J. R. Sklaroff

Redundancy Management Technique for Space Shuttle Computers

Abstract: This paper describes how a set of off-the-shelf general purpose digital computers is being managed in a redundant avionic configuration while performing flight-critical functions for the Space Shuttle. The description covers the architecture of the redundant computer set, associated redundancy design requirements, and the technique used to detect a failed computer and to identify this failure on-board to the crew. Significant redundancy management requirements consist of imposing a total failure coverage on all flight-critical functions, when more than two redundant computers are operating in flight, and a maximum failure coverage for limited storage and processing time, when only two are operating. The basic design technique consists of using dedicated redundancy management hardware and software to allow each computer to judge the "health" of the others by comparing computer outputs and to "vote" on the judgments. In formulating the design, hardware simplicity, operational flexibility, and minimum computer resource utilization were used as criteria.

Introduction

The Space Shuttle avionics system contains five identical general purpose digital computers, each capable of communicating with the avionic subsystems to perform flight-critical and non-critical functions. During time-critical mission phases (i.e., recovery time less than one second), such as boost, reentry, and landing, four of these computers operate as a redundant set, receiving the same input data, performing the same flight-critical computations, and transmitting the same output commands. (The fifth computer performs non-critical computations.) In this mode of operation, comparison of output commands and "voting" on the results in the redundant set provide the basis for efficient detection and identification of two flight-critical computer failures. After two failures, the remaining two computers in the set use comparison and self-test techniques to provide tolerance of a third fault. This paper describes the computer set configuration, its operation in collecting and transmitting data, the redundancy management requirements, design considerations in meeting the requirements, and the actual design implementation.

The Space Shuttle represents the first planned operational use of multiple, internally simplex computers to provide continuous correct system operation in the presence of computer hardware failures. The concept was flight-proven in the Tactical Aircraft Guidance (TAGS) research and development program [1], which demonstrated a helicopter flight control system using three simplex digital computers to provide single-fault tolerance. The F-8 aircraft research and development program [2], currently under way, also uses three digital computers in a triply redundant flight control system. By comparison, previous space programs used either specially designed, internally redundant computers or multiple computers in a prime-backup configuration to provide fault-tolerant operation. The Saturn IB and Saturn V launch vehicle digital computer was a triply redundant design providing redundancy at a modular level within the computer [3]. The Orbiting Astronomical Observatory processor was four-fold redundant at the circuit level. The Skylab program used dual computers in an active/standby mode (in-orbit), relying on self-test techniques to detect failures in the active computer, and redundant hardware to switch to the standby computer (up to 2.75 s switchover time) [4].

Internally redundant computers use considerable extra circuitry to provide the required fault tolerance for continuous system operation [5]. This circuitry also provides the means to detect failures by making logical comparisons at selected points in the data flow within the computer. The concept of redundancy, when employed at the computer unit level, uses comparisons of data generated at the normal computer interface and thus does not incur the cost of special computer or circuit development. The Space Shuttle avionics design, constrained to the use of standard, off-the-shelf comput-

