GUIDANCE, NAVIGATION AND CONTROL

R-682

A FAULT-TOLERANT INFORMATION PROCESSING CONCEPT FOR SPACE VEHICLES

by

A. L. Hopkins, Jr.

DECEMBER 1970

MIT CHARLES STARK DRAPER LABORATORY

CAMBRIDGE MASSACHUSETTS 02139

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

APOLLO

GUIDANCE, NAVIGATION AND CONTROL

Approved: Eldon C Hall Date: 1/19/71
E. C. HALL, DIRECTOR, DIGITAL DEVELOPMENT
CHARLES STARK DRAPER LABORATORY

Approved: David G Hoag Date: 26 Jan 71
D. G. HOAG, DIRECTOR
APOLLO GUIDANCE AND NAVIGATION PROGRAM

Approved: R R Ragan Date: 28 Ja 71
R. R. RAGAN, DEPUTY DIRECTOR
CHARLES STARK DRAPER LABORATORY

to be published by the IEEE in
"Transactions on Computers"

R-682

AFAULT-TOLERANT INFORMATION PROCESSING CONCEPTFOR SPACEVEHICLES

by

A.L. Hopkins, Jr.

DECEMBER 1970

MIT CHARLES STARK DRAPER LABORATORY

CAMBRIDGE, MASSACHUSETTS, 02139

ACKNOWLEDGEMENT

This report was prepared under DSR Project 55-23890, sponsored by the Manned Spacecraft Center of the National Aeronautics and Space Administration through Contract NXS Y-4065 with the Massachusetts Institute of Technology.

The author would like to acknowledge the invaluable work of Alan I. Green, Robert J. Filene, and William W. Weinstein; and the encouragement and support of Ralph R. Ragan and Eldon C. Hall, of the MIT Charles Stark Draper Laboratory. He also wishes to thank Cline W. Frasier and Thomas V. Chambers of NASA/MSC for their important early contributions.

INTRODUCTION

Fault-tolerant computer development has been compared to surgery, where new techniques may be practiced only upon “otherwise hopeless cases”. [1] This comparison may be somewhat strong in terms of computer technology today, but without question the use of computer redundancy has been virtually limited to cases where digital processing and infallibility have been indispensable or irresistably rewarding. The SAGE air defense computer system, the ESS electronic switching telephone exchange, and the LVDC Saturn V guidance and control computer are examples of relatively early approaches to fault tolerance, which were all wrought at what appeared to be a very large cost, i.e. replicated equipment. Without any doubt, however, these were reasonable approaches both because the cost of additional equipment was tolerable, and because there was no expeditious alternative.

Increasingly more fault-tolerant system applications are at hand, but the fault-tolerant systems available in the market place, with a few exceptions, provide scarcely more fault tolerance than systems designed a decade ago. Cne possible conclusion is that, despite the fact that one can demonstrate that fault tolerance can be achieved with fewer active elements than by outright replication, the greatest economy is achieved in most cases by duplex configurations and two-processor multiprocessors. The exceptional systems available today are multiprocessors which contain more than two processors, and which endeavor to provide graceful degradation of their processing and memory resources.

To date, there has been no commercial system announced which has had the capability of tolerating any fault whatsoever with guaranteed recovery. There are a number of developmental efforts in this direction, but the prevailing attitude has been that it is too complex and costly in nearly all cases to provide such total fault immunity, and, in any case, such immunity exists only if faults are statistically independent. Some of the current ambitions in the aerospace field, however, are providing us with some “otherwise hopeless cases”, where survivability of a kind heretofore unknown is required. How to provide it is a question being addressed by many independent designers and scholars, and the suggested

approaches[1] differ considerably, to some extent because there are as yet no agreed-upon figures of merit for the required system, and no single date by which they must be in the field.

No unified figure of merit or solidification is attempted in this paper. Rather it is a representation of the consensus of opinion of one group of workers with experience in the field of manned space flight as to the most expeditious approach to fault-tolerant information processing in that field and closely related fields in the 1975-1985 time frame.

GENERAL DESCRIPTION OF A SPACE FLIGHT INFORMATION PROCESSING SYSTEM

For space flight in the 1975-1985 time period, and particularly for manned missions, an information processing system is required which will have functional longevity and a very high probability of performing correctly during time-critical mission-phases. To meet these requirements, the components of the system will have to exhibit very low failure rates, probably lower than any so far observed in components of the complexity required. The system will also have to be able to tolerate one or more transient or permanent failures during its mission life with only occasional opportunities or perhaps with no opportunities for repair. Component reliability is at best a random variable, and no system can truly be depended on which does not allow for one or more failures. On the other hand, however, it is just as inappropriate to design for fault tolerance to a high number of failures, because if the probability for n failures is not negligible, the probability for n + 1 failures is less apt to be negligible, the greater the value of n. It is difficult to conceive of designing for n greater than three or four if failures are random.

In addition to equipment redundancy and reconfiguration, the system must provide for continuity of information flow. This requirement can sometimes prevent cost savings which would otherwise be possible. Implicit in this requirement is the necessity of detecting and correcting errors,

[1] See, for example, References [2], [3], [4], [5].

whether transient or permanent, before the dataneeded for smooth recovery is destroyed. This rules out the possibility of depending on periodic self-testing programs for fault-tolerant behavior unless all such data is buffered between times. Transient errors can not be detected by this method.

A large number of other system attributes are desirable as well, all of which are familiar to aerospace computer designers, including low cost, small size, expandability, adaptability, ease of programming, ease of debugging, ease of integration into the system, minimum connections, and so on. Probably the first decision one would like to make is whether the systems should be centralized, dedicated, or distributed, the latter of which embodies elements of central and dedicated computers. Since this decision must be made in context with the problem, it is first useful to summarize some advantages and disadvantages of central and dedicated computers.

Advantages of a central computer — A central computer allows the pooling of resources for the sake of repair (reconfiguration) and allocation to meet instantaneous demands. The more centralized a system is, the more efficient it will be of cost and size, due to such overhead considerations as packaging and power conversion and regulation. In particular, the multiprocessor type of structure affords an excellent way to configure the elements of a central computer to realize these advantages.

Disadvantages of a central computer — In general applications, a central computer is made feasible only by the incorporation of digital logic elsewhere in the system, e.g. peripherals, modems. This is primarily for two reasons: one is the inefficiency of dedicated interfaces for far-flung communications; the other is the awkwardness of incorporating all sequencing and coding functions in a central computer. In this case, if the central computer is a fault-tolerant multiprocessor, the incorporation of a large dedicated set of interfaces would be so expensive to accomplish in redundant and infallible fashion as to mask the fundamental economy of the multiprocessor.

