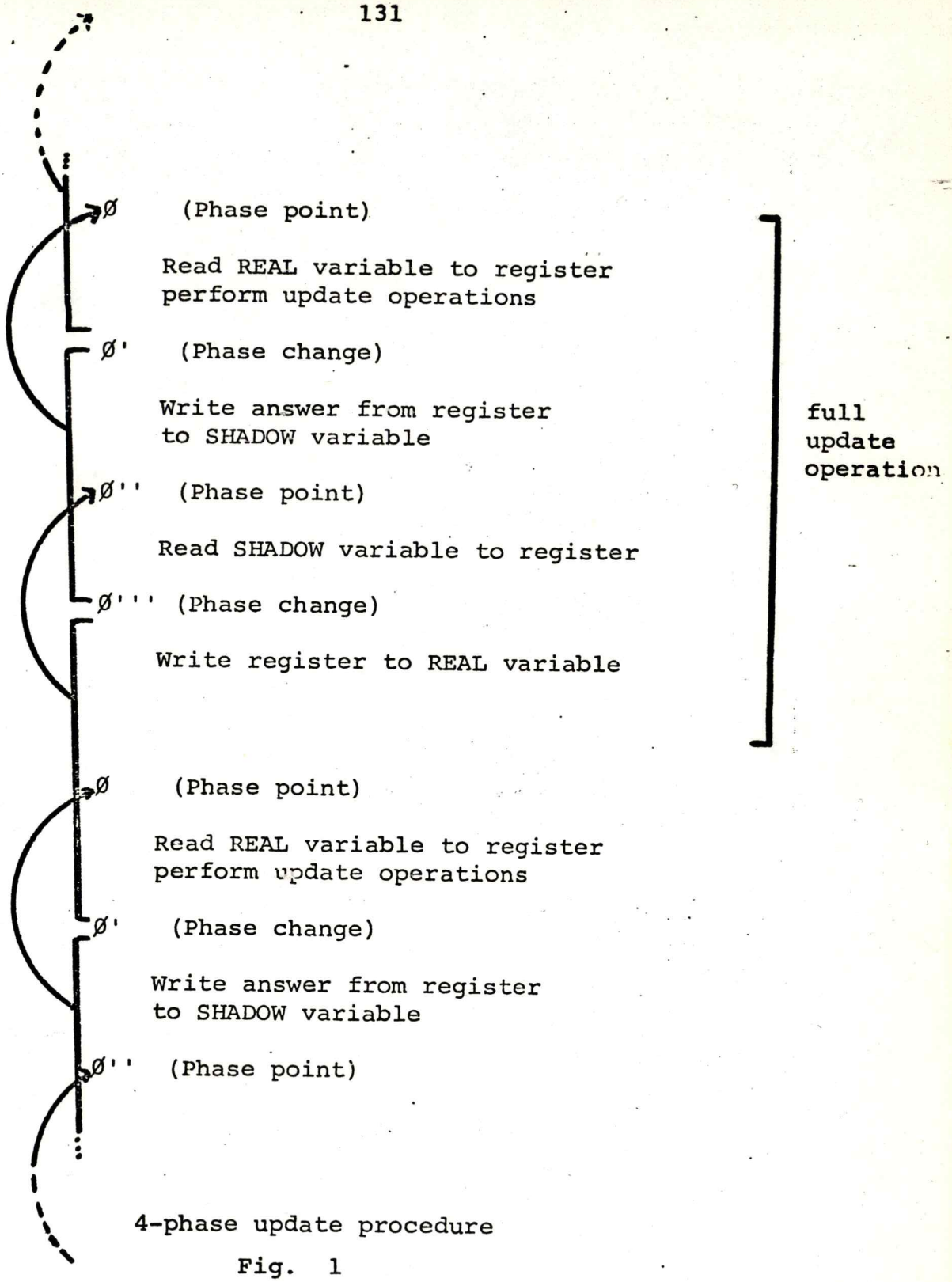
A DRO memory is more difficult to deal with since a variable can be destroyed during a read access. Clearly, since an input to a phase can be destroyed by reading it, merely repeating the phase, as in the NDRO case, will do no good. The most straightforward solution is to triple store all permanent variables and use triply stored shadow variables. The proper value of the variable can always be determined if one of its copies was

destroyed during the course of a deliberate access. As in the NDRO case, the shadow variables can be released after the update procedure is over.

One method of recovery from a detected error requires a repair routine (which has knowledge of the locations of all triple variables) to make these variables self-consistent before the affected phase can be reexecuted. The variable location tables required by the repair routine consume memory space and tend to reduce programming flexibility. An alternative approach dispenses with the repair routine but requires that a variable must be triply read and the copies checked for consistency each time that the variable is used. On the positive side, the triple read and check could be implemented in a microprogram. On the negative side, the throughput penalties incurred are severe.

An alternative scheme for DRO memories is a 4-phase update. This approach eliminates the need to triple store all variables, but the shadow variables cannot be released after an update, so in effect all variables are doubly stored. The principle of operation of a 4-phase update is the same as that of a 2-phase update and is derived from Rule 1. Since in a DRO memory a word can be destroyed during a read or a write operation, both copies of a variable cannot be accessed during a single phase. That is, the real variable cannot be read and the shadow written during the same phase; these two operations must occur on different sides of a phase point. (The computed data resides in a central register while the phase change is being made.) In the 4-phase approach, rollback does not go to the beginning of the phase in which the error occurred; it goes to the beginning of the nearest "read" phase prior to the one in which the error occurred. This is shown in Fig. 1.

The pros and cons of the 2-phase with triple store versus 4-phase techniques for DRO memories are numerous and will not be expounded upon further. Suffice it to say that both methods are fairly cumbersome to implement, and may be the cause of more problems than they cure.



4-phase update procedure

Fig. 1

Finally, consider the data triple store for a variable which is updated by incremental external inputs. The current value of the variable resides triply in main memory. The update can be accomplished by:

1. Reading the variable.
2. Checking for consistency.
3. Adding the increment to the "old" value of the variable.
4. Triply storing the new value.

If the computer is zapped during this operation, the maximum data loss is the increment to the variable; either the old or the incremented value can be properly recovered. If the increment exists in a hard form at the subsystem, then it can be recovered. Recovery cannot be guaranteed for subsystem quantities which are not hardened, such as PIPA counts.

4.3.2.3.3 Controlled Rollback Summary

In a situation where one word of memory has been lost during a read or write of that word, the ability to execute a controlled rollback recovery implies:

1) Triple store of phase information.
2) If the memory is NDRO (errors can occur only during a write access), a 2-phase update is needed. The shadow variable is needed only during the update and can then be released.
3) If the memory is DRO (errors can occur during a read or a write access), a triple store of variables is needed in addition to the 2-phase update. The triple stored shadow variable is needed only during the update.
4) An alternate scheme for DRO memories uses a 4-phase update without triple store of variables. The shadow variable is permanent, however, so this amounts to a double store of all variables.

4.3.2.4 Interference with Memory Addressing

If the transient detector is not fast enough, a situation arises in which the memory logic may reference the wrong word of memory (i.e., a word other than the one being addressed by the current instruction). This behavior impacts the ability of the aforementioned recovery schemes to guarantee a controlled recovery. DRO memories exhibit a more severe reaction to the possibility of incorrect addressing than do NDRO memories.

If memory is NDRO, then program recovery schemes are not impacted. Data recovery can be accomplished with a 2-phase update, but requires the triple store of permanent variables, since an incorrectly addressed write can destroy any singly stored variable in the computer. (The effects of an incorrectly addressed write are not restricted to variables associated with the current phase of computation.)

If the memory is DRO, a transient during the memory restore cycle can result in the loss of two words of memory. The originally selected word will fail to be rewritten and will become all zeros; and the contents of the memory buffer register will be ORed into some other unknown word.

Since two words may be bad, vertical parity techniques will not work for program reconstruction.