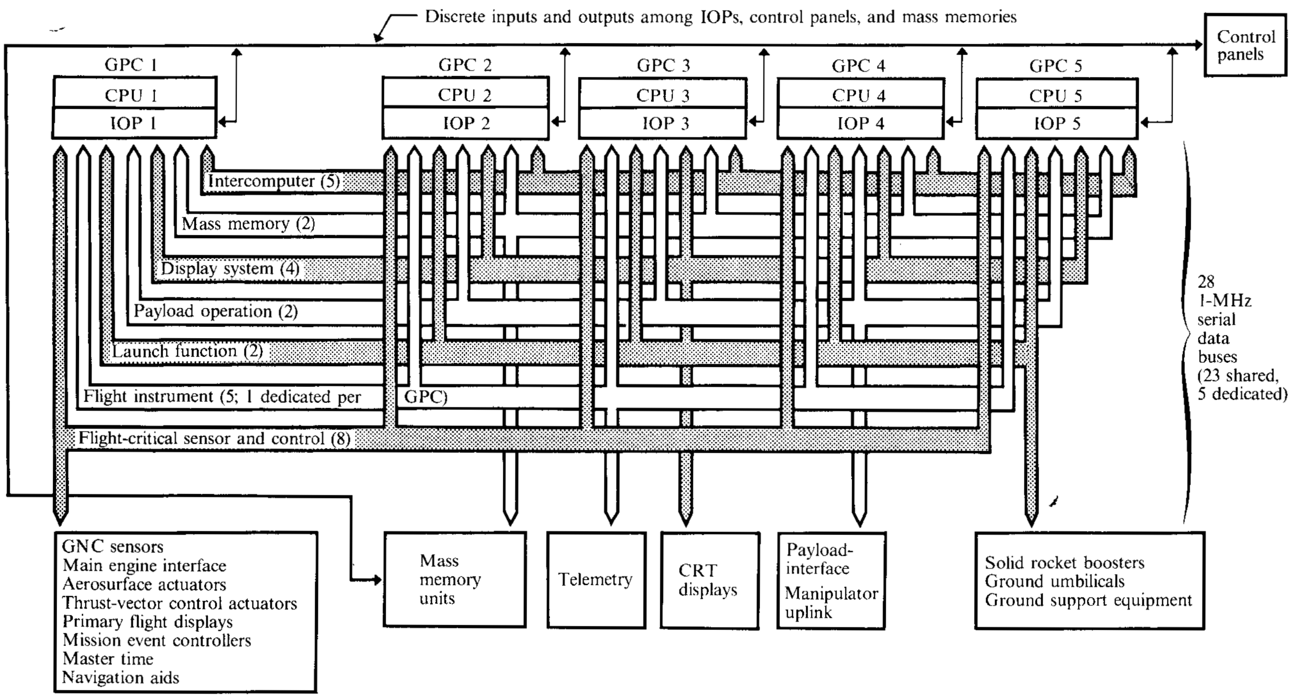

Figure 1 Space Shuttle avionics system block diagram.

ers, presented a unique engineering opportunity to properly utilize redundance at the unit level to provide the necessary failure coverage and fault tolerance.

System configuration

The digital processing subsystem defined for the Space Shuttle [6, 7] contains five general purpose computers (GPCs) communicating with the avionic subsystems over serial data buses (Fig. 1). Four of the five GPCs are identically programmed to perform flight-critical functions, such as guidance, navigation, and control. The fifth computer is programmed to perform non-flight-critical avionic functions.

A GPC consists of an IBM AP-101 central processor unit (CPU) and an input/output processor (IOP). Each IOP is transformer-coupled to the buses, and can transmit or receive 1-MHz serial digital data over each of 24 bus channels. The data buses, in turn, are transformer-coupled to multiplexer/demultiplexer units (MDMs) and digital subsystems. The MDMs (containing analog-to-digital and digital-to-analog converters) interface with analog subsystems, such as flight control sensors and effectors.

Subsystems performing similar functions are assigned to the same data-bus group. (There are seven such groups in total.) Subsystems have varying levels of redundancy at the unit level, depending on their criticality; e.g., there are three inertial measurement units, two radar altimeters, and four air data transducer assemblies. Each unit is addressable by a command word over the bus. To prevent the loss of more than one redundant unit when one data bus fails, no two redundant units interface with the same bus.

Some subsystems are internally redundant, such as the hand controllers and keyboard units. Also, all safety-of-flight critical effector subsystems, such as the actuators for the main engine and for the aerosurfaces, the main engine interface units, and mission event controllers are internally redundant at different levels. Such subsystems receive redundant commands on separate input channels and, using internal algorithms, generate one output. The algorithms detect incorrect commands and eliminate such commands from consideration in the output.

An example of a four-input "voting" effector is the aerosurface actuator, which uses four independent servo channels driving a four-element force-summed actuator. Failure of any three of the four channels can be tolerated without loss of operational capability. Hydraulic fault detection is provided by sensing the pressure differential at each element of the secondary actuator. With more than two channels operating, the actuator element is automatically bypassed when the threshold pressure differential is exceeded for a given time. With two channels operating, no actuator element is bypassed when the pressure differential is exceeded, thereby producing a "standoff" until one of the channels is manually reset.

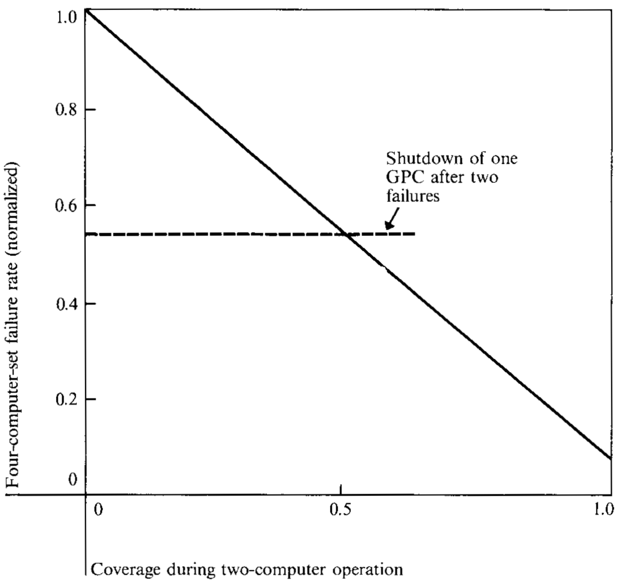

Figure 2 Effect of coverage during two-computer operation on the failure rate of a four-computer redundant set; coverage of the first two failures is one. Normalization is to the zero-coverage failure rate. The GPC failure rate is 8 × 10^{-2} per hour.

