NASA Contractor Report 145352

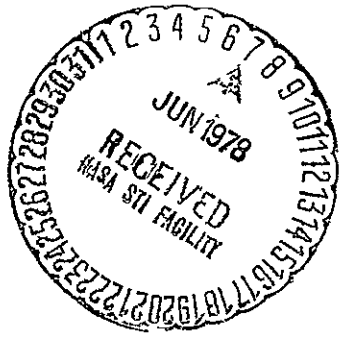
CRITICAL FAULT PATTERNS DETERMINATION IN FAULT-TOLERANT COMPUTER SYSTEMS

E. J. McCluskey and J. Losq

STANFORD UNIVERSITY
Stanford, CA 94025

NASA Grant NSG-1410
January 1978

National Aeronautics and Space Administration
Langley Research Center
Hampton Virginia 23665



DIGITAL SYSTEMS LABORATORY

S T A N F O R D U N I V E R S I T Y

Stanford, California 94305

CRITICAL FAULT PATTERNS DETERMINATION IN FAULT-TOLERANT COMPUTER SYSTEMS

FINAL REPORT

Covering the period: 1 April 1977 - 30 September 1977

NASA Grant no. NSG 1410

SEL Project 24-77

Prepared for the:

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION

Langley Research Center

Hampton, Virginia 23665

Principal Investigator:

Prof. E. J. McCluskey

Project Leader:

J. Losq

ABSTRACT

Digital systems that use redundancy to achieve high reliability are designed so that they recover from most failures. The problem of accurate evaluation of the failure tolerance of any particular redundant system is quite difficult. The methods based on general analytical modeling techniques suffer from the need to make simplifying assumptions. Thus, the confidence that can be granted to their results is low. Some techniques use fault-simulation and try to estimate the size of the failure population that causes system crash by simulating a random sample of failures. However, ultra-reliable systems will tolerate most of the failures, so, very many faults need to be simulated to reach an acceptable level of confidence. The method proposed here tries to enumerate all the critical fault-patterns (successive occurrences of failures) without analyzing every single possible fault. From the system description, one can find all the output devices that allow the system to communicate with the outside world. From the description of these output devices, one can find the conditions that must be satisfied for the system to be operating correctly. Also, one can enumerate all the possible operating modes according to their criticality (high, low, or no tolerance to subsequent failures) and list the corresponding conditions. Most highly redundant systems are provided with capabilities to disable some part of the system or , at least, to ignore data coming from them. These constitute switches and they control the system configuration. At any point in time, the list of the faulty units and the actual system configuration give the static state of the system. The conditions for the system to be operating in a

given mode can be expressed in term of the static states. Thus, one can find all the system states that correspond to a given critical mode of operation. The next step consists in analyzing the fault-detection mechanisms, the diagnosis algorithm and the process of switch control. From them, one can find all the possible system configurations that can result from a failure occurrence. Thus, one can list all the characteristics , with respect to detection, diagnosis, and switch control, that failures must have to constitute critical fault-patterns. Such an enumeration of the critical fault-patterns can be directly used to evaluate the overall system tolerance to failures. Present research is focused on how to efficiently make use of these system-level characteristics to enumerate all the failures, defined at the gate level (for example line i stuck-at-one), that verify these characteristics.

I. CRITICAL FAILURES AND REDUNDANT SYSTEMS

The purpose of using massive redundancy in computer systems is to protect them from the effects of internal failures. When several computers perform the same computations in tight synchronism, it is possible to detect the occurrence of a computer failure by comparing the computer outputs. The faulty computer is then switched off and it is said that the system has recovered from the failure. As first approximation, one can say that system failure, also referred to as system crash, occurs only when there are not enough fault-free computers left to allow a meaningful comparison. For example, Triple Modular Redundant systems (also called TMR systems) [von Neumann; 1956], Fig. 1, fail upon the second failure occurrence: the majority voter will not produce the correct output when two of its inputs are faulty.

But even for systems as simple as TMR systems, the actual system behavior in presence of failures is far more complex. Some of the failures that affect the voter, (cf. Fig. 1) cause a system crash; for example, a stuck-at-one failure in the OR gate (line a_7, a_8, a_9, or a_{10} stuck-at-one). Also some double failures will be tolerated. For example, if the output of computer 1 (line y_1) is stuck-at-zero while the output of computer 2 (line y_2) is stuck-at-one, the system will still operate correctly since the faulty logic one on line y_2 compensates for the faulty logic zero on line y_1 in the voting process. This phenomenon is known as compensating failures. It was shown in [Siewiorek; 1971] that many double failures are compensating failures and that the actual reliability performance of TMR systems is far better than what is estimated when compensating failures are ignored.

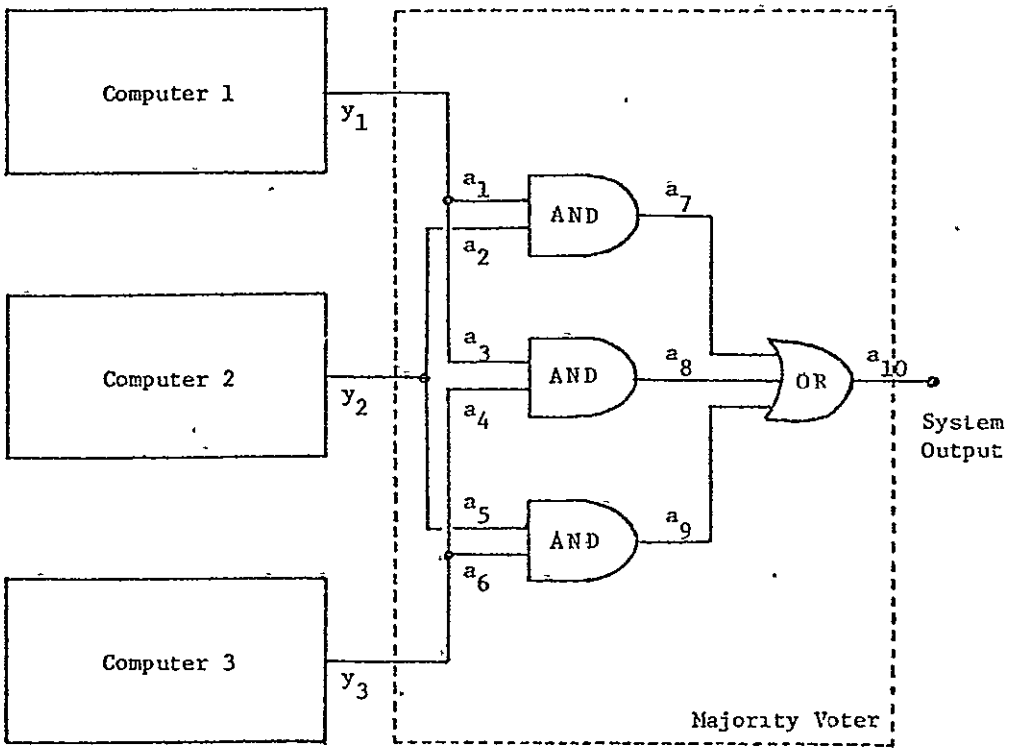


Fig. 1. T.M.R. Redundant System.

For the simple system of Fig. 1 , one can enumerate all single and double failures from which the system does not recover. The uncoverable single and double failures are respectively the most and the second most critical failures. For example, the set of uncoverable single failures is:

{line a_7 s-a-1, line a_8 s-a-1, line a_9 s-a-1, line a_{10} s-a-1}.

The set of uncoverable double failures is quite large, even for such a simple example. Some simplifications can be obtained if one uses the various relations that exist between failures, for example fault equivalence and fault dominance [Boute; 1972].

Most actual systems, however, are far more complex than the oversimplified example of Fig. 1. Pure enumeration of all the faults and their classification according to their respective criticality is a prohibitive task. In most redundant systems, the problem of finding the critical failures is further complicated by the fact that each computer has some internal fault-detection mechanisms (self-test programs, parity verification, watchdog timers,...) and some way to disable itself when a fault-detection mechanism gives some error indication. The ability for a computer to detect some of its failures without requiring comparison between several computers greatly improves the overall system reliability but also greatly complicates the system analysis. For each fault one needs to consider the state of the faulty computer (whether the fault was locally detected, correctly diagnosed and the proper reconfiguration steps taken) and also the global state of the system (the decision taken by the fault-free computers and their agreement with the state of the faulty computer).

II. DESCRIPTION OF THE AIRBORNE ADVANCED RECONFIGURABLE COMPUTER SYSTEM

II.1 Introduction

The Airborne Advanced Reconfigurable Computer System -the ARCS- [Bjurman; 1976] is an integrated navigation/guidance/flight control system for commercial aircrafts. The system is a Triplex-to-Duplex-to-Simplex fault-tolerant computing system with the capability to be expanded to a Quadruplex-to-Triplex-to-Duplex-to-Simplex mode of operation. Fig. 2 shows a block diagram of the ARCS.

II.2 Channel Description

A channel in the ARCS terminology refers to a computer with its associated sensor and servo equipment. A channel contains all the electronics and the mechanical component to fully implement all the navigation, guidance and flight control operations. Thus, a single channel can fly the aircraft. The ARCS contains three channels (four in the expanded version).

II.2.1 Channel Components

The components within each channel are: sensors and mode controls, computer unit, iterative timer and watchdog monitor, servo electronics, servo monitor, and switch and servo actuator.