The 4-phase update is not adequate for data protection since the second affected location (the first being the one that was originally addressed) may be outside the current phase of computation. Although such a variable has two copies (the real and the shadow) there is no way to determine which one is really the correct value.

The 2-phase update with triple store fares better than the 4-phase method, but still has a weakness in that the second affected location may be in the same triple as the word originally addressed. This "unusual" case causes the triple to become invalid. The logical remedy is to store all permanent variables as 5-tuples (perhaps 4-tuples are adequate if it can be shown that the two erroneous words will never be equal), but such a remedy is terribly inefficient of time and memory.

4.3.2.5 Multiple Noise Bursts

Noise transients which are appropriately related in time can defeat certain of the controlled rollback schemes. Such a diabolical relationship exists when the second (or subsequent) transient occurs during the recovery from the previous transient.

If the transient can cause the writing of an incorrect location in memory, then no controlled rollback scheme is foolproof. A multiply stored variable which is not related to the current update procedure could be destroyed, one location at a time, by successive transients. When the time comes to use that variable, it would be impossible to recover the proper value. For example, the procedure to make a triply stored variable self-consistent involves a write of the bad copy. If addressing is affected during this write, bad data can be written into one of the good copies, thus invalidating the triple.

Even if erroneous addressing could be ruled out, DRO memories have another problem which stems from the fact that a word can be destroyed during a read access: If the first transient destroys a copy of the phase pointer, and the second transient destroys another copy of the phase pointer during the recovery attempt, all phase information will be lost. Adding two more copies of the phase pointer protects against a second transient. In general, 2n+1 copies will protect against n transients (1 transient during an update and 1 transient during each of n-1 recovery attempts). It can be shown by induction, however, that in the face of multiple transients, the validity of phase information can never be guaranteed, so the probability of controlled recovery with a DRO memory is strictly less than one.

4.3.3 Category 3—Central Registers are Lost but Main Memory is All Good.

This situation comes about if the computer has a noise detection mechanism, as described in Category 2, which prevents the initiation of a

new memory cycle when a potentially dangerous noise event is detected. In this case, however, the rise time of the noise is slow enough so that the current memory cycle is not affected. It is assumed that the noise level is sufficiently high to zap the registers but not the memory. Since the memory is never cycling during a critical noise event, DRO vs. NDRO is not a question.

Recovery is accomplished via a controlled rollback. The phase pointer can be singly stored, but in order to satisfy the first premise of controlled rollback (i.e. a phase does not destroy its own inputs) it is necessary to use the 2-phase updating technique described earlier. To see why this is so, consider the situation where some, but not all, of the inputs to a phase are updated and a glitch occurs. If the updates have been written directly into the variables instead of into shadow variables, then the input data to the phase is only partially updated and is therefore inconsistent. A self-consistent set of variables cannot be regenerated so a controlled rollback becomes impossible.

4.3.4 Category 4—Registers and Memory are not Affected by Noise, but there is a Power Loss to the Computer.

In the absence of a hardware scheme to circumvent the effects of a power loss, Category 2 or Category 3 information losses could result. Such problems are avoided by implementing a power fail detector which activates a hardware or software "hibernation" algorithm. This algorithm causes all volatile semiconductor storage to be written into dedicated memory locations before power is completely lost. It is assumed that the CPU is sufficiently shielded so that EMI transients that cause power loss do not affect the central registers as they are being stored away. It is also assumed that the power supply has sufficient storage to operate long enough to allow the registers to be stored away after the detection of a power fail. When power comes up again operation resumes at the point where it left off. Essentially all recent computers have a feature of this sort.

SKC-2000 ?
A + B + 64X + 256SP
+ IR + SR + M I/O IF.

The major problem is loss of knowledge of the state of the real world due to the time spent in hibernation. Mission time must be available somewhere that is not susceptible to power failure. Other real-world variables can probably be extrapolated in an orderly manner if the hibernation period has not been too long.

BIBLIOGRAPHY FOR PART II

1. NASA TM-X-64589, "Terrestrial Environment (Climatic) Criteria Guidelines for Use in Space Vehicle Development", 1971.

2. Paul, Fred. W., and Burrowbridge, Donald, "The Prevention of Electrical Breakdown in Spacecraft", NASA SP-208, NASA, Washington, D.C., 1969.

3. ARINC No. 413, Guidance for Aircraft Electrical Power Utilization and Transient Protection, 1 May 1967.

4. Military Specification, "Bonding, Electrical, and Lightning Protection for Aerospace Systems", MIL-B-5087B(ASG), 1964 and Amendment 2, 31 August 1970.

5. Military Specification, "Electromagnetic Interference Characteristics Requirements for Equipment", MIL-STD-461A, 1 August 1968.

6. "Accidents/Incidents 1965 to Date Involving Lightining Strikes - Sorted by Aircraft Type and Accident Class", USAF Directorate of Aerospace Safety, Norton AFB, Automated Data Stripout requested by Mr. R.B. Shanks, IGDSSE.

7. Appleman, H.S., "Lightning Hazard to Aircraft", Hq. Air Weather Service (MAC), USAF, Tech. Report 179, April 1971.

8. Gordon, W.F., "Lightning Environments", SCL-DR-69-40, Sandia Laboratories, Livermore, April 1969.

BIBLIOGRAPHY FOR PART II (con.)

9. Peterson, B.J., and Wood, W.R., "Measurements of Lightning Strikes to Aircraft", SC-M-67-549, January 1968.

10. "Lightning", AFWL Vulnerability News and Views", Vol. 31, 1965.

11. Fitzgerald, D.R., "USAF Flight Lightning Research", Lightning and Static Electricity Conference, Tech. Report AFAL-TR-68-290, Part II, Dec. 1968.

12. Brook, Holmes and Moore, "Lightning and Rockets: Some Implications of the Apollo 12 Lightning Event", Naval Research Reviews, April 1970.

13. NASA MSC-01540, "Analysis of Apollo 12 Lightning Incident", February 1970.

14. Lloyd, K.J., Plumer, J.A., Walko, L.C., "Measurements and Analysis of Lightning Induced Voltages in Aircraft Electrical Circuits", NASA CR-1744, February 1971.

15. Harrison, L.P., "Lightning Discharges to Aircraft and Associated Meteorlogical Conditions, "NACA Technical Note 1001, May 1946.

16. Perry, B.L., and Eng, C., "Lightning and Static Hazards Relative to Airworthiness", Lighting and Static Electricity Conference, Sponsored by AFAL and SAE, December 1970.

17. Ricketts, L.W., "Fundamentals of Nuclear Hardening of Electronic Equipment", John Wiley & Sons, 1972.

18. Hall, E.C., "MIT's Role in Project Apollo", Vol. III, "Computer Subsystem," MIT C.S. Draper Laboratory, R-700, August 1972.

19. Axtell, J.C., and Ockberg, T.C., "An Electrostatic Charge Phenomenon Associated with Minuteman Missile Flights", Lightning and Static Electricity Conference, Tech. Report AFAL-TR-68-290, Part II, December 1968.

20. "DNA EMP (Electromagnetic Pulse) Handbook" (C), Vol. 1, Design Principles, Defense Nuclear Agency, Nov. 1971.

21. "DNA EMP (Electromagnetic Pulse) Handbook" (C), Vol. 2, Analysis and Testing, Defense Nuclear Agency, Nov. 1971.

22. Perry, B.L., "British Researches and Protective Recommendations of the British Air Registration Board", Lightning and Static Electricity Conference, Tech. Report AFAL-TR-68-290, Part II, December 1968.

23. "Poseidon Guidance Computer Special Tests Performed at G.E. High Voltage Laboratory" (S), R. Roussin, Feb. 1970.

24. "Current Insertion Tests, Product Assurance and Surveillance of the Poseidon Guidance Electronics Assembly, MK 3 Mod 0", Code Ident. 10001 NAVORD OD 45212, 1971.