The use of voting effectors provides, in effect, downstream protection against failures in the redundant computer set. It allows a computer to transmit incorrect commands to critical subsystems for an indefinite number of cycles without actually having adverse effects on system operation. Thus, it is not necessary to detect computer failures immediately and stop the transmission of the incorrect output. This feature significantly relaxes the failure detection time constraint, which is one of the requirements that control the selection of the redundancy management technique.

Another feature of the system that facilitates the selection of a redundancy management technique is the interconnection of buses to computers. This feature allows each computer to have access to all flight-critical data received or transmitted by the other computers and makes possible the comparison of identical computations among computers.

System operation

Each bus within a data-bus group is assigned, under software control, to operate in either a command or a listen mode. In the command mode, data requests and commands are issued to the subsystems over the bus and data are received over this same bus. In the listen mode, data are only received on the bus.

In the flight-critical sensor and control-data-bus group (two subgroups of four buses), one bus in each subgroup is assigned to operate in the command mode (in each redundant-set computer) and the remaining three are assigned to operate in the listen mode. In the inter-computer channel (ICC) data-bus group (five buses), one bus (in each computer) is in the command mode and the remaining four are in the listen mode.

For data collection, since each of the redundant subsystems is connected to a different bus, a different computer requests data from each of the subsystems and the returned data are available to all other computers in the set. The listening computers are alerted that the subsystem data are available by receiving a listen command, which is issued by the command computer just prior to issuing the data request command to the subsystem. When operated in this manner, identical input data are available to each computer in the redundant set.

For data output, since each channel of the effector subsystem is connected to a different bus of the group, a different computer transmits command data to each of the voter-effector channels. Thus, a voter-effector subsystem requiring four inputs receives inputs from four different computers. Also, since the buses are interconnected to all computers, the capability exists for each computer to listen to the command data sent out by each of the other computers.

For inter-computer information transfer, each computer communicates with all other computers, passing data to the others, requesting data from the others, and performing any other tasks required to operate the computer set. No subsystem is connected to the ICC buses.

As a consequence of the distributed control of redundant sensors among computers, an unacceptable time-skew may exist between redundant inputs unless the computers are synchronized prior to initiating the inputs. Similarly, unacceptable data-skew may exist at the voting effectors unless synchronization occurs prior to initiating outputs. Moreover, unacceptable command differences may exist at the voting effectors unless synchronization occurs at appropriate points during program execution. Synchronization is accomplished in the Space Shuttle computer set by using inter-computer discrete signals and synchronization software.

Program synchronization is required because computers that do not use exactly the same data for computing flight-control outputs experience command divergence [8]. The time required to synchronize program execution depends on the design of the flight software operating system. A fixed time-slice system (i.e., one in which all processes are run within a given cycle time) requires one synchronization point in each computational cycle. An interrupt-driven system must synchronize at all points at which data are calculated in one process and used in another, and at all points needed to preserve identical process sequences in all computers of the set.

Redundancy management design

- *Requirements*

Five redundancy management (RM) design requirements established for the GPCs are presented and discussed below.

1. A failed computer must be identified to the crew, prior to assignment to the redundant set, using self-test techniques that provide at least 96 percent coverage. (Coverage is defined as the probability of detecting a failure, given that a failure has occurred.)

The purpose of this requirement is to enable the crew to identify a failed computer (a) before lift-off (to abort the mission) or (b) before a critical in-orbit phase (to reallocate the data bus assignments or to attempt to restore a GPC using an initial program load (IPL) from mass memory). A goal of 96 percent coverage of computer failures in an autonomous environment (i.e., no external test equipment or cooperative use of other GPCs) was established. A computer fail-discrete interface with the crew panel suffices for GPC failure identification.

2. Of four GPCs in the redundant set, the first two to fail and cause incorrect flight-critical output must be automatically identified to the crew as failed; the third should also be identified as failed, but only by achieving as much coverage as is possible within a limited processing and storage overhead.