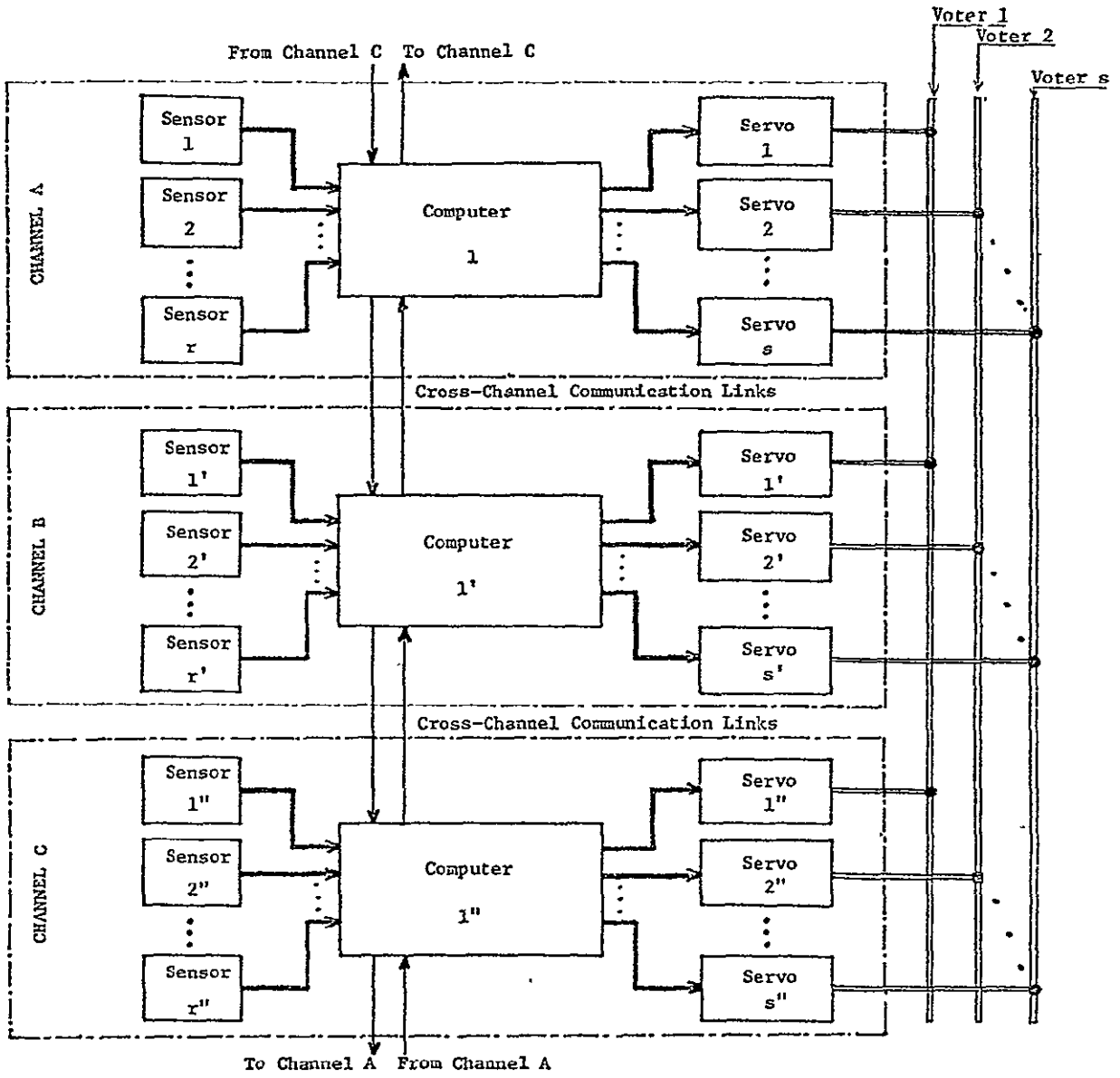


Fig. 2. General concept of ARCS.

II.2.1.1 Sensors and Mode Control

These are inputs to the computer unit. The sensors provide data for flight control of the aircraft and the mode controls are instructions given by the pilot to the system.

II.2.1.2 Computer Unit

The computer unit is the major element of a channel. It is a microprogrammed general purpose 16 bit computer. The instruction set provides for real-time control applications. It has built-in test and self-monitoring features for detecting arithmetic errors, memory and bus errors (by parity checks), and input/output hardware faults (by self-test loops). The major function is the control of the flight of the aircraft. The computer is designed with cross-chanel communication interfaces to implement data links between channels. The use of optical coupling removes the potential dangers associated with electrical coupling.

II.2.1.3 Iteration Timing Reference and Watchdog Monitor

These are devices independent of the computer unit. Under normal operation, the computers operate in frame synchronous mode where certain computations are performed within a fixed time frame. The end of a time frame is marked by an interrupt from the iteration timing reference. When more than one computer unit are in operation, the synchronization indicator signals generated by each unit in response to the interrupt are examined by other units in other channels. When all syncronization

indicators are set all channels will synchronize. Thus begins a new time frame.

The watchdog monitor checks the synchronization signal of the computer unit to determine if it is set and reset within acceptable upper and lower time limits. If not, the monitor will cause the associated servos to be disengaged and the computer unit marked as faulty.

The iterative timing reference also serves the purposes of interrupting the computer unit at regular intervals after power-on or any transient fault condition to attempt recovery and of synchronizing with the other units.

II.2.1.4 Servo Electronics

The servo electronics interface the computer with the servo actuator. In each actuator, the electronics are duplicated and two inputs are required to operate them. This provides the means for electric current comparison (between the servo electonics outputs) for the purpose of self-monitoring.

II.2.1.5 Servo Monitor and Switch

The servo monitor controls the engage/shut off switch between the servo electronics and the actuator. When a failure is detected by comparison of the output current of the dual servo electronics, or when a signal is received from the watchdog monitor or the computer itself, the servo monitor commands the servo switch to shut off the servo.

II.2.1.6 Servo Actuator

The servo actuators are the mechanical/hydraulic output devices of the channels. They produce hydraulic pressures that are used in the force voting devices that position the aircraft flaps.

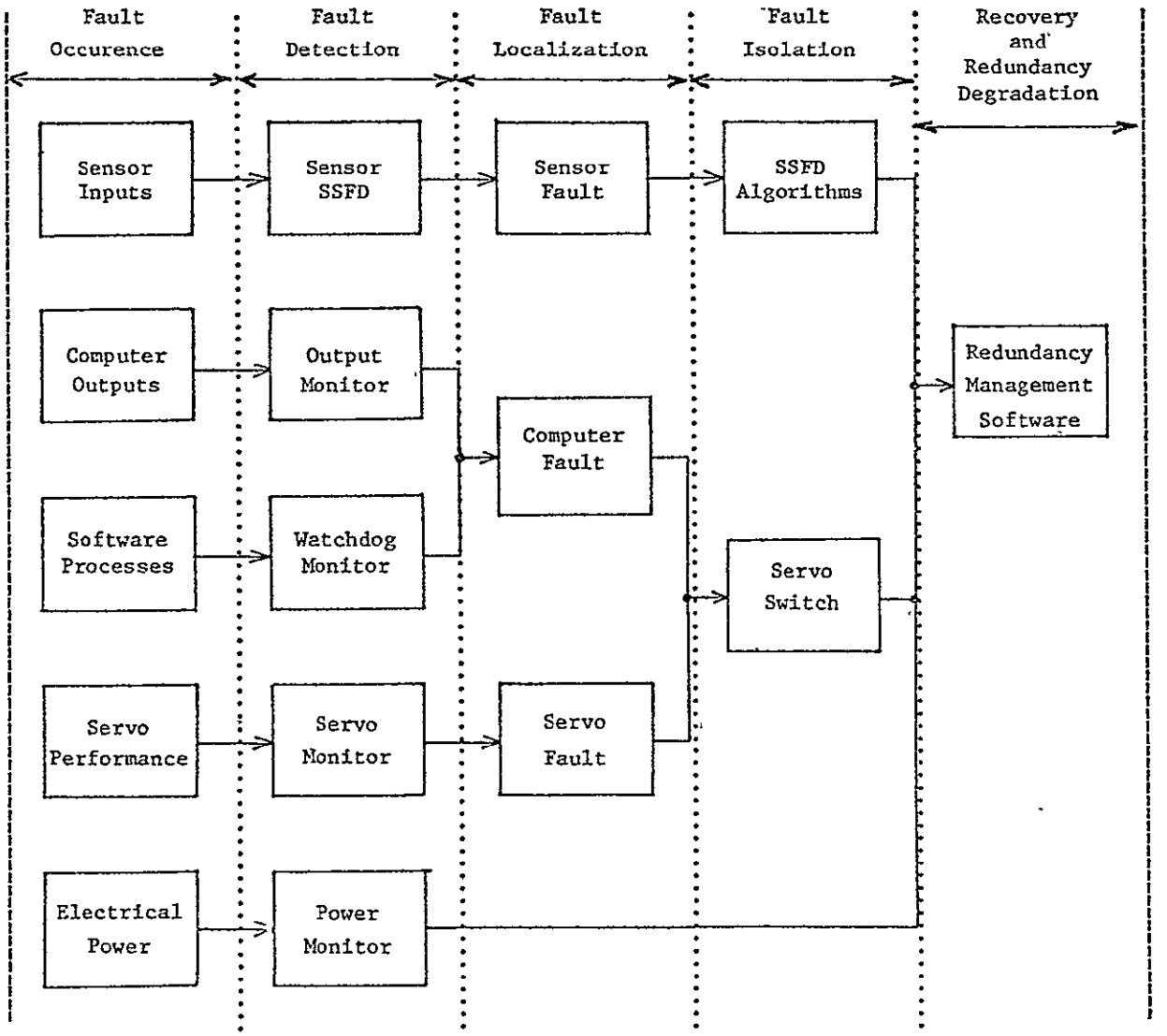


Fig. 3. ARCS reconfiguration process.

II.3 Detection, Diagnosis and Recovery Description

The fault-tolerance objective in the design of ARCS is to provide a system coverage of unity for any first-failure condition in a triplex configuration. Further, a second-failure module coverage of 0.95 or better and a simplex failure-detection probability of 0.90 are design goals for the computer and interfaces modules.

Reconfiguration is the process of attempting to tolerate a fault. It consists of the sub-process of fault detection, fault localization, fault isolation, recovery and redundancy degradation. Fig. 3 shows which sub-processes are initiated in response to particular faults, and Fig. 4 gives a description of the ARCS channels that is based upon the detection, diagnosis and reconfiguration processes.

II.3.1 Reconfiguration for sensor failures

Faults that occur in the sensors are monitored by the sensor Signal Selection and Fault Detection (SSFD) algorithm. Sensors are afflicted with a number of error characteristics: bias error, scale factor tolerance, dynamic response tolrance and noise. In ARCS, compensation of these error characteristics is part of the SSFD process.