25. "OD 45212 (Addendum) Current Insertion Tests Product Assurance and Surveillance of the Poseidon Guidance Electronics Assembly MK 3 Mod 0", (S), Strategic Sys. Proj., June 1971.

26. "STS Survivability Characteristics", 1 March 1972, (S).

BIBLIOGRAPHY FOR PART II (con.)

27. Griggs, K.M., and Schwartz, G., "The DCA Computer", MIT C.S. Draper Laboratory, E-2590, December 1970.

A Division of Massachusetts Institute of Technology
The Charles Stark Draper Laboratory
68 Albany Street, Cambridge, Massachusetts 02139 Telephone (617) 258-

PART III

COMPUTER SYSTEM RELIABILITY

Robert J. Filene

1. INTRODUCTION AND CONCLUSIONS

1.1 INTRODUCTION

This memorandum investigates the effect that the Shuttle requirements for probability of mission success and crew safety have upon the GN&C computer system. These requirements have been recently stated as 0.9 probability of mission success and 0.999 probability of crew safety, where these requirements are for the entire orbiter.* The crew safety requirement is interpreted to cover the whole mission, that is, at the time of launch the probability of the safe return of the crew is .999 or better. To meet these goals, each of the systems effectively in series for calculating reliability must exceed these probabilities of success. For purposes of this memo, we have arbitrarily chosen 0.99 as the computer system contribution to mission success and 0.9999 as the computer system contribution to crew safety.

The model used in this memo for the computer system is N computers where one of these is initially designated as prime and the other (N-1) are spares.

Failure of a computer is detected with probability C, the computer's error coverage, whether the failed unit is currently prime or a spare. (Coverage usually includes the probability of detection and recovery. To highlight the detection problem, however, the recovery mechanism is assumed to be perfect for this memo.) We also assume that the failure of

---

*Current Shuttle contractor efforts are addressing higher reliability requirements of about 0.98 for mission success and 0.9998 for crew safety. The impact of these requirements upon the computer system will be addressed in a later memorandum.

a spare is detected as rapidly as a failure of the prime. If in fact some spares are off to conserve power, this assumption would not be valid.

If the prime computer fails, the new prime computer is selected from those spares which have not reported themselves as failed. It is possible, therefore, that a spare, which has already failed without detection, may be designated prime. It is assumed that the computer system fails if all computers fail, if the prime fails without detection, or if a spare which failed without detection is then designated prime.

The Shuttle mission is assumed to be 168 hours (1 week) in duration. Two mission models are considered here: (1) The no abort model. Completion of the mission is attempted regardless of the number of computer failures. (2) The early return model. The mission is to be aborted after a predetermined number of detected computer failures. A return to earth is then initiated. This return is assumed to take 3.36 hours. For both models, we assume that failure of the computer system results in failure of the mission and loss of the crew. Additionally, for Model 2 only (possible early return), the mission fails if the predetermined number of acceptable computer failures is reached, since the mission is then aborted. Section 2 presents detailed reliability equations for these mission models. Section 3 simplifies some of these equations by identifying the major terms. These simpler forms help one to see the effect that varying parameters such as computer MTBF and coverage have upon the results. Section 4 outlines some operational aspects of the mission not currently included in the computer system reliability model.

TABLE A

Summary of computer MTBF/COVERAGE combinations which make the computer system's contribution to crew safety at least 0.9999 and to mission success at least 0.99.

NUMBER OF COMPUTERS	ABORT MISSION AFTER K DETECTED FAILURES	REQUIRED COVERAGE		
		MTBF OF EACH COMPUTER = 2,500 HR.	MTBF OF EACH COMPUTER = 5,000 HR.	MTBF OF EACH COMPUTER = 10,000 HR.
3	NEVER*	NOTE 1	.9982	.9950
	K = 2	NOTE 2	.9972	.9942
	K = 1	NOTE 2	NOTE 2	NOTE 2
4	NEVER*	.9989	.9972	.9942
	K = 3	.9986	.9971	.9941
	K = 2	NOTE 2	.9971	.9941
	K = 1	NOTE 2	NOTE 2	NOTE 2

*MISSION RULE IS TO NEVER ABORT.

NOTE 1: COMPUTER MTBF IS TOO LOW TO MEET CREW SAFETY REQUIREMENT, REGARDLESS OF COVERAGE.

NOTE 2: COMPUTER MTBF IS TOO LOW TO MEET MISSION SUCCESS REQUIREMENT, REGARDLESS OF COVERAGE.

MISSION TIME = 168 HOURS (1 WEEK)

RETURN TIME = 3.36 HOURS

1.2 CONCLUSIONS

Table A summarizes the major conclusions of the study based upon the exact equations of Section 2, not the simplifications of Section 3. The following points can be seen from this table:

Conclusions related to Model 1.

With mission model 1 (no aborts) the reliability requirements can be met with three computers each having MTBF of 5000 hours and coverage of 0.9982, or with four computers each having MTBF of 2500 hours and coverage of 0.9939.

With mission model 1 (no aborts) and three computers each having MTBF less than 2500 hours, the reliability requirements can <u>not</u> be met, regardless of coverage.

Conclusions related to Model 2.

Imposing a mission rule to abort after a predetermined number of detected computer failures provides little relief for the high coverage required for Model 1 to meet the reliability demands. Section 2 discusses the reason for this.

With three computers, and a mission rule to abort after the 2nd detected computer failure, the requirements can be met with MTBF of 5000 hours and coverage of 0.9972 for each computer.

Table B illustrates the following conclusions. With a given coverage and computer MTBF in the range 2500 hours to 10,000 hours, imposing a mission rule to abort after a predetermined number of failures reduces the probability of mission success significantly. The probability of crew safety shows only small improvement unless coverage is very near to 1.

	C = .996		C = 1	
	NO ABORT	ABORT AFTER 2	NO ABORT	ABORT AFTER 2
MISSION SUCCESS	.994	.988	.9997	.988
CREW SAFETY	.9994	.9997	.9997	.99998

TABLE B. 3 Computers. Each has MTBF of 2500 hours. Never abort vs. abort after 2nd detected failure.

2. DETAILED RELIABILITY ANALYSIS OF COMPUTER SYSTEM

The major conclusions of this section are listed in Table A. The text describes the equations used to reach these conclusions.

2.1 MODEL 1 - MISSION IS NEVER ABORTED

For this case, completion of the mission is attempted regardless of how many computer failures have been detected. Therefore, the probability of mission failure and the probability of crew loss are both equal to the probability that the computer system fails before the mission is completed. This can be expressed as:

$$F_{system} = F_1(1-C_1) + F_1C_1F_2(1-C_2) + \ldots \quad (1)$$

$$+ (F_1F_2\ldots F_{N-2})(C_1C_2\ldots C_{N-2})F_{N-1}(1-C_{N-1})$$

$$+ (F_1F_2\ldots F_{N-1})(C_1C_2\ldots C_{N-1})F_N$$

where N = the number of computers

F_i = the probability that computer i fails

C_i = the coverage of computer i.