The purpose of this requirement is to enable the crew to take appropriate action during flight based on complete knowledge of the failure status, such as aborting the mission, de-energizing the computers, or reallocating bus assignments. Moreover, in the case in which a critical failure occurs when only two GPCs are operating, if the crew can reconfigure accordingly, the voting effectors will continue to operate without simply arriving at a standoff.

The relation between failure coverage during a two-computer operation and the probability of an N-unit system loss is presented in the Appendix. The result shows that for a system which achieves total coverage on the first $N - 2$ failures, system loss is a linear function of coverage during the two-unit operation and is the same as in the one-unit operation when the coverage is 0.5. Thus, for any coverage of more than 50 percent, the probability that the redundant set will fail when two computers are operating is less than the failure probability when only one is operating.

The result in the Appendix has been applied to a typical Shuttle mission. The graph shown in Fig. 2 illustrates the importance of providing adequate coverage in the two-computer operation. If two computers are allowed to operate until one critically fails and is removed 100 percent of the time (coverage = 1), the probability that the computer set will fail is 5.5 times lower than if one fails and is removed only 50 percent of the time (coverage = 0.5). Also, if two computers are allowed to operate without ever removing one that fails, the computer-set failure rate is 5.5 times greater than if one computer is arbitrarily shut down.

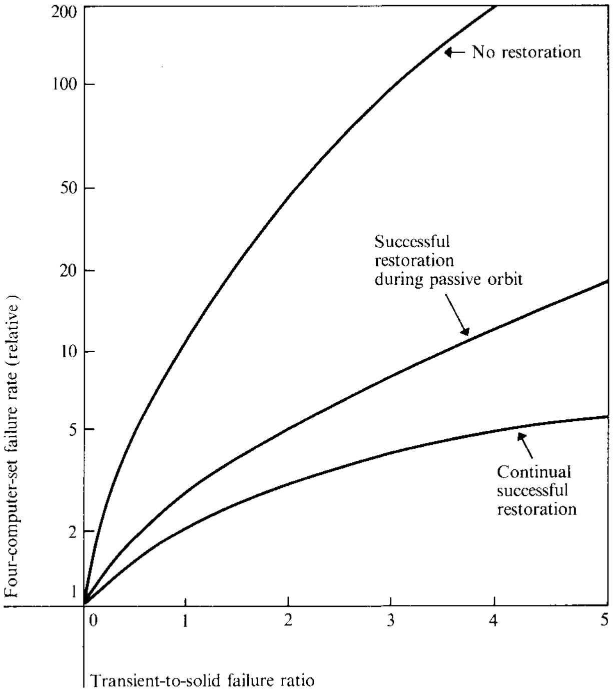

Figure 3 Effect of GPC transient failures on the computer-set failure rate as a function of restoration policy, in relation to the zero-transient failure rate. The operational-hour equivalents are 6.6, ascent; 128, passive orbit; 43, active orbit; and 4.4, reentry/landing.

3. A GPC indicated as failed should be capable of automatically inhibiting all transmission, but this capability must be enabled by the crew.

The purpose of this requirement is to provide the crew with the option of automatic reconfiguration of the redundant set.

4. A GPC failure should not cause another GPC to either identify itself as failed or generate an incorrect output.

The purpose of this requirement is to minimize the occurrence of (a) a non-failed computer being identified as failed because of the redundancy management design implementation and (b) a failed computer causing another computer to execute an incorrect program.

5. Restoration of a GPC in which a permanently incorrect critical output exists because of a transient failure should be accomplished wherever practical.

The purpose of this requirement is to minimize the impact of transient GPC failures on system operation. Figure 3 illustrates the effect that such transients have on the number of computer-set losses, depending on the restoration policy used. If restoration is never attempted, computer-set losses increase by a factor of 16 over those of a continual restoration policy at a 2:1 "transient-to-solid" (i.e., intermittent-to-permanent type) failure ratio, and by 60× at a 4:1 ratio. However, if computer restoration takes place only during passive orbit (non-time-critical phase), computer-set losses increase only 1.6× over a continual restoration policy at a 2:1 transient-to-solid failure ratio, and 2.8× at a 4:1 ratio. For a continual restoration policy, an undue amount of computer resource and operational complication is involved in restoring and adding a computer to the redundant set during time-critical mission phases. Consequently, the redundancy management implementation need not automatically attempt restoration on detecting a computer failure. Instead, restoration can be attempted under crew control during passive orbit, when no maneuver takes place.