For continuous (non-discrete) signals, comparison between the current compensated input data and the average of the input values from the previous iteration is used to monitor for dynamic faults (rapidly deviating raw signal inputs). Calculations of the bias error for the

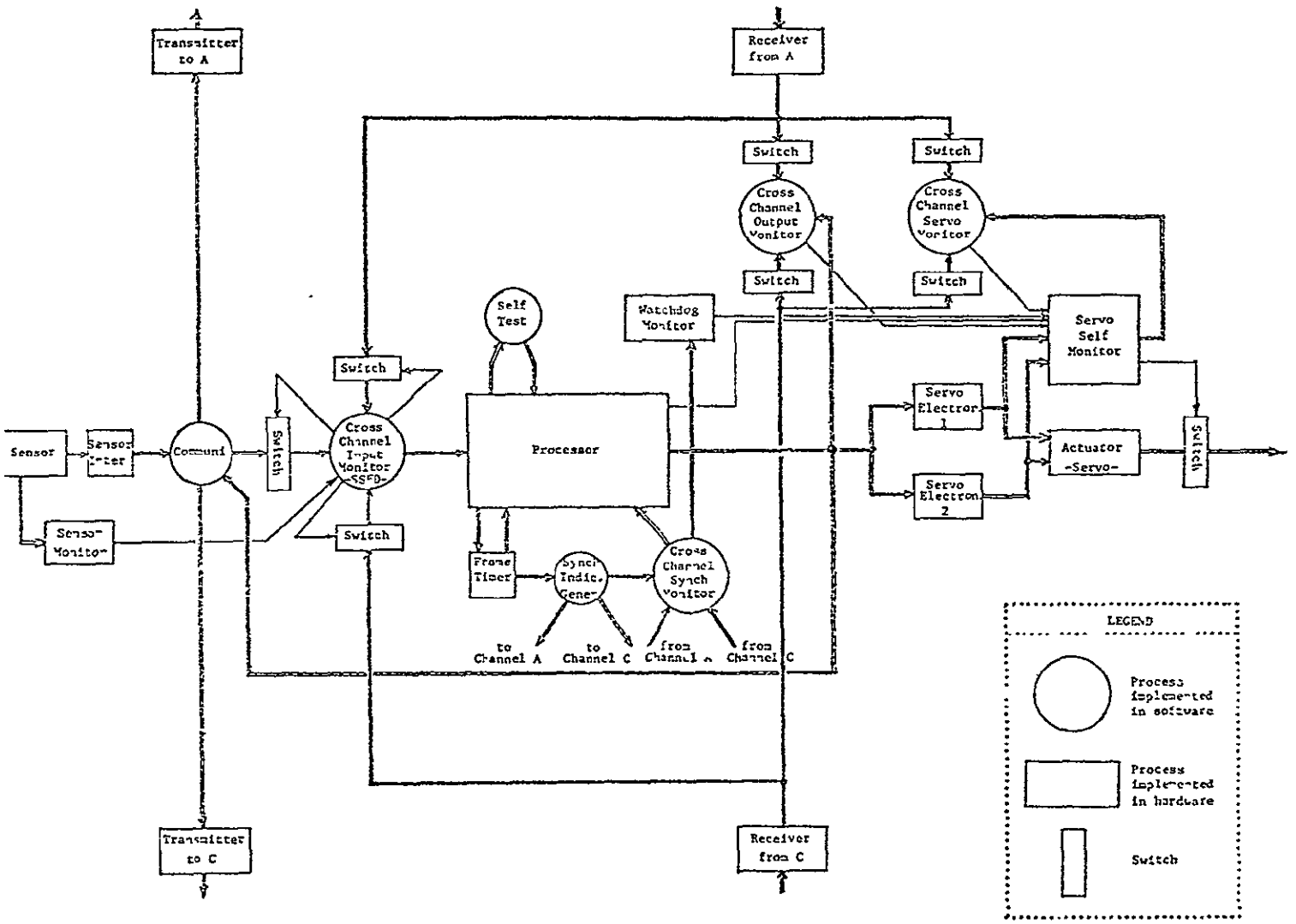


Fig. 4. Description of the channels in ARCS (Channel B reprented).

next iteration provide a means of monitoring static faults. When the bias error reaches a predetermined level, fault is declared. Redundant sensor data (from different channels) are then averaged, and further fault detection is done by comparing these data. For discrete signals, time skews are resolved in the SSFD algorithm. Consistent disagreement of input from one channel with others indicates a fault.

The SSFD algorithm thus consolidates redundant sensor data and provides identical data for all computer units. Further computations operate on identical data, and any discrepency between channels in later stages indicates a fault condition.

Fault isolation is effected by raising do-not-use flags against faulted sensors on a per-channel basis. If the fault is transient, the input will become acceptable after a time delay and the flag will be removed. If it persists, a permanent fault flag is raised, disconnecting this input from the computations. These are the recovery and redundancy degradation processes for sensor faults, which are part of the SSFD algorithm within the Redundancy Management process.

II.3.2 Reconfiguration for computer failures

II.3.2.1 Output Monitor

The output monitor is a software process that compares the computed outputs of a processor with the computed, cross-channel transferred outputs of the other processors. An output monitor flag is set for the affected output if the comparison does not agree within the monitoring

threshold. In such a case, each channel determines if itself can be faulty. A single disgreement in a triplex or quadruplex mode of operation can be determined immediately. Otherwise, each channel checks the fault status table for related faults. If such a fault is registered, the associated servo is disengaged and recovery is attempted using data from a fault-free channel. If the fault persists, the predetermined upper limit on the number of output monitor trips will be exceeded and the fault will be declared permanent.

If no related fault is registered in the fault status table of a channel, the routine to check for a persistent fault in other channels is still performed. Thus, permanent failures of the total system are determined independently by each channel.

II.3.2.2 Watchdog Monitor

The Watchdog Monitor is an independent (in term of hardware, from the processor) fail-safe monitor of the real-time operation of each computer. A computer fault condition will be indicated if the synchronization indicator from the computer arrives outside a specified time interval. If, in a following time frame, the synchronization indicator falls back in the specified time interval, the fault indication will be cleared. When the watchdog monitor trips, it sends a command to the servo monitors to disengage the servos.

Timer interrupts occur when the time counts that define frames reach zero. This initiates the synchronization process for a new frame. The

local synchronization indicator is set and the local computer checks other computers to determine if their synchronization indicators are set. When all indicators are set, all channels clear their local synchronization indicator and they are thus synchronized. Cross-channel communication of the synchronization indicators is made through dedicated hardware links.

II.3.2.3 Additional Testing of the Computer

In addition to these first-level monitors, the computers are provided with hardware parity error detection for the Random Access Memory (RAM) and arithmetic error detection for the processor. These interrupt the processor when any error condition is detected. Furthermore, software implemented self-test functions operate in the background mode, that is, the self-test routines are executed whenever the processor is done with all the computations allocated to the current time frame and there is time remaining before the commencement of the next frame. The self-tests include checking of all the instruction of the processor, the integrity of the memory and the input/output ports.

Any fault detected is entered into the system fault status table, which is used by the redundancy management software for reconfiguration and maintenance routines to update failure records.

II.3.3 Reconfiguration of servo failures

II.3.3.1 Servo Self-Monitor

The behavior of the servo self-monitor was briefly described in a previous section. It compares the two output currents produced by the dual servo electronics and performs the difference. If this exceeds a predetermined threshold, a command is sent to the servo switch to disengage the servo. The servo self-monitor also looks at the actuator to determine whether it is engaged or shut off and signals the processor. The servo self-monitor receives signals from the watchdog monitor and the computer and disengages the servo accordingly.

II.3.3.2 Cross-Channel Servo Monitor

The cross-channel servo monitor is a software process that compares the the sum of the coil currents for corresponding servos in the three channels. Because the servo are close-loop control systems (they are fed back the difference between the actual flap position and the desired position), the cross-channel servo monitors can detect inconsistencies of the force applied by actuator. If a disagreement is noted and the output monitor is not tripped, a servo fault is indicated and the affected servo is disengaged.

II.3.4 Reconfiguration for Power Failures

The power monitor's primary purpose is to indicate a power-on condition. Once a power-on condition is indicated, the channel initiates a recovery

process. This is the same recovery process as is generated after a transient fault has disappeared.

During a initial startup (power-on interrupt) the local channel sets its synchronization indicator. It then waits for 1.1 frame times and tests if other channels are operating. If there are no other activities, the local channel configures itself to operate in the simplex mode. If another channel is operating, the local channel will synchronize with it. As all state variable data required for recovery are transmitted across the channels during every frame by the operating channels, the new channel may now copy this data and start processing.

II.4 Groung Test Mode of Operation

Extensive tests of the overall ARCS are performed in the ground test mode. The computer units are tested by a self-test of all intructions, all registers and all addressing modes. The ROM and RAM are checked. The hardware monitor are tested to insure that they are capable of detecting and enunciating existence of fault conditions. The analog and digital input/output ports are tested by wrap-around testing. Sensors are tested utilizing built-in fault-detection mechanisms, self-tests and operator simulated tests. Servo testing requires synchronization of the redundant channels and includes tests for synchronization, engagement control testing, dynamic response tests and force override tests.

III. SYSTEM STATES AND THEIR CRITICALITY

III.1 Introduction

This section describes a method that makes it possible to enumerate all the dangerous states in which the system can be. A very general description of the method will be given and ARCS will be used as an example. The first part introduces notations that are needed to be able to represent the static state of the system at any point in time. The second part looks at the various degrees of criticality the system can be in. The last part gives a method to enumerate all the system states with a given degree of criticality.

III.2 Notations

III.2.1 Partition of the channels into units

III.2.1.1 Switches

Each channel is composed of a collection of hardware and software components. Among these components, there are some are switches. When they are commanded, they disable some of the channel components (for example the servo switches). Some other 'components' have similar effects: they allow the channel to ignore data coming from some of its components or from other channels. As a example in ARCS, one can

mention the "do-not-use" flags that control whether or not sensor outputs will be ignored, and also the fault-status table that indicates whether or not a channel trusts the other channels. All these components (whether hardware like the servo switches or software like the "do-not-use" flags or the fault-status table) will be referred to as switches. In ARCS, one can list as switches the trios of "do-not-use" flags,the couples of "disregard-channel" flags in the local fault status table, the servo switches and possibly, depending upon the exact software specification of the system, a computer flag that will order the computer to discontinue to send any further commands to the servo electronics (Fig. 5).