Advantages of dedicated computers — If one were to adopt a free interpretation of what a computer is, the remote logic used in central computer systems might be called dedicated computers. Indeed some of the peripheral controllers made today are so complex that they will soon be realized using small computers. In this light, therefore, one could argue that dedicated computers exist whether or not they are wanted. The advantage of using them to the exclusion of central computers lies in the fact that they can be integrated into subsystems early in the design and production cycles and can be programmed without possible interference from other programs in other computers except via the interfaces. If desired, they can also be tailored to their specific applications.

Disadvantages of dedicated computers — These are more or less opposite to the advantages of a central computer with respect to resource pooling. Another disadvantage is in the amount of intercomputer communication which has to take place in a complex system.

Enough has been said already to reveal that in most applications there will almost necessarily be some combination of central and dedicated elements, even if one of the dedicated computers were to serve a coordinating role. Therefore, the first decision is really not so much the form the system should take, but rather how the distributed system should be partitioned between central and dedicated elements. The requirement for fault tolerance in a central computer can be met by a carefully constructed multiprocessor, which can also possess some of the other desirable system attributes named earlier. The principal limitations of this design technique lie in its input/output bandwidth and reaction speed. To interface a multiprocessor with a set of subsystems requires some form of local digital processing with dedicated interfaces to subsystem electronics. Here, a different approach is needed to implement the required degree of fault tolerance, which will depend on the nature of the subsystem itself, and what technique is used to prevent its faults from adversely affecting the mission.

The nature of the dedicated processing equipment depends on what it is expected to do and, of course, on the characteristics of the digital circuitry to be used as predicted at the time of the design. To be somewhat forward-looking, it is not unreasonable to assume that each subsystem will contain one or more digital computers acting as sequencers, format converters, test elements, and data filters. As such, they would constitute an excellent functional complement to the central multiprocessor. They would have the high input/ output bandwidth and reaction speed needed to communicate with subsystem electronics, with the capability to pre-digest this data and communicate with the central multiprocessor at substantially lower bandwidth and with less reaction speed required.

Communication between central and dedicated (local) computers would be via a multiplexed bus, with a bandwidth capability of the order of a million pulses per second. The bus must be redundant for the sake of fault tolerance, and must be interconnected in such a way that no fault can render all of the buses useless as would happen, for example, if a defective processor transmitted "garbage" on all copies of the bus.

Figure 1 shows the hierarchy of such a system. The multiprocessor, the local processor, the bus, and the multiplexers which protect the bus are described in the following sections.

A FAULT-TOLERANT MULTIPROCESSOR

The multiprocessor concept described in this section has as its goal the ability to reconfigure without loss of data, to be flexible and expandable, and to be easy to program. To do this requires error detection, infallible data recovery, redundant processors, memories and data paths, and some means of avoiding memory conflicts. Error detection, moreover, must be accomplished soon enough to enable correction to be made before errors spread.

Although multiprocessors need little justification, it might be pointed out that if a central computer is implemented as a single "large" computer with spares, or as several "large" computers in a voting arrangement,

then to tolerate n faults one needs n + 1 or more “large” computers, and expandability is very poor owing to the non-modularity of the system. The multiprocessor provides a means of using “smaller” computers simultaneously, where the overhead for fault tolerance is n “small” computers rather than n “large” computers. The greater the ratio of “large” to “small” the greater the equipment saving. Also, the system is modular, and changes in size can be accommodated relatively late in the design cycle.

The emphasis in design is on fault tolerance, rather than large throughput. For this reason, the multiprocessor will differ in several respects from commercial multiprocessors whose goal is large and efficient throughput.

To meet the goals for this multiprocessor, there must be comprehensive error detection in all elements. Each processor must be able to detect, sufficiently quickly, the occurrence of an error, and retire gracefully. Each memory unit must be paralleled by at least one other, and be able to detect its own error and retire or to acquire a new parallel unit when necessary. There must moreover be a means of error correction and/or recovery; and a substantial benefit accrues if these procedures are transparent to applications programs.

The remaining unspecified design consideration is the origin and nature of faults which may occur. At worst, faults may occur in a correlated way, such as by battle damage in combat aircraft, where multiple faults are induced virtually simultaneously. In the more benign environment of spacecraft however, fault occurrences are more nearly independent of one another, provided there are no systematic failure modes built into the equipment. If one can assume randomness of faults, then the concept of redundancy is valid, and advantage can be taken of the high probability that faults will not occur close together, e.g. within one tenth of a second in a thirty-day mission.

The design of this multiprocessor, then, is an exercise involving the design of error-detecting processors, error-detecting and correcting memories, and the additional mechanisms and procedures to guarantee smooth recovery.

The Processor

Error detection in processors has been accomplished in many ways, using techniques ranging from program checks to coded and massive redundancy. At this point in time, however, the state of the art of error detection is rapidly being overtaken by the state of the art of integrated circuitry, and the most economical method of providing comprehensive single error detection will be to use a second, synchronized, processor to check the first one via a redundant comparator on the memory interface. This is not only superior to coded redundancy in that it detects all but a very few multiple errors (i.e. identical errors in each processor), but it is more economical, in that the absence of the many encoders and decoders needed for comprehensive coded redundancy drastically reduces interconnections. This is the dominant cost criterion in integrated circuits, to the extent that two identical processors will be comparable in size to, and cost less than, one processor using extensive coded redundancy for error detection. Logic is inexpensive, if and only if it is highly regular, minimally interconnected, and mass produced.

The basic processor element is thus envisaged to be two processors with absolutely no attempt made to salvage and reassign individual processors during a mission. To do so would require more connections than would be economical. An alternative approach is possible, however, in which two or three individual unchecked processors are assigned a job at about the same time, and have their outputs compared or voted on by the memory or input/output unit receiving these outputs. With paired processors, the programmer is relieved of any burden relating to error detection, and an error is detected immediately upon occurrence. What is done upon occurrence depends on the location of the data to be salvaged. If it is stored in memory units, it is necessary to salvage the central register contents to allow another processor to re-run the instruction. If a scratchpad memory is used, the central register contents plus the scratchpad contents must be salvaged for automatic data continuity. The alternative to central register salvage is to provide re-run points, which the programmer is normally responsible for updating with each data store and job request. Even this will fail if an error occurs partway through data storing, unless

a two-step storing process is used. This is closely related to one of the major caveats of multiprocessing, which is that correlated data must be updated at one time without allowing another processor to read it part way through the update. Errors committed during the process exacerbate the problem.