• *Considerations*

Several factors must be considered in developing computer redundancy management techniques to satisfy the requirements listed in the previous section. One of the more significant considerations is whether faults (hardware defects) or manifestations of faults to the output interfaces should be detected. Detection of faults is helpful from a hardware maintenance standpoint, but is of little practical operational value. A computer fault is of interest when the effect of the fault shows up in an operational output. Therefore, redundancy management design for the Space Shuttle should be oriented to the detection of failures that affect commands on the flight-critical buses.

Another important consideration is the amount of resource needed to attain the total coverage required for the one- or two-failure case. Such coverage cannot be attained in single off-the-shelf computers, as specified for the Space Shuttle. Therefore, cooperative testing techniques must be established among the computers. Comparison of identical computations is one method of attaining high coverage and is practical in the Shuttle computer set, without a significant amount of additional hardware, as a consequence of the normal system operation of command and listen modes, synchronization, and grouping of buses.

Other design decisions and rationales in developing the redundancy management technique are as follows.

1. Minimize the need to depend on the computer to detect and identify its own failure by using special hardware logic dedicated to the RM process.

Even though the comparison-of-output technique can detect a very high percentage of failures, identification is not certain if the failed GPC is simply notified that it has failed. Therefore, non-failed computers in the set must have the capability of forcing the failed computer to indicate itself as having failed. This requires hardware dedicated to the redundancy management identification process.

2. Use computer software to judge the "health" of other computers.

If a computer cannot correctly monitor and test other computers, then it probably cannot compute correct output itself and will be judged bad by the others. Thus, the output-compare process need not be a part of the RM hard logic, but may be done by software.

3. Use sum-checking of critical outputs as the comparison basis.

A large range of output-compare data exists. Each separate word transmitted on the flight-critical buses could be compared, but this would result in considerable computer and bus overhead. A more effective method of reducing overhead, without losing the required coverage, is to sum the outputs to be transmitted over the flight-critical buses during one computational cycle and then compare the sum-check.

4. Transmit the compare word over the ICC buses.

Eight flight-critical buses and the ICC buses are available for transmitting the selected compare word to the other computers. Input/output processor transmission and reception on the flight-critical buses are checked when transmitting commands to the sensors and receiving data from the sensors using special communication tests, e.g., IOP BITE (built-in test equipment) and bus channel timeout. The ICC buses are available for use without interfering with the critical bus traffic.

5. Detect faults in dedicated RM hardware by programmed testing of the logic.

Faults in any redundancy management logic added to the individual computers do not propagate to the flight-critical channels and show up as critical failures. However, faults in this logic are critical, and it is important that they be detected in order to satisfy design requirement 4. Hence, faults in this logic should be detected by a test program executed in each CPU.

Design implementation

The hardware and software elements developed for the Space Shuttle and used to meet the redundancy management requirements are listed in Table 1. The elements and their operational use support failure detection, identification, and reconfiguration. These functions are performed according to which of the following conditions prevails during flight: (a) four or three GPCs operating in the redundant set (quad or triplex configuration); or (b) two GPCs operating in the redundant set (duplex configuration).

- *Quad/Triplex operation*

When more than two GPCs are operating in the redundant set, the level of achievable coverage can be significantly improved over that provided by self-testing by using cooperative techniques. Moreover, the resource utilized in obtaining this coverage with cooperative techniques is significantly less than that required for self-testing.

Two cooperative tests are performed in the Shuttle to detect a failed GPC when more than two GPCs are in the redundant set. These are the compare word test and the bus channel timeout test.

The *compare word test* consists of

a. Computing a compare word by summing critical GPC command outputs (to the effectors);
b. Transmitting the compaie word on the ICC bus to the other computers in the RM set;
c. Receiving compare words from other computers and storing them in designated locations in main memory;
d. After a time has elapsed which ensures receipt of all compare words, comparing the received words with the computed word on a bit-by-bit basis; and
e. Identifying another GPC as failed if two successive non-compares are assigned to it.

The compare word test, performed after the critical GPC commands have been computed in each computation cycle, is implemented using both CPU and IOP software. The test uses less than 0.2 percent of the CPU processing capability and achieves nearly total coverage.

The *bus channel timeout test* consists of waiting a given time after synchronization for another computer to perform an input transaction on its dedicated bus. This test, plus the compare word test, provides total coverage in the computer set of all flight-critical functions.

The compare word test uses the dedicated redundancy management hardware, shown in Fig. 4, for cooperative identification as follows:

Failure-vote register (FVR) Each IOP contains a four-bit register, which is used to judge another GPC as failed as the result of a cooperative test. When a failure is detected, a bit is set in this register. Each position of the FVR is associated with another GPC within the computer set. The FVR is set and reset under control of the software and is automatically reset for any reset of the IOP.