III.2.1.2 Units

These switches induce a partition of the channels into units. A unit is a set of components upon which all the switches have the same action. For example, a sensor and its interface constitute a unit, for the loss of either one implies the loss of both and the "do-not-use" flag will logically disconnect both out of the channel. The partition of a channel into units is not dictated by strict rules and the user has much liberty. One partition that can be given for the channels of ARCS is shown in Fig. 6 . However, one should note that, because each channel has access to other channel data by the cross-channel communication link, one should consider the computer and sensors of both channel A and channel C as units of channel B. A failure in computer A will cause computer B to disregard data coming from computer A or from its sensors. But a failure in one of the sensors of channel A will only disconnect that sensor.

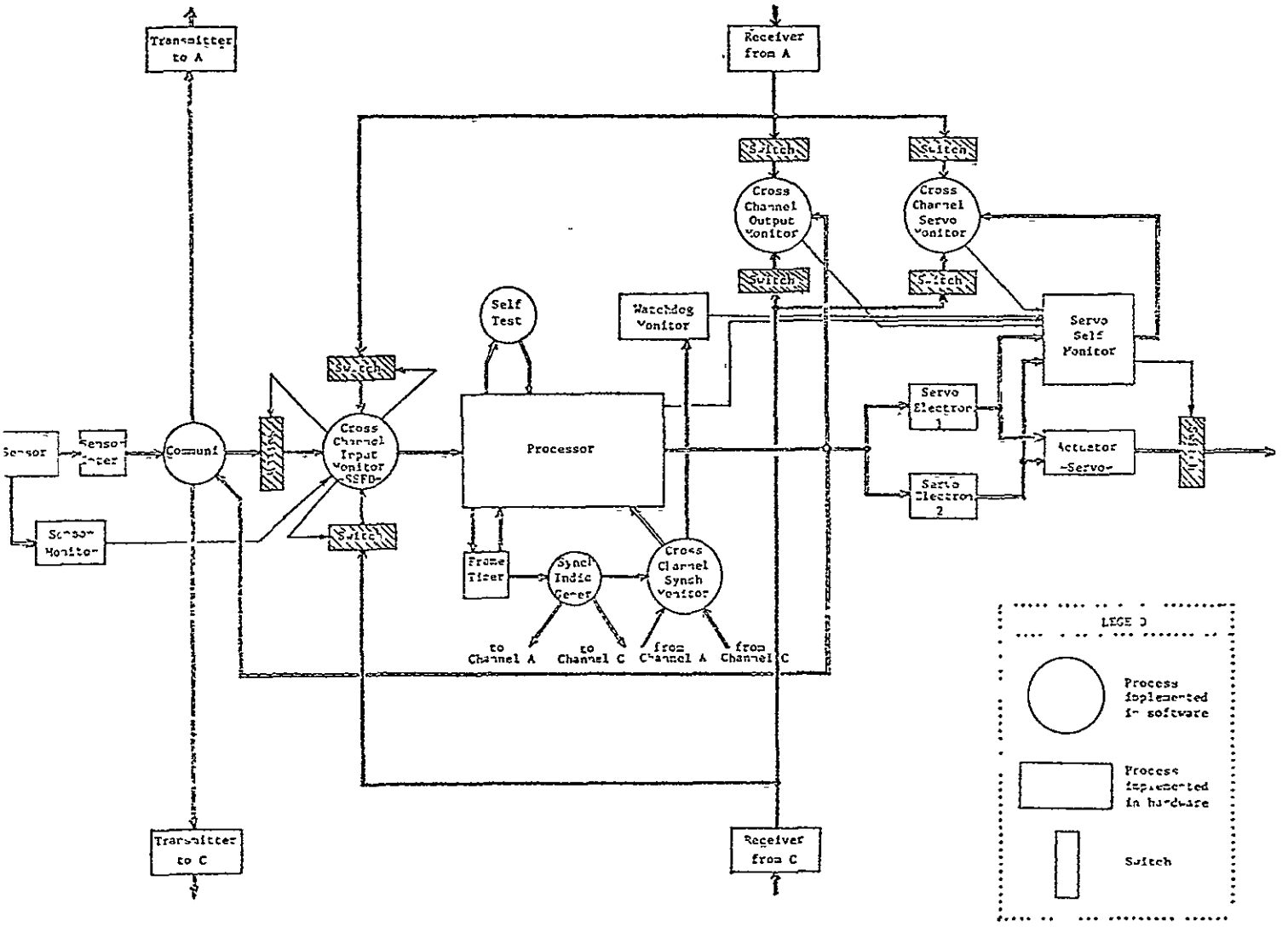


Fig. 5. Switches in ARCS.

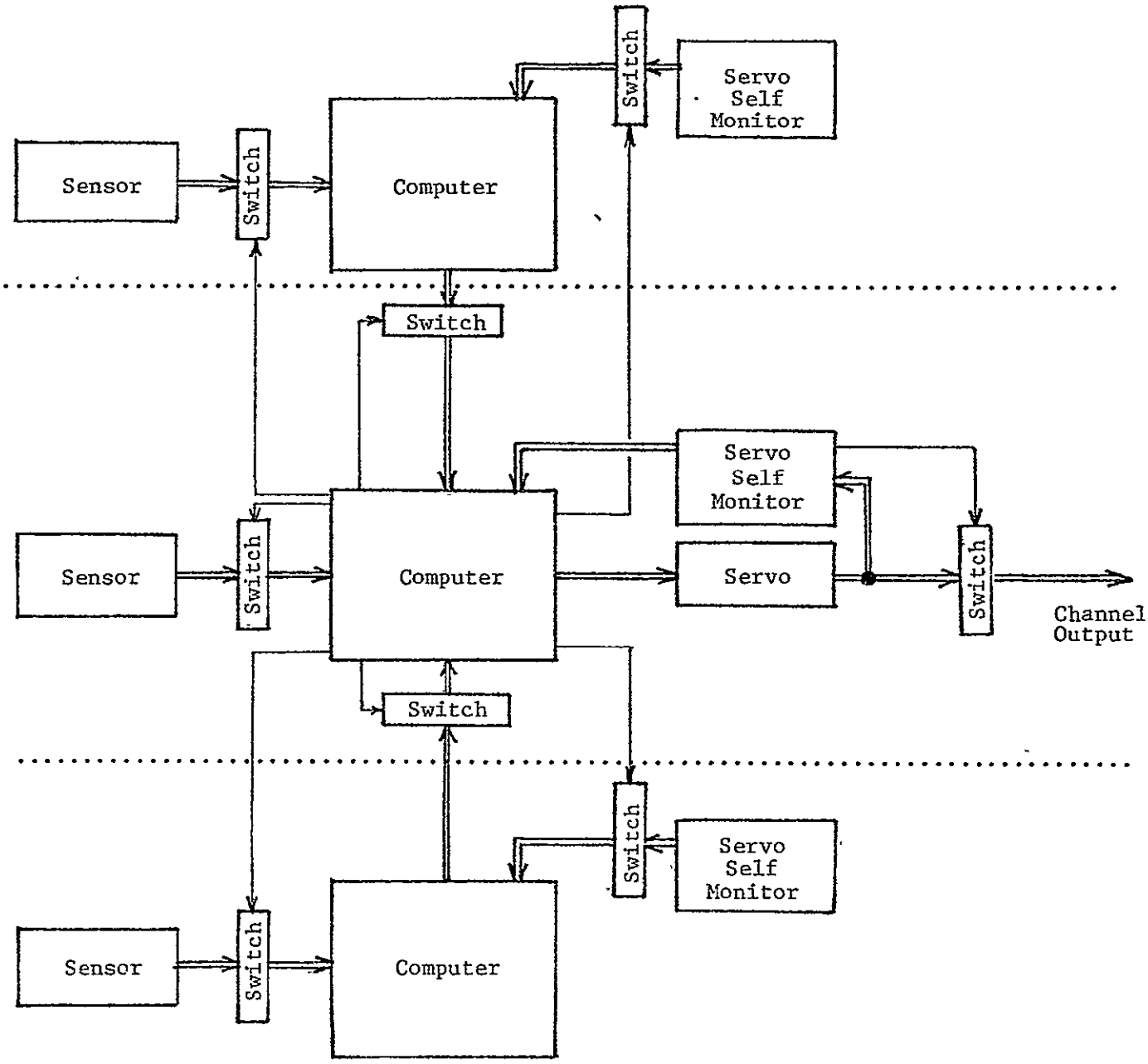


Fig. 6. Partition of the channels in ARCS (Channel B represented).

The units in one channel will be numbered from 1 to n. Switches are considered as units and are numbered from 1 to k. Because of the symmetry between channels, all channels in ARCS have the same partition.

III.2.2 Failure Configuration Vectors

When a failure occurs in a unit, the unit behavior becomes potentially incorrect. At any point in time, we can represent the state of an unit (whether or not that unit has suffered a permanent failure) by a binary variable. Let the three vectors ${}_A\vec{F}$, ${}_B\vec{F}$ and ${}_C\vec{F}$ represent the state of the units in channels A, B and C respectively:

$${}_A\vec{F} = \langle {}_Af_1, {}_Af_2, \cdots, {}_Af_n \rangle \quad \text{with } {}_Af_i = 0 \text{ if unit i of channel A is fault-free,}$$

$$\text{with } {}_Af_i = 1 \text{ if unit i of channel A is faulty,}$$

$${}_B\vec{F} = \langle {}_Bf_1, {}_Bf_2, \cdots, {}_Bf_n \rangle$$

and

$${}_C\vec{F} = \langle {}_Cf_1, {}_Cf_2, \cdots, {}_Cf_n \rangle.$$

The vectors ${}_A\vec{F}$, ${}_B\vec{F}$ and ${}_C\vec{F}$ will be called the failure configuration vector for channel A, B and C respectively. If the state of a unit is irrelevant, an x is put for its state variable. The complete system failure configuration will be represented by $\vec{F}$:

$$\vec{F} = \langle {}_A\vec{F}, {}_B\vec{F}, {}_C\vec{F} \rangle.$$

III.2.3 Channel Configuration Vectors

Following failure occurrences, each channel takes appropriate action by commanding some switches to turn off. The units that are disabled (or disregarded) by the switch actions are said to be reconfigured out of the channel. The configuration vectors, ${}_A\vec{R}$, ${}_B\vec{R}$ and ${}_C\vec{R}$ indicate whether or not units are configured out of channel A, B and C respectively:

$${}_A\vec{R} = \langle {}_Ar_1, {}_Ar_2, \cdots, {}_Ar_n \rangle \quad \text{with} \quad {}_Ar_1 = 0 \text{ if unit i of channel A is not configured out of channel A,}$$

$$\text{with} \quad {}_Ar_i = 1 \text{ if unit i of channel A is configured out of channel A,}$$

$${}_B\vec{R} = \langle {}_Br_1, {}_Br_2, \cdots, {}_Br_n \rangle$$

and

$${}_C\vec{R} = \langle {}_Cr_1, {}_Cr_2, \cdots, {}_Cr_n \rangle.$$

Similarly to the failure configuration vectors, a entry of x means that the state (configured in or out) is either one. The complete system configuration will be represented by $\vec{R}$:

$$\vec{R} = \langle {}_A\vec{R}, {}_B\vec{R}, {}_C\vec{R} \rangle.$$

III.2.4 Channel and System States

Both the failure configuration vectors and the channel configuration vectors give a static picture of the system at any point in time. The failure configuration vectors indicate which units are faulty while the

configuration vectors point to the units that are reconfigured out of the channels at that time. One should note that it is conceptually possible for one unit, for example a computer, to be reconfigured out of one channel but still kept active in another channel. For example, when a system operates in a triplex mode, a computer failure may not be detected by the computer itself - or the computer will ignore the failure indications in provenance of the fault-detection mechanisms - while the other two channels will note the disagreement.

The static state of a channel can be defined as the couple formed of the failure configuration and the channel configuration vectors for that particular channel:

$${}_A\Omega = \langle {}_A\vec{F},\ {}_A\vec{R} \rangle = \text{Static State of channel A}$$
$${}_B\Omega = \langle {}_B\vec{F},\ {}_B\vec{R} \rangle = \text{Static State of channel B}$$
$${}_C\Omega = \langle {}_C\vec{F},\ {}_C\vec{R} \rangle = \text{Static State of Channel C.}$$

The system state is the collection of all three channel states:

$$\Omega = \langle {}_A\Omega,\ {}_B\Omega,\ {}_C\Omega \rangle = \text{Static State of the system.}$$

III.2.5 Channel Validity Equations

Because the correct functioning of every single unit in a channel is not required for the channel to produce the correct pressures on the voters, there is a boolean expression, V, that relates the correctness of the channel commands to output mechanisms (actuators, displays,...) to the state of the channel units. This expression, V, involves as variables

both the failure state of the units , excluding the output units, and the configuration state (whether or not the unit is still configured in the channel). For unit i of any channel (channel A for example), ${}_Af_i$ is 1 when the unit is faulty and ${}_Ar_i$ is 1 when the unit is configured out of the channel. So, in the actual system configuration, unit $_i$ correctly performs its function if and only if both ${}_Af_i$ and ${}_Ar_i$ are 0. So, the validity expression, V, can be expressed as a function of the AND of vectors ${}_A\vec{F}$ and ${}_A\vec{R}$:

$V({}_A\vec{F}.{}_A\vec{R})$ = 1 implies that channel A in state $\langle {}_A\vec{F}, {}_A\vec{R}\rangle$ produces correct commands to all the output devices,

= 0 implies that channel A in state $\langle {}_A\vec{F}, {}_A\vec{R}\rangle$ does not produce correct commands to the output devices.

The validity expression does not indicate whether the actual outputs are correct. A faulty output mechanism, like an actuator, can yield incorrect pressures even if the channel is correct. The validity expression is important because the computers control several output devices and a failure in one output device does not influence the correctness of other output devices.

The expression V can be easily obtained from the system description. It directly expresses the redundancy that a channel can take advantage of to carry its computations.

III.3 State Criticality and Associated Conditions

Failures can be globally considered as transitions between static system states. So, the criticality of failures is directly related to the notion of system state criticality.

III.3.1 Voter Behavior and its Influence on State Criticality

Corresponding actuators in the three channels produce pressures which are force-voted by mechanical voters which position the aircraft flaps. When one of the three pressures is incorrect, the mechanical voter is presented with contradictory forces. The resulting flap position depends upon the behavior of the voter in such a situation. For the completeness of any analysis related to system crash, one must examine carefully how the mechanical voter behaves in such conditions.

III.3.1.1 Correct Flap Positioning with one Faulty Actuator

The mechanical voters are devices which translate a set of three input pressures, p_1, p_2, p_3, into a flap position. The actual motion of the flap between its initial position and the desired position indicated by the pressures is subject to a differential equation. The final flap position (if there is a steady state solution) depends upon the three pressures p_1, p_2, p_3. A detailed study of the mechanical voter behavior is required to check if the actual final position of the flap is within tolerance when one of the three pressures, let assume p_1, differs from the others. Such a analysis of the voter will likely give a range of

values for p_1, as a function of p_2 and p_3, where the final flap position is within tolerance. Only if this range covers the complete range of values for p_1, and this for all possible values of p_2 and p_3, will the voter always adequately compensate for a faulty pressure (compensation in the sense that the flap position is within tolerance). However, the flap motion is subject to a differential equation, so it is still possible that the flap will reach its final position with large oscillations that may damage the aircraft. So, in addition to an analysis of the static behavior of the voter, one also must analyse the dynamic behavior of the flap motion as a function of the three input pressures , p_1, p_2, p_3.

The complete analysis of the flap motion (including the static behavior of the voter when one input pressure disagrees with the other two and the corresponding motion of the flap) will yield operating ranges where the system operates correctly with two fault-free pressures and a faulty one. These ranges will be called the adequate compensation ranges.

Only if the adequate compensation ranges cover all the possible operating ranges with one faulty pressure, will the operating mode - two fault-free pressures, one faulty one - corresponds to a fault-free mode of operation for the system. If it is possible that a faulty actuator which is not disengaged causes potential dangers to the aircraft, for example flap positionings that are slightly off or aircraft vibrations and oscillations, the operating mode with one faulty pressure at a voter input will be called a <u>dangerous</u> mode of operation.

III.3.1.2. Modes of Operation Dictated by the Output Mechanisms

The description of the mechanical behavior of the actuators and voters is not very detailed in the ARCS report. However, it is conceivable that several other modes of operation with various degrees of criticality can be defined for the set voter-flap. In general, one can say that the output devices will dictate a partition of the operating modes into several classes with different degrees of criticality. For ARCS, a simple analysis gives the following modes of operation:

- All three pressures at the voter inputs agree: the voter is not submitted to undue stresses,

- Two input pressures agree, the third one is in disagreement: the voter may or may not compensate adequately depending upon the value of the incorrect pressure,

- Two input pressures agree, the third actuator is shut-off; the flap position will be correctly controlled by the two agreeing pressures,

- Only one pressure is fed to the voter: the flap position is controlled by that pressure,

- Two pressures are applied to the voter but they disagree: the flap position will depend upon the relation between the disagreeing pressures,

- No pressure is applied: the flap does not move.

III.3.2 Modes of Operations and Their Degree of Criticality

The modes of operation dictated by the behavior of the output mechanisms need to be extended to take into account the correctness of the pressures each channel produces. For example, if the voter receives three pressures that agree, it does not necessarily mean that they are the correct pressures. For a system such as ARCS, one can enumerate the various modes of operation:

- Each mechanical voter (one per flap) receives three identical pressures that are the correct ones: all the flaps will be correctly moved, the system is fault-free.

- Each mechanical voter receives at least two correct, identical pressures but none receives a third pressure that disagrees with the other two. At least two actuators for each voter produce the correct pressure and the third one either also produces the correct pressure or is disengaged, thus the system is fault-free (all the flaps moved correctly) and there is no potential danger (no flap oscillations nor slightly-off positioning),

- One of the mechanical voters receives only one pressure, the correct one, while all the other voters receive either two or three correct pressures but no incorrect pressures. The system is still fault-free; all the flaps are correctly positioned but the tolerance to further failures is low.

- At least one voter receives two correct and one incorrect pressure and all the other voters receive only correct pressures, either one

two or three. If the discrepency between the pressures is not too large, (if the voter adequately compensates for it), then the system is still operating correctly. However, this mode of operation is quite dangerous since an increase in the discrepency may produce oscillations or slighly incorrect positioning. Such a mode of operation will be called dangerous and warrants special study.

- The system has failed (not able any longer to position correctly the flaps) but was able to signal the fact to the pilot before issuing any wrong commands to the servos. This is a safe failure; the system has suffered a fail-safe crash.

- The system has failed but was not able to signal the fact to the pilot. Thus, the system is still in control of the aircraft but sends wrong positioning commands to the flaps. This is the worst mode of operation.

These modes of operation are listed in increasing order of criticality. In general, for any system, one can list all the operating modes and order them by increasing criticality. We will assume, in the following, that the system has d modes of operation, ordered from 1 to d by increasing criticality. One should note that the notion of modes of operation carries far more meaning that the simpler notion of fault-free/failed dichotomy. A failure pattern (successive occurrences of different failures) may take the system from a fault-free state to a failed state, but each individual failure in the pattern will correspond to a transition between different operating modes, in general from a mode with a given criticality to another mode with a higher criticality.