One satisfactory solution to this problem has been proposed at the MIT Charles Stark Draper Laboratory.[4],[6] A triplicated scratchpad memory is used for data and instruction storage in conjunction with each processor pair. All central registers are located in the scratchpad, and all have buffer registers also located in the scratchpad. Each instruction has two phases: one for computing, one for storing. Errors are detected within each phase and processing is stopped before proceeding to the next phase. An error signal in the processor pair or in the triplicated scratchpad triggers a sequence generator in each scratchpad to initiate a dump sequence into a reserved area in main memory. The dump includes the name of the phase that failed, No other processor can interrupt the dump, and if a second error does not occur by the time the dump is completed, the dump will be made correctly. The duration of the dump for a 128-word scratchpad would be of the order of 0.1 millisecond. A similar procedure is used for all updates of correlated data, which are therefore done in one motion, protected from error. The cost is one more scratchpad memory per processor than would otherwise be needed for error detection. In terms of circuit volume in 1975, this can be projected to be no more than a few cubic inches. What it provides is a mechanism for information continuity and protection of correlated data, which are both of great importance in reliability, and which are otherwise very difficult to achieve.

Figure 2 illustrates a processing unit comprising two processors, two dual comparators (which can use common circuitry), three scratchpads, and voting units necessary for fault isolation from quadruply redundant serial buses. This unit is capable of 1) operation as a collaborative processor, 2) detection of any error resulting from a single fault, 3) completing any data transfers in progress with memory, system, or other systems, 4) transferring all necessary information to memory to effect resumption of processing and reconfiguration, and 5) entering a self-check

mode to attempt to qualify for returning to active status. Too many such attempts cause the executive program to disqualify the processing unit from further activity. All of the foregoing is dependent upon its being completed before two of the scratchpads experience faults. The time scale for all but the last item is tens of milliseconds. The last may continue over a large time period, if the fault is persistent or permanent but seldom causes errors. In this case it must be assumed that enough errors are caused to disqualify the processing unit before the second scratchpad failure. This assumption is valid if, and only if, the self-check mode is exhaustive in its exposure of faults.

All communication with external units is via serial redundant buses, with the possible exception of the processor-memory bus, which may be made byte-serial or parallel for the sake of high bandwidth. The multiprocessor can be configured to tolerate the failure of n of these processing units if n extraunits are incorporated over and above the number necessary to furnish sufficient processing capability for the mission. The bus system shown is two-fault tolerant, expandable to n faults by using n + 2 buses and an appropriate configuration of voters,

The Memory Unit

Error detection and correction in memory units is essentially different from error detection and correction in processors, in that where logic is cheap, memory is not. More important, however, is that where a processor is time-shared, memory systems are not. That is, neglecting swapping in hierarchical storage systems, all words of memory must be retained and backed up actively at all times, unlike processors, where no active continuity is required. Therefore, if duplication is used for error detection, then a third memory is required for correction of the first error, and either a fourth memory or else a reassigned third is needed for correction of the second error and so on.

Coded error detection methods are far more viable in the case of memories than for processors for two reasons: they are more comprehensive, and they occupy a far smaller fraction of memory than they do of

processors. This is not to say that a trivial check like single parity is adequate; far more comprehensive checks are needed, especially if the memory has sophisticated internal sequencing such as test- and- set operations on flag bits, important in global variable write protection, or page table management. But until a coded redundancy design is actually in hand and verified to be comprehensive for all single faults, the only acceptable assumption to make is that error detection will require two memory units as set out in the preceding paragraph.

The next item to be discussed is the method of interconnecting the memoryunits with the processors. Three such methods are distkguished: crossbar, redundant time-shared bus, and multiport units. *With the crossbar, a processor makes a temporary (interruptible) hookup with a memory unit for the duration of a job whose instructions and data are located in that memory. The processor must wait if some other processor has access. With a time- shared bus, no particular affiliation pattern emerges, as each processor in turn has access to the entire set of memories through a single bottleneck. With multiport memories, processors can request access to any one memory unit at any time,, which will be granted in order of priority among those seeking access to that unit. In general, instructions are read very few at a time to avoid the inefficiencies which would otherwise result from branching early in an instruction block. Data, on the other hand, may very well be accessed in blocks, for which reason it may be stored in units separate from instructions.

The crossbar and time- shared bus techniques easily lend themselves to allowing a processor to “hold on” to a memory to write correlated data, but only the bus technique allows a processor to have momentary access to all memory units to write correlated data into scattered locations.

Whichever memory-processor interconnection technique may be used, care must be taken in its design to ensure that no combination of n or fewer faults can bring down the system. As examples, two memory approaches are described below, both of which use the time shared serial bus shown in Fig. 2. The first approach, illustrated in Fig. 3, has n + 2 redundant memory units for n-fault tolerance. Again, four buses are shown

for the case of two-fault tolerance. Each memory unit comprises enough modules (m innumber) to store all data, and may be a hierarchical memory system with mass memory. This is conceptually the simpler, more effective, but possibly more costly of the two approaches shown. Each unit has a control unit, similar to a processor, which formats information for transfer and executes access sequences such as read, write, and test-and-set. At least two of these four control units must survive for system survival.

The second approach is shown in Fig. 4. It requires fewer memory modules than the first approach, but more control modules, each of which is more complicated. In the second approach, n-l spare memory modules are available for assignment to replace failed modules in two-out-of three voting groups. This requires the ability to load replacement modules to match the survivors of the group; plus the ability to identify a disagreeing module in a group, plus the ability to ensure that a module cannot falsely respond to a request for access to data it does not contain. The total number of memory modules used is 3m + n-l, as compared to m (n + 2) in the first approach. Each memory module needs a control unit in this approach as compared to the four used in the first approach. For example, to achieve three-fault tolerance (n=3) where the number of different words stored is four times the capacity of one memory module (m=4), then the first approach uses 20 memory modules and four control units, where the second uses 14 memory modules and 14 control units.

The choice between these approaches is not immediately obvious, and, moreover, several other approaches may be competitive with these two. This area is as yet far from adequately understood.

Input/Output and Executive

Access from the central multiprocessor to the input/output redundant bus may be made by way of the memory complex or, if used, by way of the scratchpad memories. Direct access from processors is undesirable because of the problem of data continuity at the time of a processor failure. The criteria for memory error detection and correction make memories the most appropriate access points for the input/output bus.

Operationally, a time-shared input/output channel requires a certain amount of datamanagement. One promising approach, again from the MIT Draper Laboratory, [6] is the quasi-dedication of one of the processor pairs of the multiprocessor to management of the input and output through its scratchpad. In this scheme, the processor that handles the input/output continues to do so uninterrupted until it commits an error, at which time the job is transferred to any free processor. This is like having a distinct input/output processor, except that it is taken from the processor pool, and needs no special backup. The reason for quasi-dedication of this processor as opposed to letting any free processor manage the input and output on a sampled basis is to provide full use of the available bandwidth.