Table 1 Summary of the elements of redundancy management.

Function	*Element*
Fault detection	Compare word test Bus channel timeout test Built-in test equipment Self-test programs Watchdog timer
Fault identification	Failure-vote-discrete out GPC-fail-discrete out
Reconfiguration	Failure-vote inhibit, voter- and timeout-IOP reset (RM-initiated) IOP-master reset (software-initiated) IOP-fail reset (BITE-initiated) Power off and halt-discrete in (crew-initiated)

Failure-vote discrete driver/receiver Each output of the FVR is logically connected to an independent, discrete driver. Each failure-vote discrete driver has two dedicated outputs, one to indicators on the crew panel and the other to a discrete receiver in the IOP of one of the GPCs (the one corresponding to the dedicated FVR position). Thus, each GPC contains four failure-vote discrete drivers for transmission of the results of the compare word test to other GPCs (and to the crew panel) and four failure-vote discrete receivers for acceptance of the votes.

Voter logic and fail-latch The voter logic accepts the outputs of the fail-vote discrete receivers and generates an output to set a voter fail-latch whenever two or more failure-vote discretes are received from other GPCs. The voter fail-latch is reset by an IOP power-on reset or a halt-discrete input from a crew panel switch and is set by the voter logic. The output of the latch is transmitted to the computer-fail driver, failure-vote inhibitor, and transmission termination control logic.

Computer-fail driver (fail-discrete) During flight the computer-fail driver is used to transmit a GPC critical failure indication to the controls and display console as a result of both cooperative and self-test GPC failure identification. When the voter fail-latch is set, the computer-fail driver transmits an output discrete to the crew panel. No fail indication is present when a GPC is on and operating normally, is off, or has received a halt-discrete input.

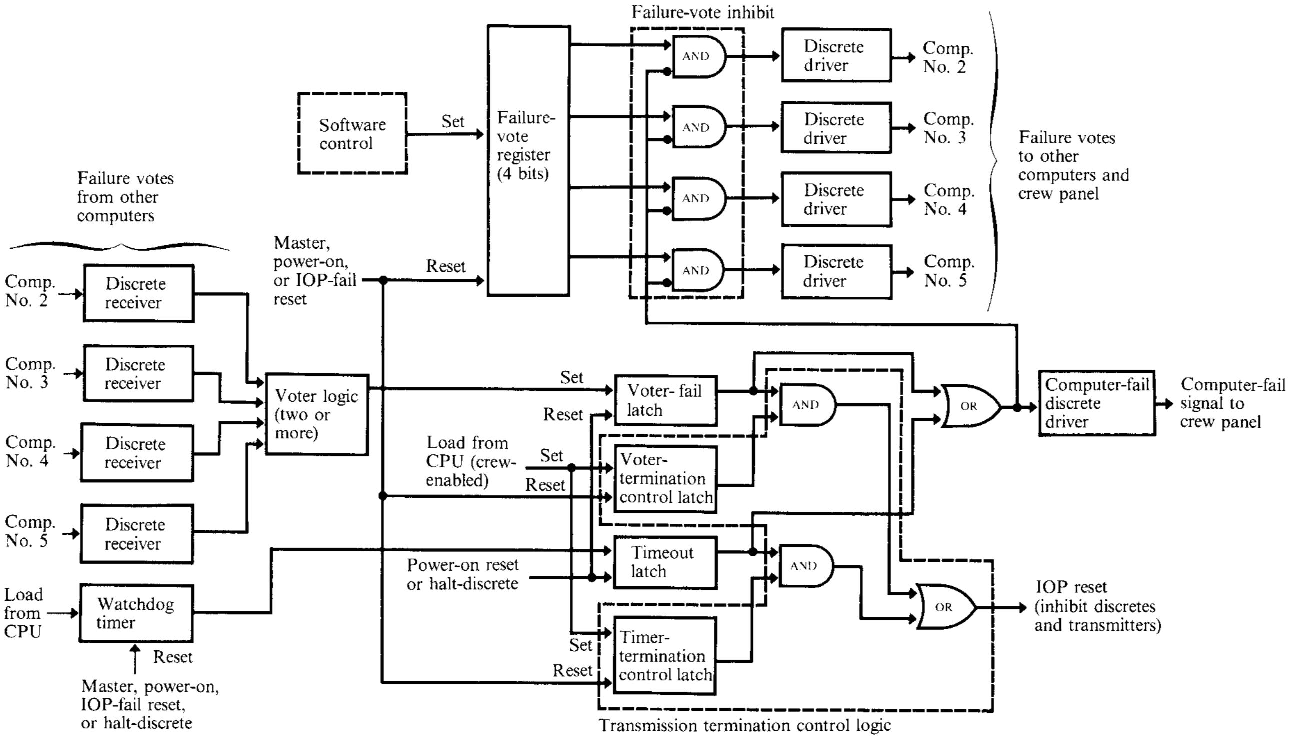
Failure-vote inhibit
AND
AND
AND
AND
Discrete driver
Discrete driver
Discrete driver
Discrete driver
Comp. No. 2
Comp. No. 3
Comp. No. 4
Comp. No. 5
Failure votes to other computers and crew panel
Software control
Set
Failure-vote register (4 bits)
Failure votes from other computers
Master, power-on, or IOP-fail reset
Reset
Comp. No. 2
Discrete receiver
Comp. No. 3
Discrete receiver
Comp. No. 4
Discrete receiver
Comp. No. 5
Discrete receiver
Voter logic (two or more)
Set
Reset
Voter- fail latch
Load from CPU (crew-enabled)
Set
Reset
Voter-termination control latch
AND
OR
Computer-fail discrete driver
Computer-fail signal to crew panel
Timeout latch
Power-on reset or halt-discrete
AND
OR
IOP reset (inhibit discretes and transmitters)
Load from CPU
Watchdog timer
Reset
Master, power-on, IOP-fail reset, or halt-discrete
Set
Reset
Timer-termination control latch
Transmission termination control logic