If all F's are equal and all C's are equal then:

$$F_{system} = F(1-C) + F^2C(1-C) + F^3C^2(1-C) \quad (2)$$

$$+ \ldots + F^{N-1}C^{N-2}(1-C) + F^NC^{N-1}$$

$$= F^NC^{N-1} + (1-C)\sum_{K=0}^{N-2} F^{K+1}C^K$$

Figure 1 plots probability of failure vs. coverage for both the three and four computer cases. Table C presents the same data for the one, two, three, and four computer cases. Equation (2) was used in all cases. Throughout this memo, the probability that a computer fails before time t, is assumed to be:

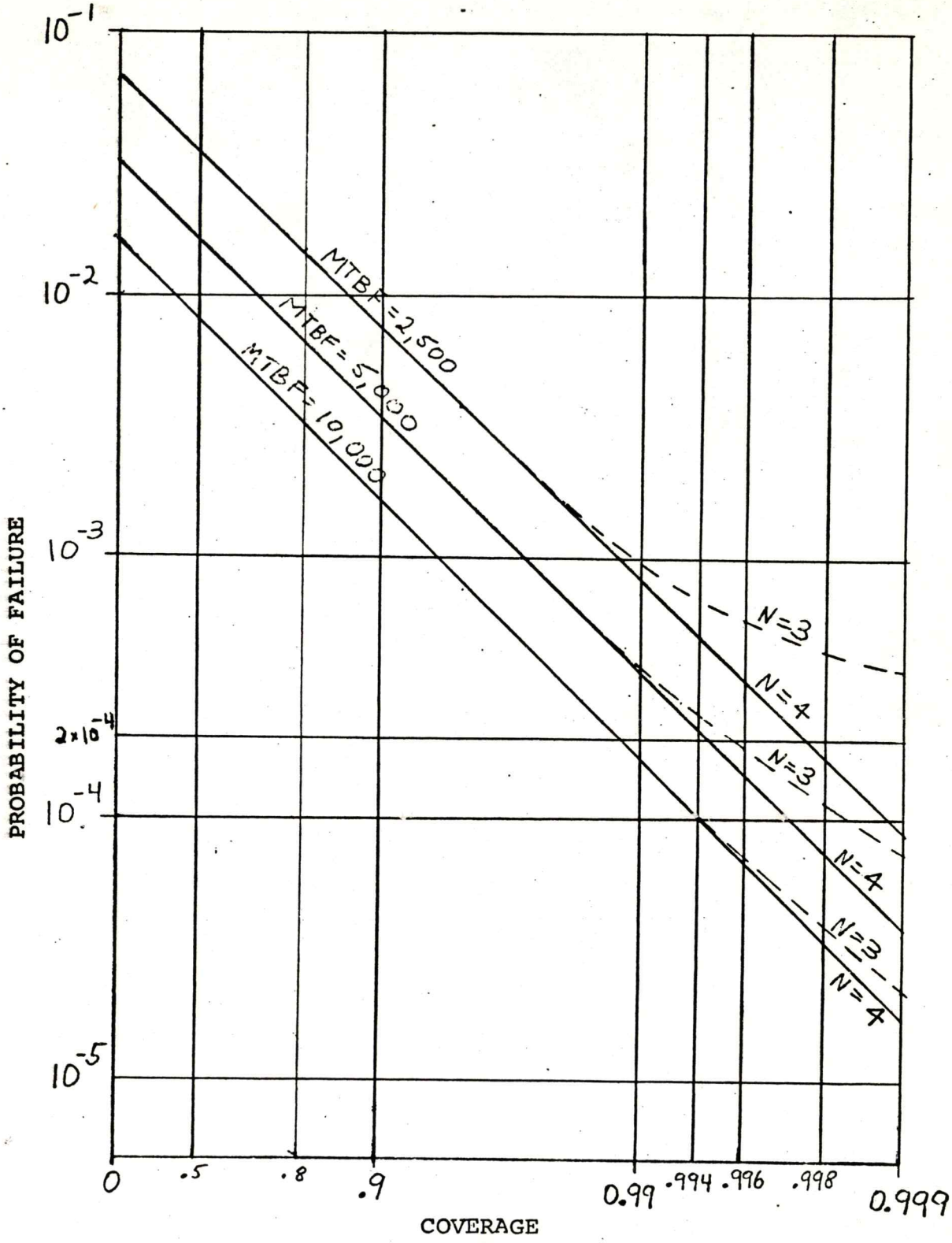
$$F = 1 - e^{\frac{-t}{MTBF}} \qquad (3)$$

For the three MTBF's considered, Table A shows the coverage required for the three and four computer cases to meet the reliability demands. These conclusions are shown in the rows marked "never" in Table A. Note that, with three computers each having 2500 hour MTBF, the probability of crew survival is less than the required .9999, even with perfect coverage.

2.2 MODEL 2 - MISSION MAY BE ABORTED

For this case, the mission may be aborted and an early return to earth initiated after a certain number of computer failures have been detected. It would seem that this early return possibility would relieve the severe coverage requirements needed for Model 1 to make the probability of crew success, R_C, .9999 or greater. In fact, the relief is small. Also, the early abort system must meet the probability of mission success, R_M, of .99 assumed for this memo. Imposing a mission rule to abort after a predetermined number of detected failures, reduces the probability of mission success, R_M.

Figure 2 plots probability of mission failure and crew loss vs. coverage for two cases: three computers with an abort after the 2nd detected failure and; four computers with an abort after the 2nd failure. Tables D and E correspond to Figure 2. Table A summarizes the coverage required to meet the reliability demands for the three MTBF's considered. Note that,



Probability of computer system failure ($=F_M=F_C$).
N computers.
Mission time is one week.
No early aborts.

Figure 1

MTBF= 2500. HOURS

	N=1	N=2	N=3	N=4
C= 0.000	6.50E-02	6.50E-02	6.50E-02	6.50E-02
C= 0.800	6.50E-02	1.64E-02	1.38E-02	1.37E-02
C= 0.900	6.50E-02	1.03E-02	7.10E-03	6.91E-03
C= 0.950	6.50E-02	7.26E-03	3.70E-03	3.48E-03
C= 0.990	6.50E-02	4.83E-03	9.61E-04	7.12E-04
C= 0.994	6.50E-02	4.59E-03	6.86E-04	4.34E-04
C= 0.996	6.50E-02	4.47E-03	5.49E-04	2.96E-04
C= 0.998	6.50E-02	4.35E-03	4.12E-04	1.57E-04
C= 0.999	6.50E-02	4.28E-03	3.43E-04	8.73E-05
C= 1.000	6.50E-02	4.22E-03	2.75E-04	1.78E-05

$< 10^{-4}$

MTBF= 5000. HOURS

	N=1	N=2	N=3	N=4
C= 0.000	3.30E-02	3.30E-02	3.30E-02	3.30E-02
C= 0.800	3.30E-02	7.48E-03	6.81E-03	6.79E-03
C= 0.900	3.30E-02	4.20E-03	3.43E-03	3.41E-03
C= 0.950	3.30E-02	2.69E-03	1.74E-03	1.71E-03
C= 0.990	3.30E-02	1.41E-03	3.77E-04	3.43E-04
C= 0.994	3.30E-02	1.28E-03	2.40E-04	2.06E-04
C= 0.996	3.30E-02	1.22E-03	1.72E-04	1.38E-04
C= 0.998	3.30E-02	1.16E-03	1.04E-04	6.95E-05
C= 0.999	3.30E-02	1.12E-03	7.01E-05	3.54E-05
C= 1.000	3.30E-02	1.09E-03	3.61E-05	1.19E-06