The input/output bus is managed by the central multiprocessor, and the remote units (dedicated processors) respond on a speak-when-spoken-to basis. In this way there are no program interrupts in the usual sense. Job requests are sent to the central multiprocessor in response to polling inquiries, issued at a rate of roughly one every few milliseconds. This is approximately the rate at which jobs are dispatched, and is a fair measure of the reaction time of the machine. It would be possible to implement conventional program interrupts, but it would be awkward and unnecessary, In fact, with no more program restrictions than are observed in conventional machines, it is possible to time share a multiprocessor without interrupt, as easily as, or more easily than, time sharing amultiprogrammed machine, provided that the number of processors is greater than the number of jobs which are allowed unrestricted running time, such as the input/output job.

The executive program, which is responsible for allocating resources to jobs which are called by automatic or manual inputs, is of a floating form. Any processor may insert a job request in the executive input file during the execution of a job segment. Each processor, upon completing a job segment (which it must do within a specified time or be dumped) first checks to see if any pathological situation exists, in which case it takes care of it. If not, the processor attempts to examine the executive file in memory. If the file is busy, the processor waits. If not, the processor updates the file and takes the highest priority job to do. There is virtually nothing else to be done at high speed. At low speed, another segment of

the executive program handles the allocation of memory resources. There is little executive overhead, a moderate amount of input/output overhead, and an overhead chargeable to memory access depending on whether a scratchpad is used and what the statistics of its use are.

LOCAL PROCESSORS IN A FAULT-TOLERANT SYSTEM [4]

The use of small dedicated processors in subsystems is a new extension of an old idea. The extension comes about because of the new, inexpensive and small realizations of processors that are being developed now. It is not unreasonable to expect to have simple processors with ten microsecond add times and 4K words of memory which occupy only about ten cubic inches of volume, available in the time span with which this effort is concerned.

No one redundancy configuration of local processors appears to be basically superior to another, because of the wide variety of applications. In rocket propulsion, redundancy is often achieved in using multi-engine stages, where the life expectancy of the engine is less than that of a computer. It would appear to be sensible to use a single processor pair per engine, which could shut the engine down if an engine failure or computer error were detected. In a power distribution system, on the other hand, the system itself is redundant, and a single redundant computer complex seems appropriate for control of the whole system. This may be done in either of two basic ways. One is a set of paired processors in which one is active (at bat), one is in standby (on deck), one inactive (in the hole) and so forth. Input/output is handled by the active processor, whose interface circuits are enabled by power switching. The second way is to use a set of single processors three at a time in synchronism with voter circuits on each output signal. This uses fewer processors for a given number of tolerable faults, but requires a more expensive input/output complex.

For either type of redundant local processor configuration, redundant inputs from the subsystem(s) are brought into all of the processors to be compared or voted upon. Analog inputs in particular do not lend themselves to hardware comparison or voting, and, in general, inputs will not coincide well enough in time to permit hardware voting.

Local processors will undoubtedly use conventional approaches to multi-job real-time control ranging from programmed scanning of inputs tomultiprogramming. This is of little concern to the fault-tolerant aspect of the system except as it emphasizes the need for multi-phased input circuitry to avoid pulse slivering that might force the redundant processors out of synchronism and restart mechanisms which can resynchronize them if they do slip out.

A FAULT-TOLERANT REDUNDANT INPUT/OUTPUT BUS

A parallel multiplexed bus can be made fault-tolerant by coded redundancy, provided the multiplexers for each bit line are independent and capable of determining for themselves when to allow access. In this way a multiplexer failure only affects one line, whose erroneous behavior is filtered out by all of the error-correcting decoders. Aserial multiplexed bus can be made fault-tolerant by replication with error detection by coded or comparative means. Voting, however, best assures information continuity in the serial case, using n + 2 buses for n-fault tolerance. Parity may be used to signal ultimate breakdown of the bus system, but it is inadequate to detect all possible erroneous behavior in a single line.

It may be expedient to route separate bus lines through separate conduits to avoid the possibility of system breakdown through physical damage. Alternatively, all lines might be run through a single, well armored, cable that may be assumed to be indestructible.

Electrically, the bus presents an interesting challenge, because there may be as many as the order of 100 stations. One solution is to arrange the stations in a daisy chain where information is transmitted from one station to one other, whence it is transmitted to a third, and so on in order through all stations. The originating station alone refrains from transmitting the information arriving from its neighbor. When this station's message ends, another station may open its tie from its receiver and begin to transmit its information. Any failure in any station makes the whole loop faulty. A second solution is to connect all transmitters and receivers in parallel. Only one station transmits at a time, the other transmitter connections

being passive. An unpowered station would not affect bus operation, so that faulty units could be retired from the system by being shut down.

Each bus line would interface with each central or local processor through a separate multiplexing station. Each station would contain a transmitter, a receiver, and logic. Functionally, the multiplexer would need to be able to recognize when it is eligible to transmit and when it is not. For local processor multiplexers, this requires logic to ascertain that its name is called by the multiprocessor, and to place a limit on the length of the message that its local processor tries to send, For central processor multiplexers, the quasi-dedicated input/output processor is the only one enabled to transmit, and it begins transmitting as soon as required after the end of a message from a local processor. This end of message is signified by a synchronizing pulse issued by the local processor's multiplexers.

Multiplexers have the additional function of turning power on and off in local processors for purposes of mission sequencing and of reconfiguration following detection of errors.

The multiplexer is a key element in the system. It has many responsibilities and is used in large number. To make the system feasible, it will be necessary to implement these devices using customized integrated circuits. Because of the irregular nature of the logic required, it will not be packaged as efficiently as memories and arithmetic units, and it may require from two to six chips for its implementation, depending upon a number of factors not yet known.

A serial implementation is proposed for the input/output bus for spacecraft requirements, assuming a requirement for two-fault tolerance, and estimating a bandwidth requirement of tens of kilobits per second. Voting is done in each processor on identical. and simultaneous information received over four identical bus lines. The voting strategy is two out of three with one replacement. This strategy may not be totally satisfactory, however, because it fails to accommodate errors which develop in different multiplexers on different lines. Thus it requires two lines with all their

multiplexers to survive, where a three-out-of-five arrangement would allow any two multiplexers in every processor to fail. The choice must be based on the reliability required and the component failure-rate predictions.