Figure 4 Dedicated redundancy management logic, shown for computer 1.

The hardware shown in Fig. 4 is also used for reconfiguration, which involves termination of the role of a GPC as a functioning member of the redundant GPC set. One manifestation of reconfiguration is an IOP reset, which can occur automatically when the GPC has been identified as failed; another is automatic inhibition of the transmission of failure votes to other GPCs, which means that the logic operates to prevent a failed GPC from further participation in cooperative failure detection.

- *Duplex operation*

When only two GPCs are operating in the redundant set, identification as a result of cooperative testing is not possible, although detection of a disagreement is possible through the compare word test. Therefore, self-testing must be used to distinguish which of two GPCs has failed. The cooperative tests are still useful, however, in providing a program cue to scheduling any planned self-test program execution. As such, cooperative tests should be run continuously in the duplex mode. Thus, whenever a failure vote is set by either of the two computers, that computer can also run self-test.

The sources of self-test coverage that can be used in the duplex mode fall into three categories:

BITE (*built-in test equipment*) BITE consists of any hardware logic or microcode which causes an interrupt to the CPU, during its normal program operation, when a GPC failure is detected.

Self-test programs These programs detect GPC failures, but must be specifically executed under program control. They can consist of either main-memory-resident (macrocode) or control-storage-resident (microcode) routines.

Watchdog timer A watchdog timer (shown in Fig. 4), located in the IOP and periodically loaded by the CPU, is used to detect failures that affect cyclic program execution by the CPU and is a means of indicating that the GPC has failed after fault detection by the CPU. Whenever the time loaded into the timer elapses, a timeout latch is set to identify the failure to that GPC.

The level of coverage of critical hardware that can be achieved by self-test techniques during duplex operation is limited by the resource utilization constraints imposed by flight-program budgets. An example of the relation between coverage and resource utilization is shown in Table 2 for the central processor (the example does not include memory or power supply). The level of CPU coverage achievable for the amount of storage and processing time expended, when a combination of BITE, watchdog timer, and microcoded self-test program is used, is considered acceptable. The use of a macrocoded self-testing program in the redundant set during flight operation would take too much storage and processing time for the additional coverage it would provide. However, the execution of a macrocoded self-testing program prior to assignment to the redundant set, to meet the 96 percent coverage requirement discussed previously, is necessary.

Summary

The use of computers in flight-critical applications has imposed a requirement for multiple-fault tolerance in the computer configuration. In the Space Shuttle avionics, multiple, internally simplex digital computers are configured to receive the same input data and calculate the same flight-critical outputs, in order to use voting fault-tolerant control effectors.

The calculation of the same outputs by each critical computer and the synchronization of inputs are used to provide the means of achieving total failure coverage of flight-critical functions for a small computational resource and hardware cost. The technique that was implemented uses each computer to judge the "health" of the others through a bit-by-bit comparison of critical data and an I/O transaction timeout test. The judgement process was implemented in software for greater flexibility, and the vote on the judgements and the generation of the failure indication were implemented in dedicated hardware for greater failure identification reliability.