$< 10^{-4}$

MTBF= 10000. HOURS

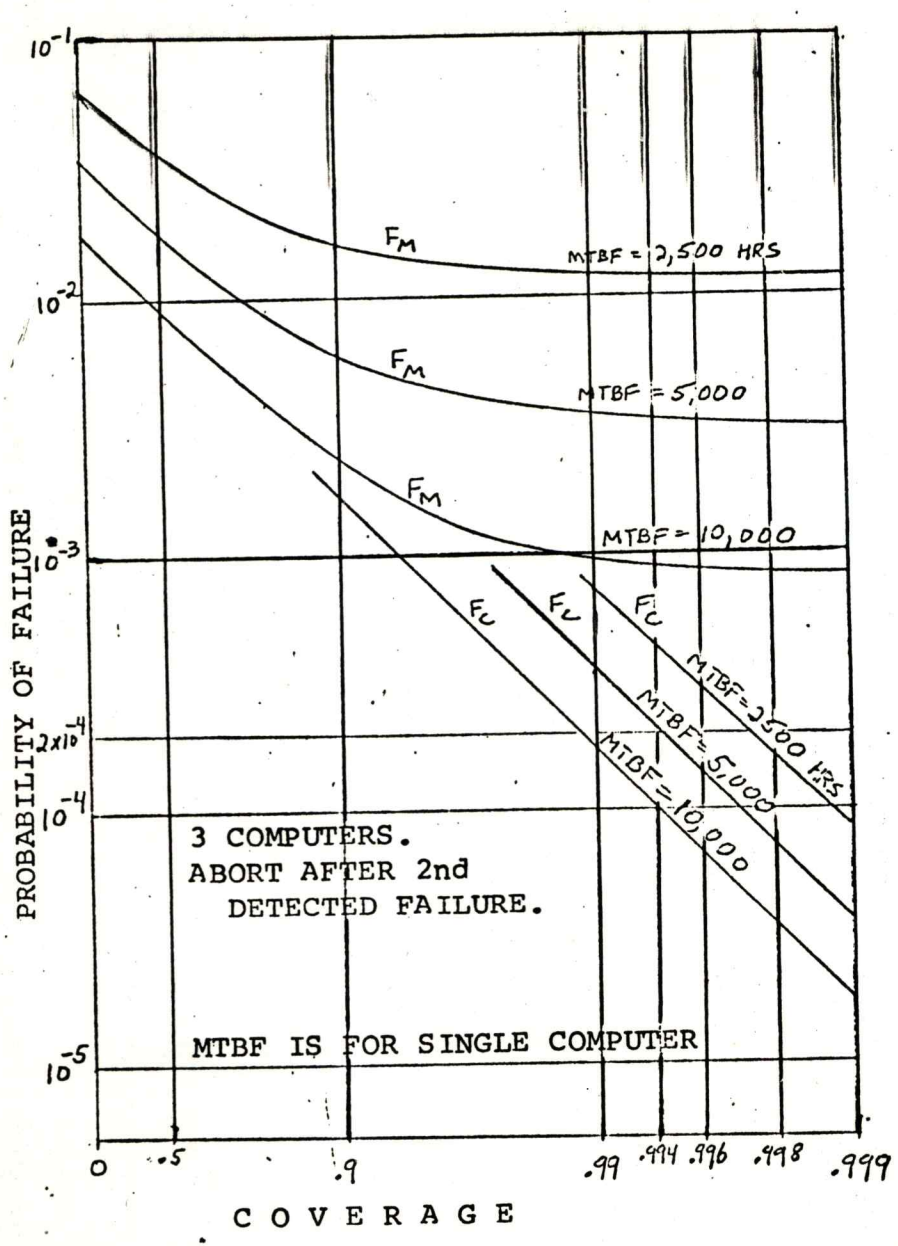
	N=1	N=2	N=3	N=4
C= 0.000	1.67E-02	1.67E-02	1.67E-02	1.67E-02
C= 0.800	1.67E-02	3.55E-03	3.38E-03	3.38E-03
C= 0.900	1.67E-02	1.92E-03	1.69E-03	1.69E-03
C= 0.950	1.67E-02	1.10E-03	8.50E-04	8.46E-04
C= 0.990	1.67E-02	4.41E-04	1.74E-04	1.69E-04
C= 0.994	1.67E-02	3.76E-04	1.06E-04	1.02E-04
C= 0.996	1.67E-02	3.43E-04	7.23E-05	6.78E-05
C= 0.998	1.67E-02	3.10E-04	3.85E-05	3.40E-05
C= 0.999	1.67E-02	2.94E-04	2.16E-05	1.70E-05
C= 1.000	1.67E-02	2.78E-04	4.62E-06	7.70E-08

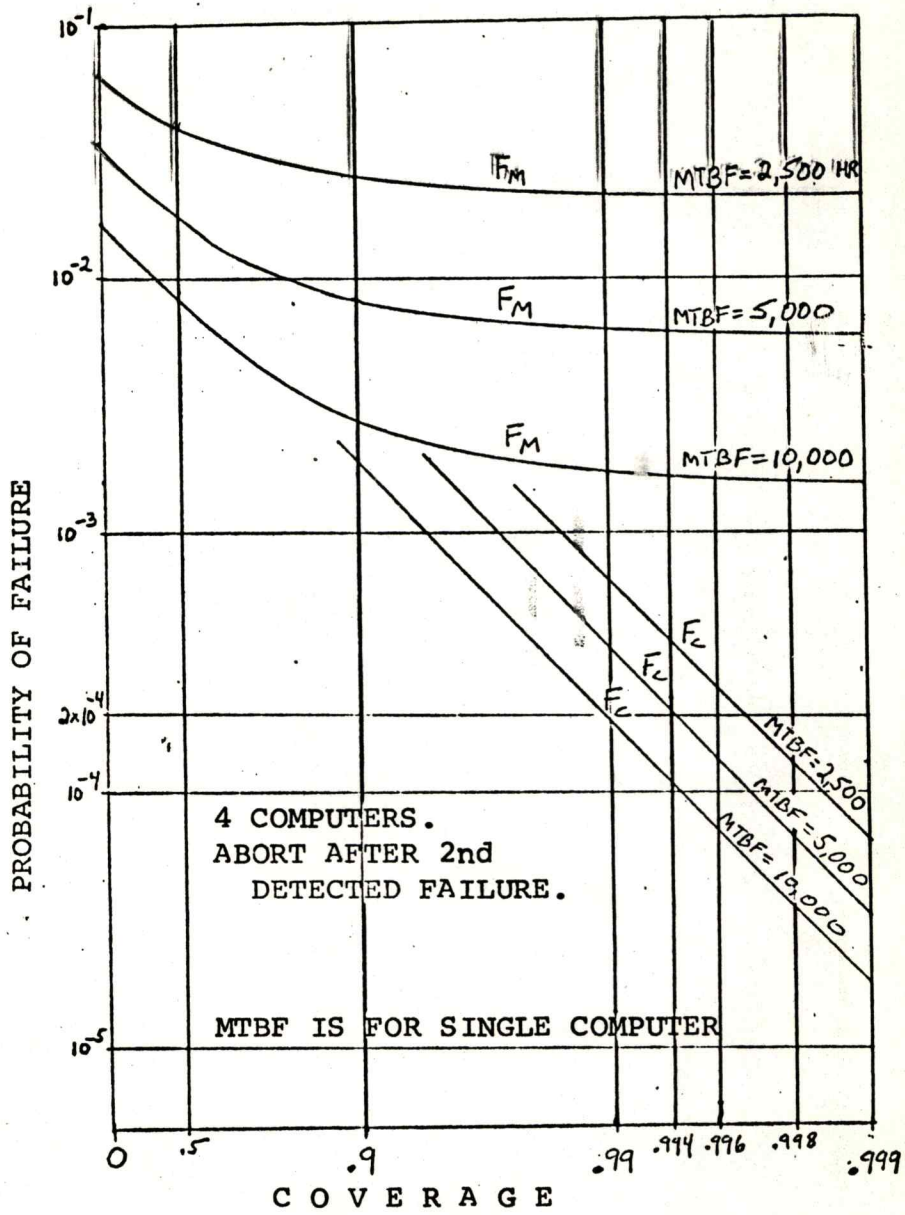
$< 10^{-4}$

E-ij means x 10^{-ij}

Probability of computer system failure ($=F_M=F_C$).
N computers.
Mission time=one week.
No early aborts.

Table C





F_M=Probability of Mission Failure
F_C^M=Probability of Crew Loss

Mission Time=1 Week
Return Time=3.36 Hours

Figure 2. Probability of Failure with Early Return Model.