III.3.3 Conditions Associated with Operating Modes

For the system to be in a given mode of operation, it must satisfy some conditions. These conditions express the requirements on the system state in terms of failure and channel configurations. For example, for the dangerous mode of operation one has the following conditions:

For at least one actuator trio, trio U, two channels produce the correct pressure while the third one gives an incorrect pressure (actuator still engaged): This can be expressed in terms of the state variables as follows:

$$\left.\begin{aligned} &\left(\left[V({}_A\vec{F}\cdot{}_A\vec{R})\cdot {}_A f_u\right] + {}_A r_u\right) = 0 \\ &\left(V({}_B\vec{F}\cdot{}_B\vec{R})\cdot {}_B f_u \cdot {}_B r_u\right) = 1 \\ &\left(V({}_C\vec{F}\cdot{}_C\vec{R})\cdot {}_C f_u \cdot {}_C r_u\right) = 1 \end{aligned}\right\}$$ Servo u of channel A produces incorrect pressures.

$$\text{or}\left.\begin{aligned} &\left(V({}_A\vec{F}\cdot{}_A\vec{R})\cdot {}_A f_u \cdot {}_A r_u\right) = 1 \\ &\left(\left[V({}_B\vec{F}\cdot{}_B\vec{R})\cdot {}_B f_u\right] + {}_B r_u\right) = 0 \\ &\left(V({}_C\vec{F}\cdot{}_C\vec{R})\cdot {}_C f_u \cdot {}_C r_u\right) = 1 \end{aligned}\right\}$$ Servo u of channel B produces incorrect pressures.

$$\text{or}\left.\begin{aligned} &\left(V({}_A\vec{F}\cdot{}_A\vec{R})\cdot {}_A f_u \cdot {}_A r_u\right) = 1 \\ &\left(V({}_B\vec{F}\cdot{}_B\vec{R})\cdot {}_B f_u \cdot {}_B r_u\right) = 1 \\ &\left(\left[V({}_C\vec{F}\cdot{}_C\vec{R})\cdot {}_C f_u\right] + {}_C r_u\right) = 0 \end{aligned}\right\}$$ Servo u of channel C produces incorrect pressures.

For all other actuator trios, there are either one, two or three correct pressures and no faulty actuator still engaged:

$$V({}_A\vec{F}\cdot{}_A\vec{R}) \cdot {}_Af_v \cdot {}_Ar_v = 1$$
$$V({}_B\vec{F}\cdot{}_B\vec{R}) \cdot {}_Bf_v \cdot {}_Br_v = 1$$
$$V({}_C\vec{F}\cdot{}_C\vec{R}) \cdot {}_Cf_v \cdot {}_Cr_v = 1$$

or $${}_Ar_v = 0$$
$$V({}_B\vec{F}\cdot{}_B\vec{R}) \cdot {}_Bf_v \cdot {}_Bf_v = 1$$
$$V({}_C\vec{F}\cdot{}_C\vec{R}) \cdot {}_Cf_v \cdot {}_Cr_v = 1$$

or $$V({}_A\vec{F}\cdot{}_A\vec{R}) \cdot {}_Af_v \cdot {}_Ar_v = 1$$
$${}_Br_v = 0$$
$$V({}_C\vec{F}\cdot{}_C\vec{R}) \cdot {}_Cf_v \cdot {}_Cr_v = 1$$

or $$V({}_A\vec{F}\cdot{}_A\vec{R}) \cdot {}_Af_v \cdot {}_Ar_v = 1$$
$$V({}_B\vec{F}\cdot{}_B\vec{R}) \cdot {}_Bf_v \cdot {}_Br_v = 1$$
$${}_Cr_v = 0$$

or $$V({}_A\vec{F}\cdot{}_A\vec{R}) \cdot {}_Af_v \cdot {}_Ar_v = 1$$
$${}_Br_v = 0$$
$${}_Cr_v = 0$$

or $${}_Ar_v = 0$$
$$V({}_B\vec{F}\cdot{}_B\vec{R}) \cdot {}_Bf_v \cdot {}_Br_v = 1$$
$${}_Cr_v = 0$$

or $${}_Ar_v = 0$$
$${}_Br_v = 0$$
$$V({}_C\vec{F}\cdot{}_C\vec{R}) \cdot {}_Cf_v \cdot {}_Cr_v = 1$$

For every operating mode, one can list all the associated conditions in terms of the variables of vectors $\vec{F}$ and $\vec{R}$. The set of conditions for the operating mode with criticality degree d will be referred to as $\{C_d\}$.

It is possible to simplify these conditions by boolean manipulations. It is also possible to express these conditions in a table form that may provide a simpler way to represent the state of the system. r

III.3.4 System States that Correspond to a Given Operating Mode.

As it was shown, it is possible to find a set of conditions that must be satisfied for the system to be in a given operating mode. These conditions are also sufficient.

These boolean conditions provide a mathematical means to find all the states the system can be in for a given operating mode. The system states are given by the combination of the F and R vectors that satisfy the conditions $\{C_d\}$.

All the system states that satisfy the conditions for a given operating mode have the criticality of this operating mode. However, it is possible that some of these states can not actually occur in the systems. A fault pattern that leads the system from an initial fault-free state to a final state characterized by the failure configuration vectors $\vec{F}$ will allow only certain actions to be taken. For example, it is not possible that the failure of one actuator in channel A causes the computer of channel B to be declared faulty (as long as everything else is fault-free). So, once all the possible system states that correspond to a given operating mode are enumerated, it is necessary to study whether or not they can actually occur in the system. This will involve looking at the detection, diagnosis and reconfiguration processes and is the subject of section IV.

III.4 A Method to Enumerate All the System States with Given Criticality

The process for finding all the system states that can correspond to a given mode of operation, hence to a given criticality degree, can be automated and implemented as a computer program. It follows closely the steps described in the previous sections (Fig. 7). The advantages of such an computer implementation are first and foremost that it will yield a complete enumeration of all the potentially dangerous states, and secondly, that it allows fast system analysis, hence, the possibility to study the effects of various system modifications.

III.4.1 First Step

The first step of the method is to look at the system under study and to find a partition into channels. One also needs to find all the output voters. Careful study of these voters (especially when they are mechanical devices as in ARCS) will tell if, when and with which confidence voters can compensate for faulty inputs. This will give a first partition of the system operating modes (for example, all inputs in agreement; the majority of the inputs in agreement, all other inputs disengaged; the majority of inputs in agreement, one input disagrees, and all remaining inputs are disengaged, ...).

III.4.2 Second Step

The second step consists in looking at each channel and in finding all the ways a channel can disable/disengage/disregard a piece of logic. This requires a study of the system hardware and of the

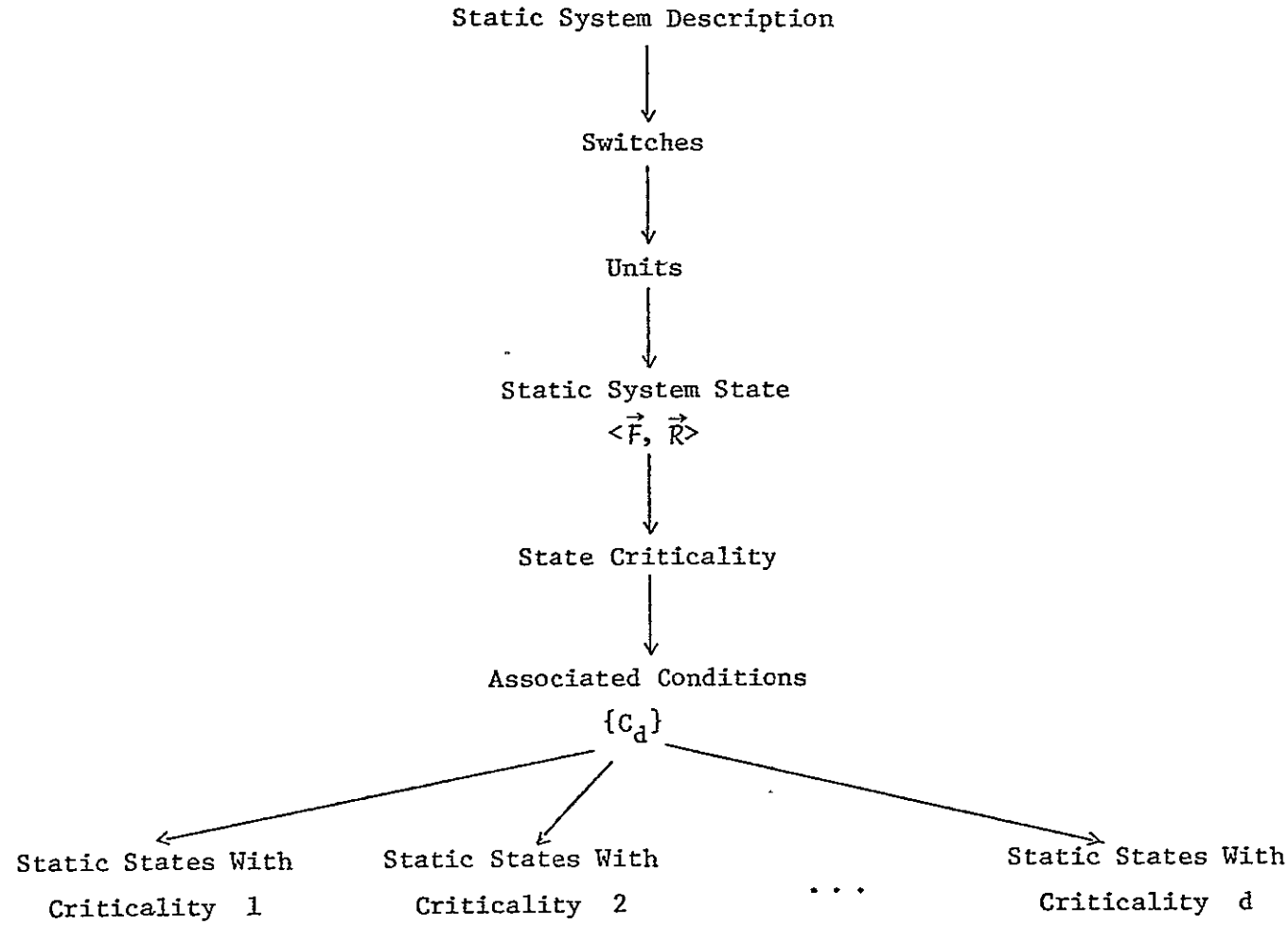


Fig. 7. Process of enumerating the critical states.

hardware/software process of reconfiguration (to find the software implemented switches). This will give the set of switches and also a partition of the channel into units (partition with respect to the control of the switces and their action on the system). This step should also provide the controllability and the controlling matrices (which units control the switches and which units are disabled by the switches). This information will be useful while looking at the detection/diagnosis/reconfiguration processes to decide which of the critical states can actually occur.

Because of the use of redundancy (both internal redundancy inside the channels and external redundancy such as the use of sensor data from other channels) one needs to find the conditions that a channel must satisfy in order that the commands it sends to its output devices be correct. These conditions can be expressed as a boolean expression, expression V, that involves the states and configuration (fault-free, faulty, enabled, disabled) of the system units.