When more than two computers are operating, the level of failure detection and identification achievable using data comparison between computers and voting techniques is significantly higher than that obtainable using self-testing. When only two are operating, an acceptable level of failure detection and identification can be obtained, within limited storage and processing time, by using built-in test equipment and a watchdog timer on a continual basis and scheduling special test microcode to follow the occurrence of a non-compare.

Acknowledgments

The author appreciates the useful comments received from F. G. Kilmer, H. A. Padinha, R. E. Poupard, and A. R. Stevens. This paper covers work performed under Space Shuttle contract M4J3XMS-483027 between Rockwell International Corporation, Space Division, and International Business Machines Corporation, Federal Systems Division.

Appendix

Here we develop the relation between coverage during two-unit operation and the probability of multiple-unit system loss.

Table 2 Relation between self-test resource utilization and coverage for the central processor (not including memory or power supply).

Central processor self-test coverage source	*Coverage of critical hardware* (percent)	*Storage* (half-words)	*Processing time* (ms)
BITE*	36.9	0	0
BITE/Watchdog timer	68.5	4	0.005
BITE/Timer/Micro-coded self-testing	88.7	14	0.15
BITE/Timer/Micro- and macro-coded self-testing	96.8	110	1.3

*BITE is the abbreviation for built-in test equipment.

The probability of an N-unit system loss, PSL, with total coverage on the first $N-2$ failures, is

$$PSL = \int_0^T P_2(t)\; PSL^*(T-t)\; dt, \tag{A1}$$

where T = mission length; $P_2(t)$ = probability of failing to a two-unit system at time t; and $PSL^*(T-t)$ = probability of system loss during two-unit operation. In turn, PSL^* has been derived [9] as

$$PSL^* = (1-e^{-u}) - [(2c-1)/(2k+1)] \times e^{-u}[1-e^{-(2k+1)u}], \tag{A2}$$

where $u = \lambda_c(T-t)$; λ_c = unit failure rate; c = failure coverage during two-unit operation; $k = \lambda_b/\lambda_c$; and λ_b = failure rate of components that do not affect the unit's function, i.e., the false alarm rate.

Equation (A2) shows that, for this particular system, system loss is a linear function of coverage c. Moreover, this loss, during two-unit operation, is the same as in one-unit operation when $c = 0.5$.

References

1. F. G. Kilmer and J. R. Sklaroff, "Redundant System Design and Flight Test Evaluation for the TAGS Digital Control System," *Proceedings of the 29th Annual National Forum of the American Helicopter Society*, Washington, D.C., May 1973.
2. C. Jarvis, "A Digital Fly-by-Wire Technology Development Program Using an F-8C Test Aircraft," presented at the AIAA 12th Aerospace Sciences Meeting, Washington, D.C., January 1974.
3. F. B. Moore and J. B. White, "Application of Redundancy in the Saturn V Guidance and Control System," AIAA Paper 67-553, *Proceedings of the AIAA Guidance Control and Flight Dynamics Conference*, Huntsville, Alabama, August 1970.
4. "Apollo Telescope Mount Digital Computer Program," Contract NAS 8-20899 Information Document, Part 1, *TR 72-WO-0039*, IBM Federal Systems Division, Huntsville, Alabama 35805, December 1971.

5. H. Hecht, "A Comparison of Fault Tolerant and Externally Redundant Computers," *SAMSO TR 74-66*, Aerospace Corporation, El Segundo, California, January 1974; available as document *AD777166/0* from the U.S. National Technical Information Service, Springfield, Virginia 22151.
6. S. Z. Rubenstein and L. O. Shroyer, "Digital Processing Subsystem for the Space Shuttle," presented at the National Aviation Electronics Conference (NAECON), May 1974.
7. E. A. O'Hern, "Space Shuttle Avionics Redundancy Management," presented at the AIAA Digital Avionics Systems Conference, Boston, April 1975.
8. H. A. Padinha, "Divergence in Redundant Guidance, Navigation and Control Systems," *Proceedings of the ION National Aerospace Meeting*, Washington, D.C., March 1973.
9. D. R. Thomas and F. G. Kilmer, "Redundancy Management Policies for a Dual Redundant Computer Configuration," *TR 75-C65-0013*, IBM Federal Systems Division, Owego, New York 13827, February 1975.

Received February 24, 1975; revised August 15, 1975

The author is located at the IBM Federal Systems Division, Owego, New York 13827.