SUMMARY

This paper has so far discussed a number of generalities about contemporary approaches to fault-tolerant systems. It also has tried to convey in brief the important elements of a proposed spaceborne fault-tolerant information processing system which is felt to represent a propitious solution to a many-faceted problem. There are several notable features in this concept. One is a new assessment of the capabilities and limitations of integrated circuits. Specialized circuits are costly to design, difficult to qualify, and wasteful of interconnections. The only such circuits contemplated are for multiplexers, which seem to be unavoidable. Nearly all of the rest of the system circuitry consists of memories and arithmetic units.

Error detection in the proposed system is accomplished largely through the use of replication and comparison of units of significant size. This provides comprehensive detection at a cost which may be lower than that of less powerful detection methods. This paradoxical situation arises from the fact that a simple, unchecked processor can now be realized with remarkably few semiconductor devices. Any circuit complexities, such as those incurred in many coded redundancy schemes, tend to result in a much greater volume and/ or a greater cost of developing integrated circuits, because of their high incidence of "random" logic.

One of the key features of the concept is that error recovery is implemented by providing adequate buffering of data to enable an expeditious and exact rerun of an instruction, job segment, or program as appropriate to the location and nature of the error. Hardware reconfiguration by itself is generally inadequate without a mechanism of dynamic program and data recovery. The expenditure of extra hardware to solve this problem is well justified. In general, fault detection can range from inexpensive reasonableness checks to outright duplication and comparison, and the means

used will have a substantial impact on the nature of the data recovery mechanism used. For example, detection and recovery are both provided in straightforward fashion by voting redundancy. The system described here, however, provides a less expensive and more efficacious combination of techniques to do the job without sacrificing too much of the conceptual simplicity of the voting method. The importance of conceptual simplicity should not be underestimated. It yields advantages in design, test, production, and marketing, i.e. the customer's willingness to accept the development and deployment risk. One other thing which must be kept in mind is that redundancy alone fails to improve dependability unless component part reliability is high. For the spaceborne applications contemplated, it will be necessary to achieve greater component reliability than ever before, even if multiple-fault tolerance exists in the system. Component reliability is not independent of circuit design, however, and simplicity of design has yet another potential benefit in this respect,

Another feature of this concept is that it is not restricted to the synthesis of a fault tolerant computer. Rather the entire information processing system must be considered. The problem of handling hundreds of real-time inputs and outputs in a single computer is difficult, and distributed systems are increasingly being developed for on-line applications. This approach is advantageous where local control functions involve simple data processing and high information rates and global control functions involve sophisticated calculation ("number crunching") and low information rates. Such is the case in most of the large aerospace vehicles now being developed. The concept presented here is a distributed system with fault tolerance concepts extended to all of its members and the communication medium between them.

A central computer handles the global computing functions. It is configured as a collaborative multiprocessor, designed to incorporate an invulnerable floating executive and input/output control. Each processor is duplicated for fault detection, and has a triplicated scratchpad memory for error detection and masking of data critical to error recovery. In the event of a processor or scratchpad fault, the scratchpad memory complex sends its contents to the data memory, where it will be acquired by the

next free processor, which will continue processing from the beginning of the interrupted instruction. No attempt is made to reconfigure the failed processor. If the fault was transient, as may be most likely, the processor will come back on line after it successfully executes a self-check program. Otherwise, it is totally expendable.

The central multiprocessor has a redundant main memory accessible to all of the processors. Although memories can be checked to a fairly high degree of confidence by coded techniques, the use of voting is urged here wherever the extra few pounds of weight incurred is acceptable, for the sake of increased confidence.

The central computer is capable of communicating with similar central computers in other areas of a vehicle, other stages, ground test equipment, etc. The communication medium is a redundant serial time-shared bus. An identical redundant bus system is used for communication between a central computer and all of the distributed, or local, processors. A bandwidth of about one megabit per second is probably adequate for all of the projected applications of such systems.

Local processors will resemble the processors in the central computer. Either a voting arrangement of three or more local processors, a switched group of paired local processors or independent processors in redundant subsystems may be used. The tradeoff between the schemes is basically the cost of voting circuits in the voting arrangement, the cost of extra processors and the switching and recovery mechanism in the switched group arrangement, and the expendability of the independent subsystems. All arrangements may be used within a single system.

A requirement for any fault-tolerant system is to be fully testable to ensure that the recovery capability is functional. In this system, for example, it will be necessary to check periodically that comparator circuits will alarm on any disagreement. This can be achieved on a routine basis by microprogram, which would be the only area of intrinsic difference between two paired processors. The comparator and alarm circuits would be exercised in this way without actually inducing errors in the processor.

Voting circuits, likewise, need to be checked, which calls for the inclusion of special modes of operation in virtually all elements of the system.

Systems structured along the lines of the preceding discussion may be configured to be tolerant of single, double, or indeed any number of faults, with the restriction that certain fault pairs must not occur simultaneously (within x milliseconds of one another) as might happen if the system is susceptible to electromagnetic interference. Such systems are within the state of the art using large scale integrated and hybrid circuits, both bipolar and MOS, The organization permits localization and identification of all errors which can expedite maintenance, and signal mission plan changes which should be made under existing circumstances.

Some of the important potential features of the system are:

1. Uniform information interfaces.
2. Minimization of harnessing through use of time shared buses,
3. Ability to add (or delete) information processing resources,
4. Flexible capability for adding or deleting functional interfaces by software.
5. Degree of fault tolerance can be varied within a system.
6. Flexible resource allocation capability in the multiprocessor.
7. Elimination of need for programmer to implement recovery algorithms for hardware fault restarts.
8. Ability to execute programs developed and verified by independent teams.

CONCLUSION

Aerospace technology is increasingly developing vehicles for which system survival is a paramount issue, yet which require sophisticated onboard information processing. Examples are the “giant” commercial aircraft, reusable space shuttles, space stations, deep planetary mission vehicles, and nuclear rocket stages.

Avionics technology lags behind structural technology in the area of survival, probably because it has only recently become critical to crew and passenger safety or to cost effectiveness. Fault-tolerant computers and systems are still in their infancy, yet they are sorely needed in the current and next generations of aerospace systems.

This system concept is not claimed to represent the ultimate that is achievable through fault tolerant design techniques. It does, however, represent a practical approach to solving a current need for comprehensive fault tolerance, for which no adequate systems are available. Moreover, it does so in a fashion which is transparent to programming and to most of the electronic design. It is a conceptually- simple information processing system well suited to the functional requirements of many of the most sophisticated space vehicles.