III.4.3 Third Step

The third step consists in listing all the operating modes as functions of the partition induced by the behavior of the output voters and the validity of the channel outputs. For each mode of operation, it is necessary to find the corresponding boolean conditions in terms of the failure vectors, $\vec{F}$, the configuration vectors, $\vec{R}$, and the channel validity expression.

III.4.4 Fourth Step

This last step consists in finding all the solutions (all the system states) that satisfy all the conditions for the various operating modes. A complete enumeration of all the solutions may require much computation since it will require looking at every state and checking whether or not it meets the conditions. Some simplification can be obtained if all the channels are similar (taking advantage of the symmetries in the solutions). Also, as we are principally concerned with the analysis of the system tolerance to the first few failures (i.e. the first, second and third failures), one can significantly reduce the amount of computation if we limit ourselves to them. This step will yield all the system states with a given criticality (the criticality of the operating mode) and with the additional restriction that only a few units have suffered failures (first few failures).

IV. FAULT-PATTERNS AND THEIR CRITICALITY

IV. Introduction

The previous section presented a method to list all the system states with a given criticality. In this section, we will make the correspondance between the critical system states and the fault-patterns (successive occurrences of failures) that take the system from the initial fault-free state to these critical states. This will provide a method to list all the failure events that can leave the system in a critical, dangerous or failed state without analyzing every single failure.

The method to go from the static states to the fault-patterns is illustrated in Fig. 8. First, for each critical state, one has to find all the combinations of failures that can drive the system in such a state (obtained from the failure configuration vectors). The order of occurrences is important since a failure can affect the functioning of a fault-detection mechanism for a later failure. So, this first step will give all the possibly critical fault-patterns that correspond to a given critical state. Secondly, we will look at the detection process and find all the possible error indications that can produced following a failure. Then, one needs to consider the diagnosis process and find which diagnosis will be produced for a given set of error indications. The next step is to find which commands are sent to the switches and what is the final system state. Thus, we will be able to decide whether, and under which conditions, a failure can leave the system in a

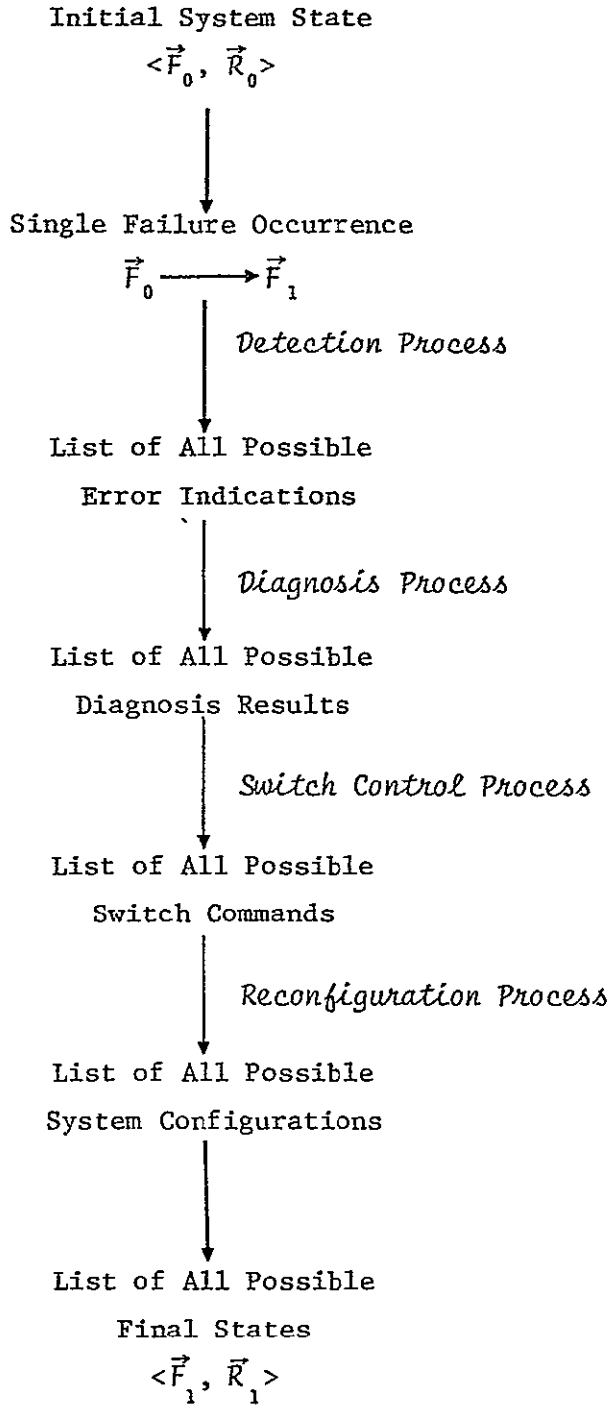


Fig.8. Failures and their effects.

given state. This process will then be repeated for each of the individual failures that make up the possibly critical fault-patterns. Then, we can decide whether a possibly critical fault-pattern can indeed drive the system into a critical state.

This method will provide the list of all the fault-patterns that leave the system in a critical state. Each individual failure in the pattern will be characterized by its location (at the unit level), the list of error indications that is produced, the diagnosis that is generated and the actual commands sent to the system switches.

The enumeration of all the critical fault-patterns is very helpful for evaluating the system tolerance to failures. It will indicate which unit(s) is (are) the most critical and, hence, where additional testing will be beneficial. It is also a necessary step before studying the detailed hardware description of each unit and enumerating all the critical hardware failures. A complete enumeration of all the single and double (without mentioning triple) failures is a prohibitive task. But the knowledge of the critical fault-patterns and their characteristics will sustantially reduce the number of actual failures that need to be analyzed.

IV.2 Definitions and Notations

IV.2.1 Fault-Patterns

A fault-pattern is a succession of failure occurrence. The order in which failures occur can be quite important. For example, if we assume

that ARCS is operating in a simplex mode (only one channel up), an undetected failure in a sensor followed by a detected failure in the processor will cause an unsafe system crash. The undetected failure in the sensor will cause wrong commands to be issued to the actuators. On the other hand, if the computer failure (which is detected) occurs first, the computer will be able to signal the pilot, hence, this will be a safe failure. One can represents a fault-pattern that takes the system from the initial fault-free state to a final state with failure vectors ${}_A\vec{F}$, ${}_B\vec{F}$ and ${}_C\vec{F}$ as the series of state transitions:

$$\Omega_0 = \langle \vec{F}_0, \vec{R}_0 \rangle \xrightarrow{dF_1} \Omega_1 = \langle \vec{F}_1, \vec{R}_1 \rangle \xrightarrow{dF_2} \cdots \xrightarrow{dF_x} \Omega_x = \langle \vec{F}_x, \vec{R}_x \rangle$$

Each of the individual failure occurrence will be denoted by dF_i where i refers to the rank of the failure in the pattern.

IV.2.2 Detection Mechanisms and Detection Vectors

Each channel of ARCS is provided with fault-detection mechanisms that are either hardware or software. We will call a fault-detection mechanism, also referred to as a detector, every hardware or software component of the channel that can detect the presence of failure. ARCS has the following detectors (Fig. 9):

a sensor self-monitor for every sensor,

a cross channel sensor monitor (also called SSFD) for every sensor,

a computer self-test in every channel,

a watchdog monitor for every computer,

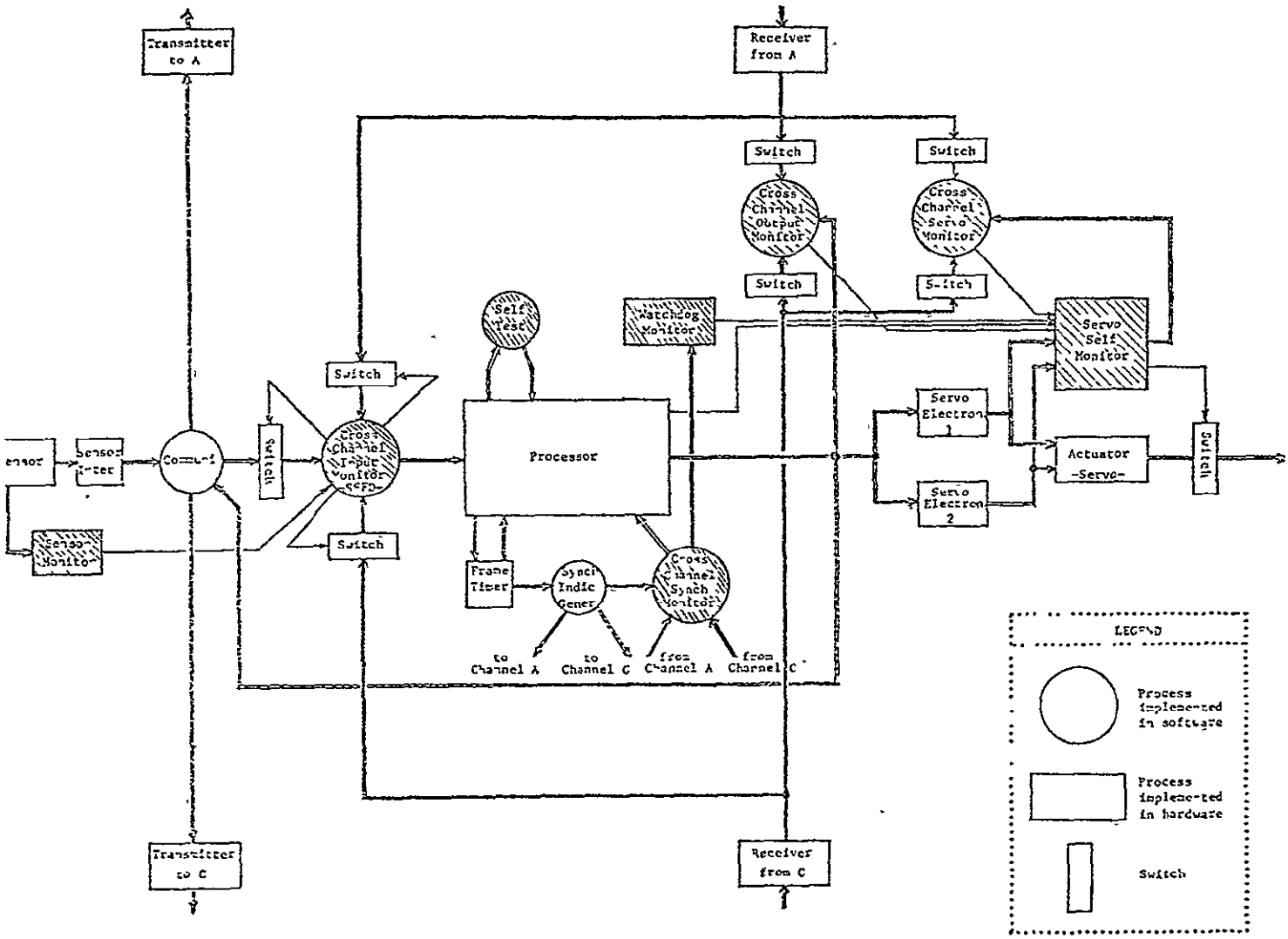


Fig. 9. Fault-detection mechanisms in ARCS.

a cross-channel synchronization monitor in every channel,

a cross-channel output monitor in every channel,

a servo self-monitor for every servo,

and a cross-channel servo monitor for every servo.