MTBF= 2500. HOURS

	F_M	F_C
C= 0.0000	6.499182E-02	6.499182E-02
C= 0.5000	3.646907E-02	3.355589E-02
C= 0.8000	2.114217E-02	1.368442E-02
C= 0.9000	1.633099E-02	6.892277E-03
C= 0.9900	1.212824E-02	7.073934E-04
C= 0.9940	1.194425E-02	4.309275E-04
C= 0.9960	1.185234E-02	2.926440E-04
C= 0.9980	1.176050E-02	1.543269E-04
C= 0.9990	1.171460E-02	8.515574E-05
C= 0.9999	1.167330E-02	2.289448E-05
C= 1.0000	1.166871E-02	1.597614E-05

$< 10^{-4}$

MTBF= 5000. HOURS

	F_M	F_C
C= 0.0000	3.304179E-02	3.304179E-02
C= 0.5000	1.756377E-02	1.679436E-02
C= 0.8000	8.754079E-03	6.784390E-03
C= 0.9000	5.897035E-03	3.404147E-03
C= 0.9900	3.359689E-03	3.432945E-04
C= 0.9940	3.247666E-03	2.068469E-04
C= 0.9960	3.191678E-03	1.386101E-04
C= 0.9980	3.135706E-03	7.036450E-05
C= 0.9990	3.107726E-03	3.623844E-05
C= 0.9999	3.082547E-03	5.523124E-06
C= 1.0000	3.079750E-03	2.110202E-06

$< 10^{-4}$

$< 10^{-2}$

MTBF= 10000. HOURS

	F_M	F_C
C= 0.0000	1.665967E-02	1.665967E-02
C= 0.5000	8.597026E-03	8.399287E-03
C= 0.8000	3.882725E-03	3.376514E-03
C= 0.9000	2.331839E-03	1.691165E-03
C= 0.9900	9.448254E-04	1.696102E-04
C= 0.9940	8.833735E-04	1.018812E-04
C= 0.9960	8.526537E-04	6.801343E-05
C= 0.9980	8.219380E-04	3.414342E-05
C= 0.9990	8.065817E-04	1.720758E-05
C= 0.9999	7.927619E-04	1.964855E-06
C= 1.0000	7.912265E-04	2.711913E-07

$< 10^{-4}$

$< 10^{-2}$

E-ij means x 10^{-ij}

3 computers. Abort after 2nd detected failure.
Probability of mission failure (F_M) and
probability of crew loss (F_C) are shown.
Mission time = one week.
Return time = 3.36 hours.

Table D

MTBF= 2500. HOURS

	F_M	F_C
C= 0.0000	6.499182E-02	6.499182E-02
C= 0.5000	3.922860E-02	3.352158E-02
C= 0.8000	2.806982E-02	1.364317E-02
C= 0.9000	2.504112E-02	6.859718E-03
C= 0.9900	2.260468E-02	6.893974E-04
C= 0.9940	2.250270E-02	4.137446E-04
C= 0.9960	2.245192E-02	2.758732E-04
C= 0.9980	2.240126E-02	1.379720E-04
C= 0.9990	2.237599E-02	6.901015E-05
C= 0.9999	2.235326E-02	6.938097E-06
C= 1.0000	2.235074E-02	4.082817E-08

$< 10^{-4}$

MTBF= 5000. HOURS

	F_M	F_C
C= 0.0000	3.304179E-02	3.304179E-02
C= 0.5000	1.831261E-02	1.678985E-02
C= 0.8000	1.065255E-02	6.778964E-03
C= 0.9000	8.291954E-03	3.399859E-03
C= 0.9900	6.249015E-03	3.409168E-04
C= 0.9940	6.160005E-03	2.045759E-04
C= 0.9960	6.115557E-03	1.363931E-04
C= 0.9980	6.071147E-03	6.820202E-05
C= 0.9990	6.048956E-03	3.410342E-05
C= 0.9999	6.028992E-03	3.412914E-06
C= 1.0000	6.026775E-03	2.755530E-09

$< 10^{-4}$

$< 10^{-2}$

MTBF= 10000. HOURS

	F_M	F_C
C= 0.0000	1.665967E-02	1.665967E-02
C= 0.5000	8.792103E-03	8.398709E-03
C= 0.8000	4.379705E-03	3.375818E-03
C= 0.9000	2.959809E-03	1.690615E-03
C= 0.9900	1.703559E-03	1.693045E-04
C= 0.9940	1.648201E-03	1.015893E-04
C= 0.9960	1.620537E-03	6.772839E-05
C= 0.9980	1.592883E-03	3.386536E-05
C= 0.9990	1.579060E-03	1.693304E-05
C= 0.9999	1.566621E-03	1.693489E-06
C= 1.0000	1.565239E-03	1.790189E-10

$< 10^{-4}$

$< 10^{-2}$

E-ij means x 10^{-ij}

4 computers. Abort after 2nd detected failure.
Probability of mission failure (F_M) and probability of crew loss (F_C) are shown.
Mission time = one week.
Return time = 3.36 hours.

Table E

with 2500 hour MTBF, three or four computers, and an abort after the second detected failure, the mission success requirement (0.99) is not met even with perfect coverage.

Comparing Figure 1 with Figure 2, we see that the probability of mission failure is significantly increased by imposing a mission rule to abort after a predetermined number of detected failures. This is because failures of spare computers do not fail the mission in the no abort case if the primary succeeds, but the failure of a predetermined number of spares causes an abort and therefore mission failure in the early abort case. Similarly, comparing the two cases in Figure 2, we see that keeping the abort rule fixed (e.g. after 2 failures) and increasing the total number of computers (e.g. from 3 to 4) increases the probability of mission failure. This is because the added computers increase the probability that the abort will be initiated, thus failing the mission.

Again comparing Figures 1 and 2, we see that the probability of crew loss is not significantly reduced by aborting early. This is because, for coverage less than about .997, the probability of crew loss is dominated by the probability that the initially prime computer fails, this is not detected, and the remaining computers have not failed. This situation is assumed to result in the loss of the crew and is unaffected by aborting early. With perfect coverage, however, more than an order of magnitude reduction in the probability of crew loss can be achieved by aborting early. This can be seen by comparing Table D and E with Table C for coverage equal to 1.

The remainder of this section describes the reliability equations used for the early abort case. The author is grateful to Professor Albert Hopkins of M.I.T. and the Digital Development Group for his help in formulating an approach to the problem, and to William Daly of the Digital Developement Group for his help in deriving the generalized form of these equations.

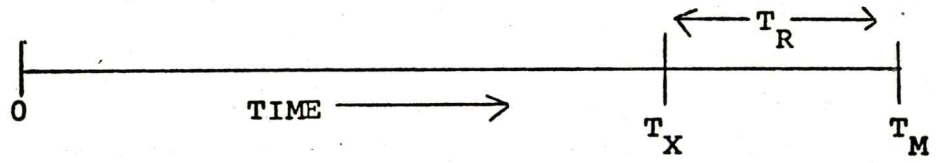
The following notation will be used:

T_M = the normal mission time, from launch to landing.

T_R = the time required to return to earth at the end of a normal mission or after an abort is initiated.

T_X = $T_M - T_R$, The time at which a normal return begins. After T_X there are no aborts since the return is already in progress.

The mission time line then has the form:



C = Coverage, the probability that a computer failure is detected. (To highlight the detection problem, recovery following detection is assumed to be perfect.)

$\overline{C}$ = 1 - C

R_M = the probability that the normal mission is completed.

$F_M = 1 - R_M$

R_C = the probability that the crew returns safely.

$F_C = 1 - R_C$

R_A = the probability that the mission is aborted before T_X and the return phase is successful.

$F_A = 1 - R_A$

R_X = the probability that a single computer succeeds till $T_X = e^{\frac{-T_X}{MTBF}}$

$F_X = 1 - R_X$

R_R = the probability that a single computer succeeds for the interval $T_R = e^{\frac{-T_R}{MTBF}}$

$F_R = 1 - R_R$

S_i = the probability that a system of i computers succeeds for the interval T_R.

$$= R_R \sum_{K=0}^{i-1} C^K F_R^K$$

The numerical results in this memo assume T_M is one week (168 hours) and T_R is 3.36 hours.