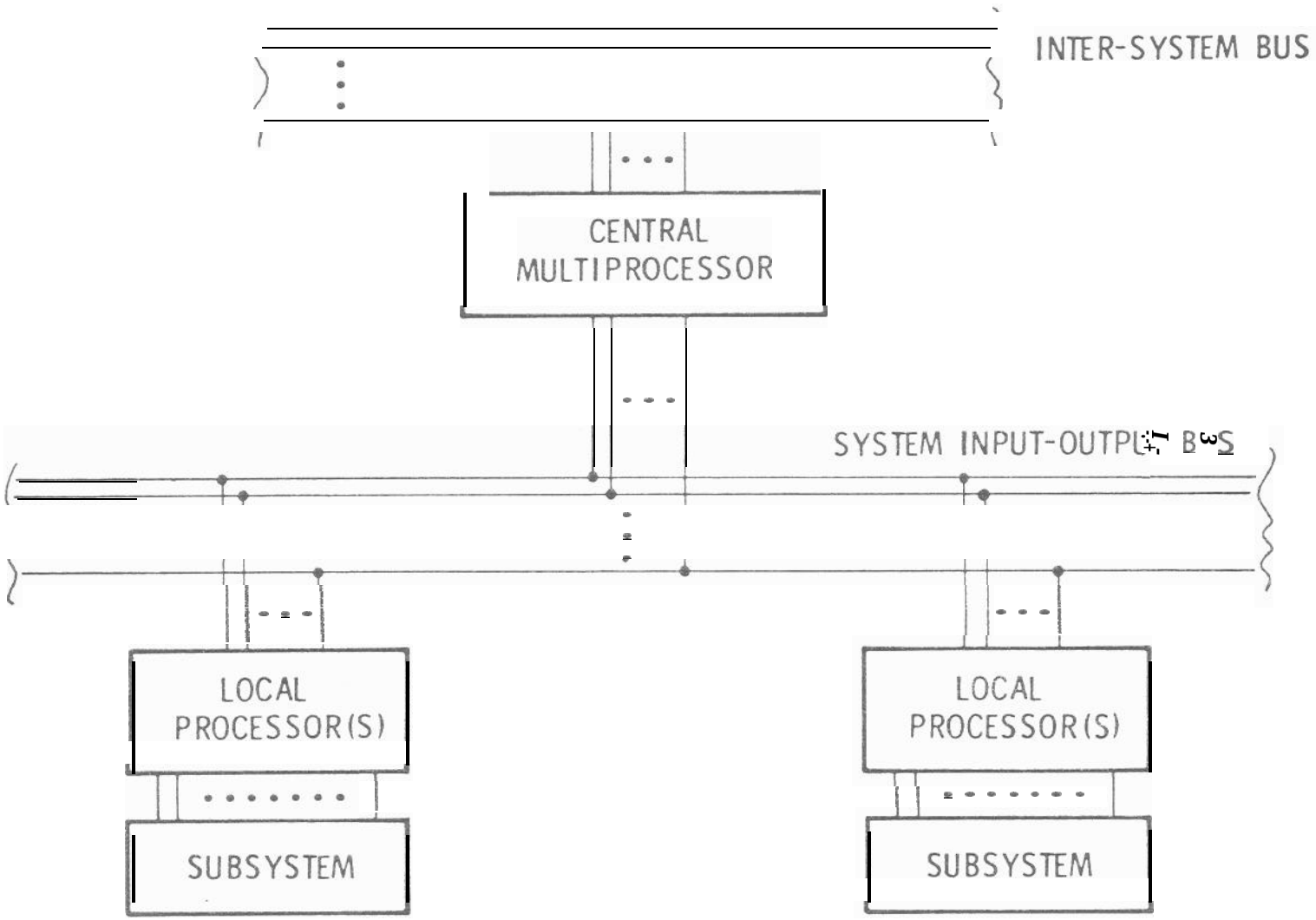

Figure 1 A Distributed Fault-Tolerant Digital System

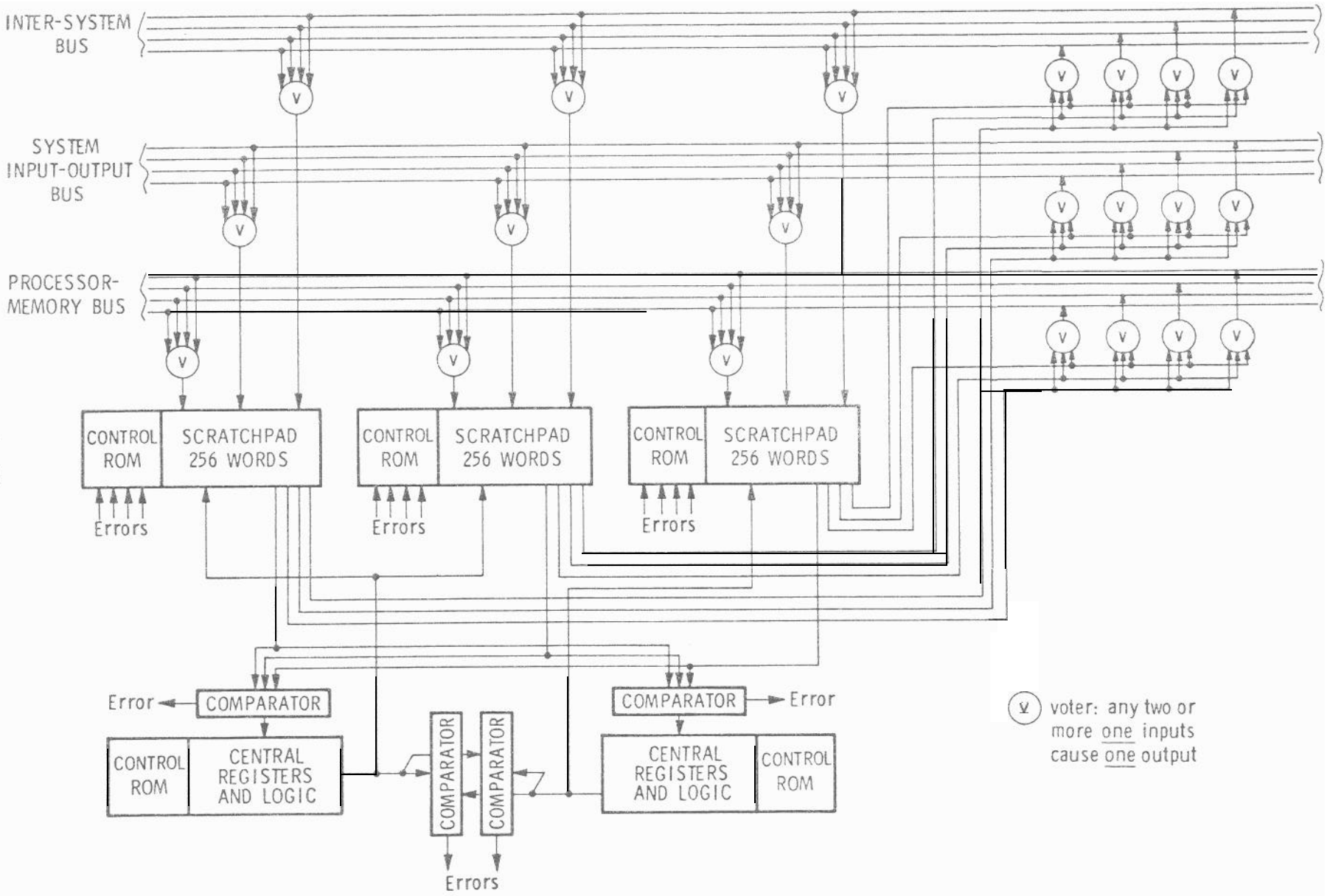

Figure 2 One Processing Unit of a Collaborative, Fault-Tolerant Multiprocessor

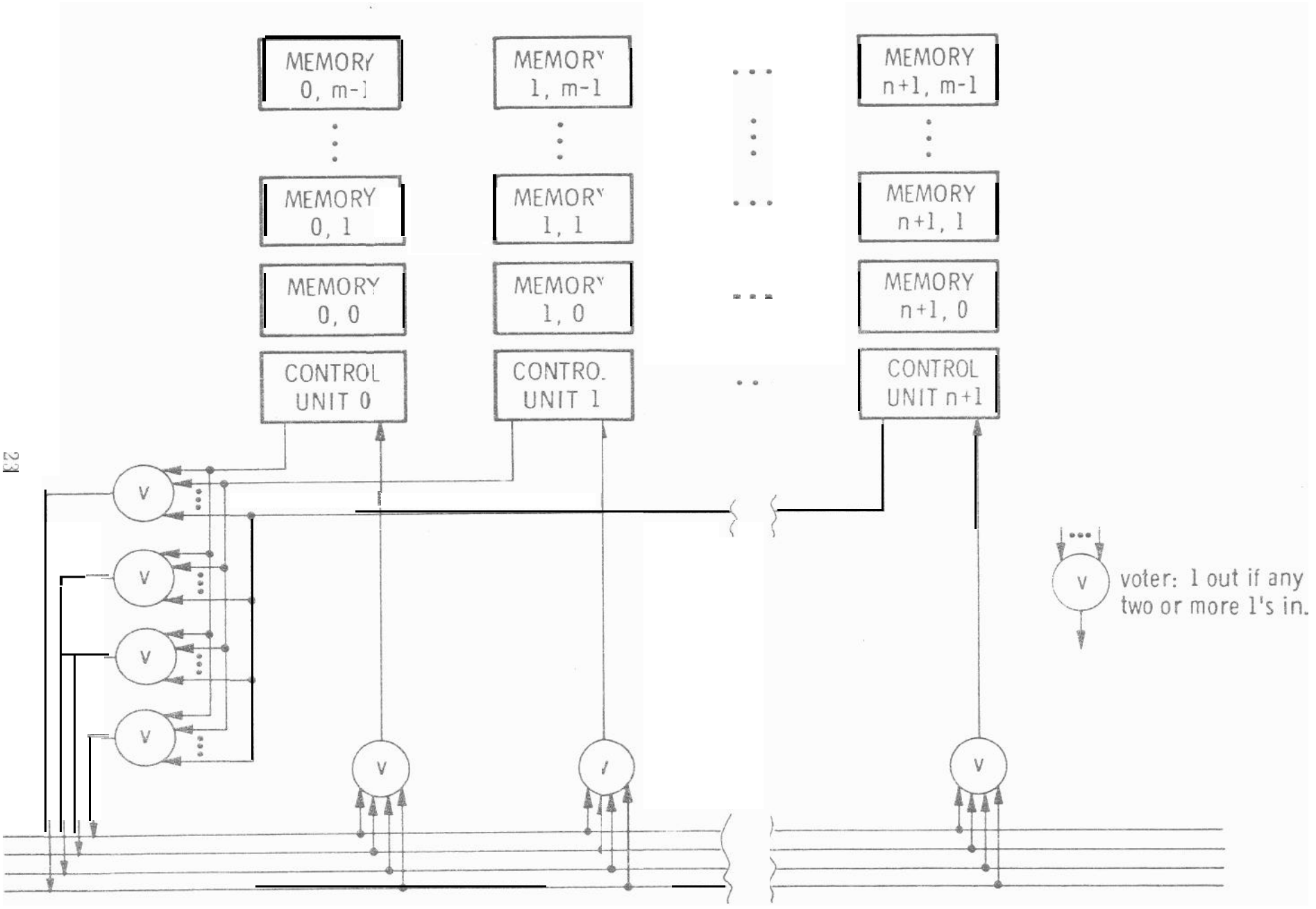

Figure 3 Memory Organization with Preassigned Modules

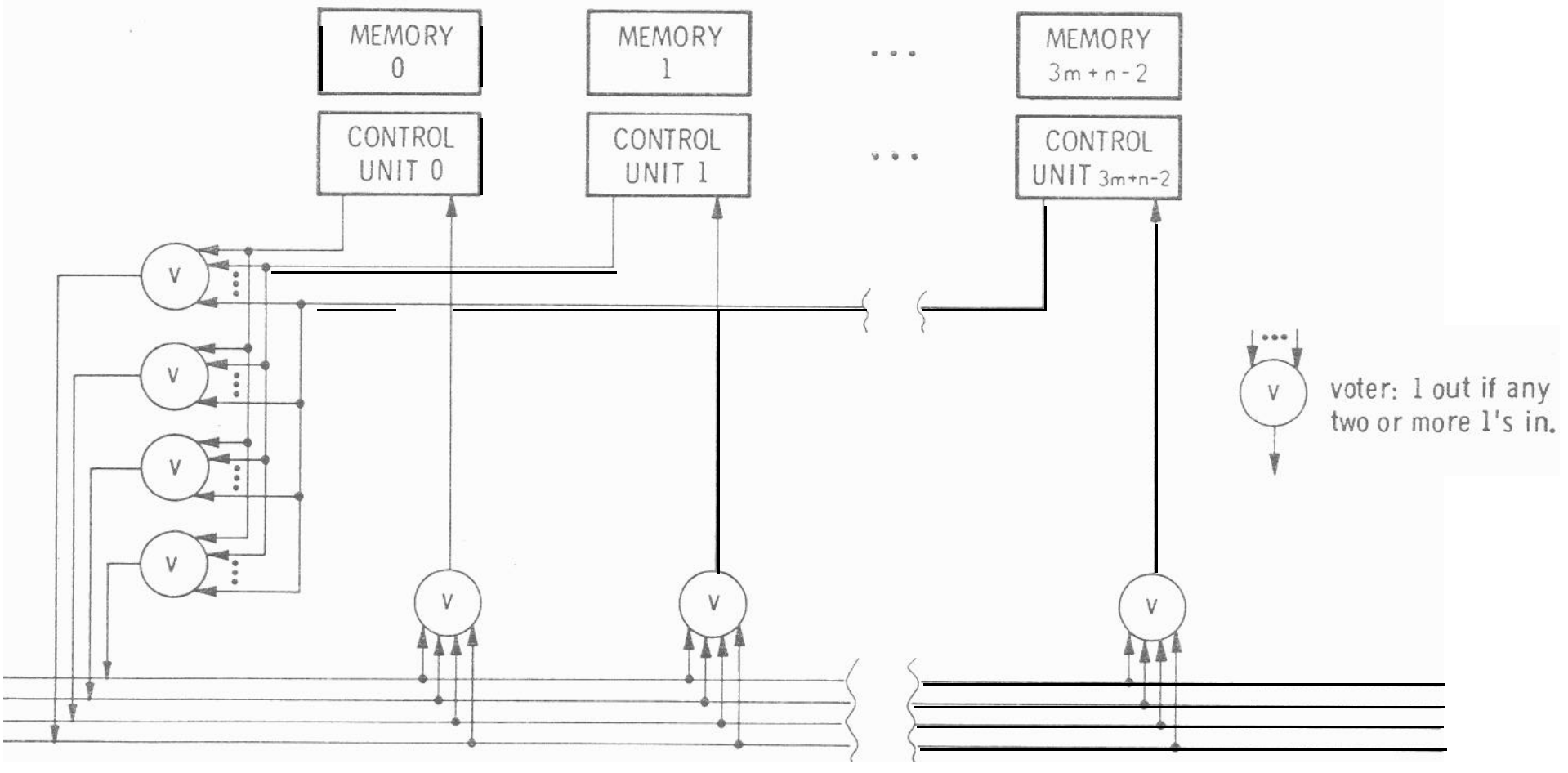

Figure 4 Memory Organization Using Assignable Modules

REFERENCES

[1] W.H. Pierce, Failure-Tolerant Computer Design, Academic Press, New York, 1965, page 1.

[2] G.Y. Wang, "System Design of a Multiprocessor Organization", NASA Electronics Research Center Internal Technical Memorandum RC-T-079, Cambridge, Massachusetts, 1969.

[3] A. Avizienis, "Design of Fault-Tolerant Computers", 1967 Fall Joint Computer Conference, AFIPS Proc., Vol. 31, Thompson Books, Washington, D.C., 1967, pp. 733-743.

[4] A.L. Hopkins, A.I. Green, et al, "A Fault-Tolerant Information Processing System for Advanced Control, Guidance, and Navigation", MIT Charles Stark Draper Laboratory Report R-659, Cambridge, Massachusetts, May 1970.