Figure 3 shows the possible computer system events which can occur before T_X, and their implications on mission success and crew safety. The mission has a possibility of succeeding only if the computer system succeeds until T_X and an abort is not intiated before T_X; this is Case 1. Case 2 covers the situations where the prime computer fails without detection before T_X, or the prime fails with detection but the spare which is then designated to take over as prime had failed earlier without detection. In either situation, the computer system fails. Case 3 covers the situations where an abort is initiated before T_X, and when the abort is initiated, either the prime computer is healthy or a healthy spare is designated to take over as prime. If Case 3 or 1 occurs, the crew may return safely. Case 4 covers the situations where an abort is initiated before T_X and, when the abort is initiated, a failed spare (which has not detected its failure) is selected to take over as prime. This results in loss of the computer system, by definition.

R_M is then the probability that Case 1 occurs and the computer system succeeds from T_X till T_M. R_C is the sum of R_M and the probability that Case 3 occurs followed by a successful return:

$$R_C = R_M + R_A \tag{4}$$

The above discussion is intended to explain the general form of the equations that follow, but not to justify them in detail.

POSSIBLE COMPUTER SYSTEM EVENTS OCCURRING BEFORE T_X.

NO ABORT	ABORT BEFORE T_X
COMPUTER SYSTEM SUCCEEDS TILL T_X ①	A HEALTHY COMPUTER IS PRIME OR IS SELECTED AS PRIME WHEN ABORT INITIATED ③
UNDETECTED FAILURE OF PRIME BEFORE T_X OR FAILED SPARE SELECTED AS PRIME BEFORE T_X ②	A FAILED COMPUTER SELECTED AS PRIME WHEN ABORT INITIATED ④

1: MISSION SUCCEEDS TILL T_X.
CREW SAFE TILL T_X.

2 and 4: MISSION FAILS.
LOSS OF CREW.

3: MISSION FAILS.
CREW SAFE TILL ABORT INITIATED.

FIGURE 3

For a mission model where an abort is initiated after the i^{th} detected computer failure, R_M, R_C, and R_A will be denoted as R_{Mi}, R_{Ci}, and R_{Ai}. The equations for the three computer case are:

$$R_{M1} = R_X^3 S_3 + R_X^2 F_X \bar{C} (S_1 + S_2) + R_X F_X^2 \bar{C}^2 S_1$$

$$R_{M2} = R_{M1} + 3R_X^2 F_X C S_2 + 3R_X F_X^2 C \bar{C} S_1$$

$$R_{M3} = R_{M2} + 3R_X F_X^2 C^2 S_1$$

(R_{M3} means an early abort is never initiated and therefore R_{M3} can also be found by solving $(1 - F_{SYSTEM})$ in eq.(2) with N = 3.)

$$R_{A1} = (1 - R_X^3) CS_2 + 1/2 (3F_X^2 R_X + F_X^3) \bar{C}CS_1$$

$$R_{A2} = (3F_X^2 R_X + F_X^3) C^2 S_1$$

$$R_{C1} = R_{M1} + R_{A1}$$

$$R_{C2} = R_{M2} + R_{A2}$$

$$R_{C3} = R_{M3}$$

(R_{C3} means an early abort is never initiated)

For a mission model where an abort is initiated after the i^{th} detected computer failure, R_M, R_C, and R_A will be denoted by R_{Mi}, R_{Ci}, and R_{Ai}. The equations for the four computer case are:

$$R_{M1} = R_X^4 S_4 + R_X^3 F_X \bar{C}(S_1 + S_2 + S_3) + R_X^2 F_X^2 \bar{C}^2 (2S_1 + S_2) + R_X F_X^3 \bar{C}^3 S_1$$

$$R_{M2} = R_{M1} + 4R_X^3 F_X C S_3 + 4R_X^2 F_X^2 C \bar{C} (S_1 + S_2) + 4R_X F_X^3 C \bar{C}^2 S_1$$

$$R_{M3} = R_{M2} + 6R_X^2 F_X^2 C^2 S_2 + 6R_X F_X^3 C^2 \bar{C} S_1$$

$$R_{M4} = R_{M3} + 4R_X F_X^3 C^3 S_1$$

(R_{M4} means an abort is never initiated and therefore R_{M4} can also be found by solving $(1 - F_{SYSTEM})$ in eq. (2) with N = 4.)

$$R_{A1} = (1-R_X^4) C S_3 + \frac{\bar{C}C}{3} (6R_X^2 F_X^2 + 4R_X F_X^3 + F_X^4) (S_1 + S_2) + \frac{C\bar{C}^2}{3} (4R_X F_X^3 + F_X^4)$$

$$R_{A2} = C^2 (6R_X^2 F_X^2 + 4R_X F_X^3 + F_X^4) S_2 + C^2 \bar{C} (4R_X F_X^3 + F_X^4) S_1$$

$$R_{A3} = C^3 (4R_X F_X^3 + F_X^4) S_1$$

$$R_{C1} = R_{M1} + R_{A1}$$

$$R_{C2} = R_{M2} + R_{A2}$$

$$R_{C3} = R_{M3} + R_{A3}$$

$$R_{C4} = R_{M4}$$

(R_{C4} means an abort is never initiated)

3. SIMPLIFICATION OF THE RELIABILITY EQUATIONS

This section identifies the dominant terms for the no abort case discussed in Section 2.1 and for one of the abort cases discussed in Section 2.2, namely, four computers with an abort after the second detected failure.

3.1 MODEL 1 - MISSION IS NEVER ABORTED.

For this case, completion of the mission is attempted regardless of how many computer failures have been detected. Therefore, the probability of mission failure and the probability of crew loss are both equal to the probability that the computer system fails before the mission completes. This is given in equation (1). For coverage less than one, each of the summed terms in eq. (1) is non-negative, so by dropping all but the first term we have a lower bound on the probability of failure:

$$F_{system} = F_C = F_M > F_1(1-C_1) \qquad (5)$$

where F_C = probability of crew loss

F_M = probability of mission failure.

Setting F_C and F_M to required value (1×10^{-4}), obtaining F_1 from equation (3) with t = 168 hours, and solving equation (5) for C_1, identifies the minimum coverage required to make the lower bound on F_C and F_M less than 1×10^{-4}:

$C_1 > 0.9934$ for MTBF = 2500 hrs.

$C_1 > 0.9969$ for MTBF = 5000 hrs.

$C_1 > 0.9939$ for MTBF = 10,000 hrs.

Note that providing coverage higher than the values shown does not insure that F_C and F_M are lower than 10^{-4}. However with coverage lower than these values, F_M and F_C are higher than 10^{-4}.

Since equation (5) is only a function of F_1 and C_1, these bounding results are independent of the number of spare computers provided. The results are therefore also unchanged if one assumes that the spares are powered down until needed and that the dormant failure rate is lower than the active failure rate.

If all the F's in equation (1) are small, say less than 0.1 (which is the case for a 168 hour mission and computer MTBF greater than 1680 hours) and all the coverages are equal and less than one, then each term in equation (1) except the last is at least an order of magnitude smaller than its predecessor so that the first term, F_1 $(1 - C_1)$ is a good approximation to the sum of the first N - 1 terms. The first term simply represents the probability that the computer initially designated as prime fails and this is not detected. As coverage approaches one, the last term in equation (1) becomes dominant and all the preceeding terms approach zero so that the probability of system success becomes limited only by the MTBF of the individual computers.

Combining the two cases discussed, we see that for any coverage and with F less than 0.1, a good approximation to F_M and F_C is:

$$F_M = F_C \approx F_1 (1 - C_1) + F^N C^{N-1}$$

3.2 MODEL 2 - MISSION MAY BE ABORTED

For this case, the mission may be aborted and an early return to earth initiated after a certain number of computer failures have been detected. This section will consider only the case of a four computer system where an abort is initiated after the second detected failure. If the second detected failure occurs when the vehicle is already returning, then no change in mission plan occurs.

3.2.1 Crew Safety

The complete equation for the probability of crew safety with four computers and a mission rule to abort after the second detected computer failure was given in Section 2.2 (see R_{C2} for the four computer case). To bound or approximate this result, it is more convenient to work with the probability of crew loss, denoted here as F_C. F_C is equal to $(1-R_{C2})$. If all computers have the same MTBF:

$$F_C = F_1 (1-C_1) R^3 + W \tag{6}$$

Where W is positive and represents the terms not shown. F_1 is the probability that the computer initially designated as prime fails before T_M, the normal mission time; R equals $(1-F_1)$. Dropping the positive W, we have a lower bound on F_C:

$$F_C > F_1 (1-C_1) R^3 \tag{7}$$

Since this expression becomes zero when coverage is one, the bound is useful only for coverage less than one. Setting F_C to the required value (1×10^{-4}), obtaining F_1 from equation (3) with $t = 168$ hours, and solving equation (7) for C_1, identifies the minimum coverage required to make the lower bound on F_C less than 1×10^{-4}:

$c_1 > 0.9981$ for MTBF = 2500 hrs.

$c_1 > 0.9966$ for MTBF = 5000 hrs.

$c_1 > 0.9936$ for MTBF = 10,000 hrs.

Comparing these bounds with those in model 1, we see that aborting early provides little relief for the coverage requirements. Note that providing coverage higher than the values shown above does not insure that F_C is lower than 10^{-4}. However, with coverage lower than these values, F_C is higher than 10^{-4}.

Equation (7) and the above conclusions also hold if the mission rule is to abort after the first or third detected computer failure.

If each computer's probability of failing during a complete mission is small, say less than 0.1 (which is the case for a 168 hour mission and computer MTBF greater than 1680 hours) and coverage is not very close to one, then equation (7) is a good approximation to F_C. When coverage approaches one, F_C becomes approximately:

$$(6F_X^2\, c^2\, R_X^2)\ (F_R^2\, c).$$

where F_X, R_X, and F_R are defined in Section 2.2. This represents the probability that exactly two computers fail with detection before T_X (so that the mission is aborted) and the two remaining computers fail during the return phase with detection of the first of these two. The sum of these two approximations is a good approximation to F_C for all values of coverage (still requiring a computer's probability of failure to be less than 0.1 for 168 hours):

$$F_C \approx F\ (1-C)\ R^3 + 6\ F_X^2\ C^3\ F_R^2\ R_X^2$$

This can be further simplified by approximating R and R_X as unity.

3.2.2 Mission Success

The complete equation for the probability of mission success with four computers and a mission rule to abort after the second detected failure was given in Section 2.2 (see R_{M2} for the four computer case). To bound or approximate this result, it is more convenient to work with the probability of mission failure, denoted here as F_M. F_M is equal to $(1-R_{M2})$. With T_X defined as in Section 2.2 to be the time at which a normal mission with no failures begins the return to earth, mission failure can be expressed as:

F_M = (the probability that the computer system fails before T_X) (8)

+ (the probability that an abort is initiated before T_X)

+ (the probability that the computer system fails after T_X)

The first term, the probability of system failure before T_X can be expressed as:

$$F_X R_X^3 (1-C_1) + F_X^2 R_X^2 [(1-C_1) + C_1 (1-C_2)] \quad (9)$$
$$+ F_X^3 R_X (Q) + F_X^4 (V)$$

where Q and V represent the sum of positive terms not shown, and F_X and R_X are as defined in Section 2.2. The second term in equation (8), the probability that an abort is initiated before T_X, can be expressed as:

$$F_X^2 R_X^2 (C_1C_2 + C_1C_3 + C_1C_4 + C_2C_3 + C_2C_4 + C_3C_4) \quad (10)$$
$$+ F_X^3 R_X(W) + F_X^4 (Y)$$

where W and Y represent the sum of positive terms not shown.

Letting Z represent the third term in equation (8) (where Z is positive), and setting $C_i = C$, we obtain F_M by the sum of equations (9), (10), and Z:

$$F_M = F_X R_X^3 (1-C) + F_X^2 R_X^2 (1 + 5 C^2) + F_X^3 R_X (Q+W) + F_X^4 (V+Y) + Z \quad (11)$$

Dropping the F^3, F^4 terms and Z, we have a lower bound on the probability of mission failure:

$$F_M > F_X R_X^3 (1-C) + F_X^2 R_X^2 (1 + 5 C^2) \quad (12)$$

Setting F_M to the desired value (1×10^{-2}), obtaining F_X from equation (3) with t = (168 - 3.36 hours), and solving equation (12) for C, identifies the minimum coverage required to make the <u>lower bound</u> on F_M less than 1×10^{-2}:

$C > 1.0$ for MTBF = 2500 hrs.

$C > 0.8$ for MTBF = 5000 hrs.

$C > 0.38$ for MTBF = 10,000 hrs.

What this means is that with computer MTBF of 2500 hours, F_M is sure to exceed 1×10^{-2}. With coverage higher than the values shown for the other two cases, F_M may be lower than 1×10^{-2}. With coverage lower than the values shown for 5000 and 10,000 hours, F_M is higher than 1×10^{-2}.

Equation (12) is a good approximation to F_M if each computer's probability of failing before T_X is small, say less than 0.1 (which is the case for a 168 hour mission with a 3.36 hour return time and computer MTBF greater than 1648 hours), all coverages are equal, and the return time (T_R) is a small fraction of the mission time (T_M), say less than 0.1 (which is the case for T_R of 3.36 hours and T_M of 168 hours).

3.2.3. Summary of Bounds for Early Return Model

To summarize Section 3.2, we have found that for the three specific MTBF's considered, the F_M requirement is the driver for MTBF of 2500 hrs. and that the F_C requirement is the driver for MTBF of 5000 or 10,000 hrs. To <u>possibly</u> meet both requirements, an MTBF of 5000 hours and coverage greater than .9966 is required, or an MTBF of 10,000 hrs. and coverage greater than 0.9936 is required. With MTBF of 2500 hrs., the mission success requirement cannot be met. All of these conclusions are substantiated by the results of Section 2 shown in Table A.

4. POSSIBLE REFINEMENTS TO RELIABILITY MODEL.

The computer system reliability model used in this memo makes certain simplifying assumptions about the operational aspects of the Shuttle mission. The assumptions made in the model were generally intended to represent the worst case situation as far as identifying the impact of the reliability requirements upon the computer system. A more refined model should consider the following points:

1) Not all computer failure modes result in the loss of the mission or crew. Certain computer functions are not life critical or mission critical.

2) The maximum allowable time for error detection and recovery varies with mission mode. In non time critical periods, a slower response may be acceptable. In these more relaxed periods, the effective coverage may be increased by assistance from the crew and possibly the ground which may not be possible during time critical periods.

3) The probability of recovery after error detection is assumed to be perfect in this memo, but, in a standby replacement system as is discussed in this memo, errors are not masked. The assumption of perfect recovery is optimistic, and realistic recovery probabilities should be accounted for.

4) Some of the spare computers may be powered off for certain phases of the mission. The failure rate of a dormant computer may be lower than when it is active.

5) Computer failure rate is a function of the mission environment. Factors such as vibration, thermal cycling, and turn-on transients influence the effective computer MTBF.

REFERENCES FOR PART III

1. W.G. Bouricius, W.C. Carter, and P.R. Schneider, "Reliability Modeling Techniques for Self-Repairing Computer Systems," Proc. ACM 1969 Annual Conf., pp. 295-309.

(This paper is a very good treatment of the effect of coverage upon reliability. It concludes that coverage is the single most important parameter in high reliability system design).

2. G.P. Edmonds, "Proposed Mission Reliability Analysis Technique for the Shuttle", Draper Lab Group 23S Memo # 72-56, 25 September, 1972.

(This memo discusses a reliability model for the entire GN&C system using Apollo reliability statistics. Since the publication of this memo, considerable refinement of the model and statistics has taken place.)