[5] W.G. Bouricius, W.C. Carter, J.P. Roth, and P.R. Schneider, "Investigations in the Design of an Automatically Repaired Computer", Digest of the First Annual IEEE Computer Conference, IEEE, New York, 1967, pp. 64-67.

[6] A.I. Green; et al, "STS Data Management System Design", MIT Charles Stark Draper Laboratory Report E-2529, Cambridge, Massachusetts, June 1970.

R-682

DISTRIBUTION LIST

Internal:

Eldon Hall
A. Hopkins (100)
D. Bowler
R. Filene
J. McKenna
A. Green
H. Blair -Smith
D. Hanley
J. Martin
J. Partridge
G. Schwartz
F. Alarcon
K. Griggs
F. Gauntt
W. Weinstein
J. Allen
M. Johnston
W. Daly
R. Tavan
G. Jones
R. Scott
R. Ragan
D. Hoag
N. Sears
J. Gilmore
R. McKern
R. O'Donnell (KSC)
T. Lawton (MSC)
J. Hand
J. Harrison
G. Edmonds
G. Silver
A. Laats
R. Crisp
K. Fertig
D. Cox
S. Cohen
E. Duggan
V. DeMarco
P. Mimno
J. H. Laning
P. Adler
J. Kernan
M. Hamilton
D. Densmore
L. Quagliata
N. Pippenger
K. Glick
J. Nevins
R. Metzinger
K. Sorenson
J. Scanlon
A. Woodin
J. Barker
R. Cushing
R. Battin
s. Copps
D. Fraser
G. Levine
R. Larson
P. Felleman
T. Fitzgibbon
P. Bowditch
R. Weatherbee
Apollo Library (2)
CSDL/TDC (10)

External:

NASA/RASPO (1)

DELCO Electronics (3)

Kollsman (2)

Raytheon (2)

MSC: (35&1R)

National Aeronautics and Space Administration
Manned Spacecraft Center
Houston, Texas 77058
ATTN: Apollo Document Control Group (BM 86) (18&1R)

M. Holley	(2)	G. Xenakis	(1)
T. Gibson	(1)	T. Gibson	(1)
C. Frasier	(1)	H. Tindall	(1)
P. Sollock	(1)	S. House	(1)
T. Chambers	(1)	C. Levy	(1)
C. McCullough	(1)	K. Cox	(1)
J. Hughes	(1)	R. Chilton	(1)
E. Chevers	(1)	R. Gardiner	(1)

KSC: (1R)

National Aeronautics and Space Administration
J. F. Kennedy Space Center
J. F. Kennedy Space Center, Florida 32899
ATTN: Technical Document Control Office

LRC: (2)

National Aeronautics and Space Administration
Langley Research Center
Hampton, Virginia
ATTN: Mr. A. T. Mattson

GA: (3&1R)

Grumman Aerospace Corporation
Data Operations and Services, Plant 25
Bethpage, Long Island, New York
ATTN: Mr. E. Stern

NAR: (8&1R)

North American Rockwell, Inc.
Space Division
12214 Lakewood Boulevard
Downey, California 90241
ATTN: CSM Data Management
D/096-402 AE99

NAR RASPO: (1)

NASA Resident Apollo Spacecraft Program Office
North American Rockwell, Inc.
Space Division
12214 Lakewood Boulevard
Downey, California 90241

GE: (1)

General Electric Company
Apollo Systems
P. O. Box 2500
Daytona Beach, Florida 32015
ATTN: E. P. Padgett, Jr. /Unit 509