In general, it is quite simple to list all the fault-detection mechanisms for any computer systems. In the following, one will assume that each channel has k fault-detection mechanisms.

Following the occurrence of a failure, some of the fault-detection mechanisms may indicate the presence of the failure. Let ${}_At_j$ denote the binary variable which indicates whether or not detector j in channel A has detected the failure:

$$ {}_A\vec{T} = \langle {}_At_1, {}_At_2, \cdots, {}_At_k \rangle $$ with ${}_At_j = 0$ if detector j of channel A does not give an error indication;

with ${}_At_j = 1$ if detector j of channel A gives an error indication,

$$ {}_B\vec{T} = \langle {}_Bt_1, {}_Bt_2, \cdots, {}_Bt_k \rangle $$

and $$ {}_C\vec{T} = \langle {}_Ct_1, {}_Ct_2, \cdots, {}_Ct_k \rangle . $$

The vectors ${}_A\vec{T}$, ${}_B\vec{T}$ and ${}_C\vec{T}$ are called the detection vectors (also referred to as trip vectors). The overall detection vector is denoted by $\vec{T}$:

$$ \vec{T} = \langle {}_A\vec{T}, {}_B\vec{T}, {}_C\vec{T} \rangle . $$

IV.2.3 Diagnosis Mechanisms and Diagnosis Vectors

When a fault-detection mechanism gives an error indication, it is sometimes required for each channel to perform some diagnosis. For example, when the cross-channel output monitor in channel B detects a disagreement, the computer of channel B needs to test itself, to send information to other channels, to check the validity of other channels (based on the information it receives) and to make a decision based on its fault-status table and past failure history. This process is called the diagnosis process realized by the computer. However, the diagnosis process is not run for every error indication. For example, when the watchdog monitor or the servo self-monitor indicates an error, this information is sent directlyt to the servo self-monitor to command the actuator switch to turn off. The computer does not run the diagnosis program.

In general, one will call diagnosis mechanism any component that looks at some of the detectors and makes a decision concerning where the failure is located. In most systems, diagnosis will be performed by the computers (when there is need for diagnosis). However, to be quite general, one will assume that each channel can have more than one diagnosis mechanism (let m represent such diagnosis mechanism). Each diagnosis mechanism looks at some of the indications coming from the detectors and decides where the failure is located inside a subset of the channel. So, for each diagnosis mechanism, there is a corresponding diagnosis vector that indicates the decision made by that particular diagnosis mechanism concerning the location of the failure:

$_A\vec{D}_u = \langle {}_Ad_1, {}_Ad_2, \ldots, {}_Ad_n \rangle$ with ${}_Ad_i = 0$ if the diagnosis mechanism of channel A does not locate the failure in unit i of channel A,

with ${}_Ad_i = 1$ if the diagnosis mechanism locates the failure in unit i of channel A,

with ${}_Ad_i = 2$ if unit i of channel A is not part of the diagnosis range of the detection mechanism.

$_B\vec{D}_u = \langle {}_Bd_1, {}_Bd_2, \ldots, {}_Bd_n \rangle$,

$_C\vec{D}_u = \langle {}_Cd_1, {}_Cd_2, \ldots, {}_Cd_n \rangle$.

These diagnosis vectors carry the information that the system has about itself. For notation simplicity, one will use ${}_A\vec{\mathcal{D}}$ (${}_B\vec{\mathcal{D}}$ or ${}_C\vec{\mathcal{D}}$) to refer to the set of all diagnosis vectors produced in channel A (B or C):

$$_A\vec{\mathcal{D}} = \langle {}_A\vec{D}_1, {}_A\vec{D}_2, \ldots, {}_A\vec{D}_m \rangle$$
$$_B\vec{\mathcal{D}} = \langle {}_B\vec{D}_1, {}_B\vec{D}_2, \ldots, {}_B\vec{D}_m \rangle$$
$$_C\vec{\mathcal{D}} = \langle {}_C\vec{D}_1, {}_C\vec{D}_2, \ldots, {}_C\vec{D}_m \rangle.$$

Also, we will use $\vec{\mathcal{D}}$ to refer to the complete system diagnosis:

$$\vec{\mathcal{D}} = \langle {}_A\vec{\mathcal{D}}, {}_B\vec{\mathcal{D}}, \ldots, {}_C\vec{\mathcal{D}} \rangle.$$

IV.2.4 Switch Control Mechanisms and Switch Vectors

When an error indication is produced by a fault-detection mechanism, it is necessary to command some switches in order to reconfigure the faulty unit out of the channel. These commands may be sent directly by the

fault-detection mechanisms or by the computer following the diagnosis process. For example, as soon as the watchdog monitor indicates an error, a signal is sent to all the servo self-monitor to order them to turn off the actuator switches. The units that actually control the switches can be different from the units that perform the detection or the diagnosis. For example, the control of the actuator switches is done by the servo self-monitor (and not directly by the watchdog monitor). So, a careful analysis of the system needs to take into account the fact that while signals may be sent to the servo self-monitors to order them to turn off the actuator switches, some of these monitors may be faulty and not command their switch to disengage the actuator. When switches are implemented in software, for example, the "do-not-use" flags for the inputs sensors, they are directly controlled by the mechanisms that perform the diagnosis, for example the computer. In general, one will assume that some switches have control mechanisms different from the diagnosis mechanisms. These control mechanisms will be called the switch control mechanisms.

So, the switches are controlled either directly by the diagnosis mechanisms or indirectly through the switch control mechanisms. In both cases, one car represent the actual commands issued to the switches by the binary variable s_j where s_j is one if the switch j is commanded to turn off;

$$_A\vec{S} = \langle {}_As_1, {}_As_2, \cdots, {}_As_k \rangle \text{ with } {}_As_j = 0 \text{ if switch j of channel A is not commanded to turn off,}$$

$$\text{with } {}_As_j = 1 \text{ if switch j of channel A is commanded to turn off,}$$

$$_B\vec{S} = \langle {}_Bs_1, {}_Bs_2, \cdots, {}_Bs_k \rangle,$$
$$_C\vec{S} = \langle {}_Cs_1, {}_Cs_2, \cdots, {}_Cs_k \rangle.$$

Similarly to the previous sections, $\vec{S}$ will refer to the set of the three channel switch vectors:

$$\vec{S} = \langle {}_A\vec{S}, {}_B\vec{S}, {}_C\vec{S} \rangle.$$

These switch vectors carry all the information concerning the control of the switches after diagnosis. From them, if one knows the state (fault-free or faulty) of every switch, one can deduce the actual system configuration.

IV.3 Detection Process

IV.3.1 Detection Range

Each fault-detection mechanism can not detect failures in all the units. For example the sensor self-monitors can detect failures only in the corresponding sensor. For each detection mechanism, one can list all the units for which it can detect the presence of failures. This will be called the range of the detector. In general, one can use a matrix notation to express the range of detection:

Units $\xrightarrow{N}$ Detectors

with

$N = [n_{ij}]$ with $n_{ij} = 0$ if detector j can not detect the presence of failures in unit i,

$n_{ij} = 1$ if detector j is able to detect the presence of failures in unit i.

In ARCS all the channels are identical, so it will be the same matrix for every channel. In systems with dissimilar channels, it will be required to use one detection matrix for every channel. It should be noted that the detection matrices indicate only if failures can be detected. Even if a detector can detect only some of all the failures that can occur in a unit, this unit will still be considered as part of the range of the detector.

IV.3.2 Complete Versus Incomplete Detection

While most of the fault-detection mechanisms can not detect every single failure in the units they cover, the detectors that use cross-channel comparison are able to achieve complete detection (at least for the first two failures). Thus, one needs to classify the fault-detection mechanisms as providing either complete or incomplete detection. The difference is important since most of the critical system states are reached following failures that are not detected. Furthermore, as it will be our goal to list all the possible fault-patterns along with their detection, diagnosis, switch and reconfiguration vectors, one needs to find all the possible detection vectors that can occur following a failure. If a detector does not achieve 100% detection, then we need to consider that it may or may not detect the failure.

Even for the detection mechanisms that use cross-channel comparison, 100% detection is not always guaranteed. For example, when two channels are down, the output cross-channel comparison in the third channel becomes a meaningless process (which is not in fact carried out). Thus, complete detection is still dependent upon the system state. In general, it is easy to list the conditions that must be satisfied for a fault-detection by cross-channel comparison to achieve 100% detection. This can be mathematically expressed as:

$$\text{channel state } {}_A\Omega = \langle {}_A\vec{F}, {}_A\vec{R} \rangle \xrightarrow{g} \{0, 1\}$$

with

$\langle {}_A\vec{F}, {}_A\vec{R} \rangle \xrightarrow{g} 0$ if, in state $\langle {}_A\vec{F}, {}_A\vec{R} \rangle$, detector j does not achieve

complete detection,

$\langle {}_A\vec{F}, {}_A\vec{R}\rangle \xrightarrow{g} 1$ if, in state $\langle {}_A\vec{F}, {}_A\vec{R}\rangle$, detector j achieves perfect detection.

IV.3.3 Implementation of the Fault-Detection Mechanisms

The fault-detection mechanisms need to use some of the units to perform their detection function. If these units are defective, then the behavior of the fault-detection mechanisms may possibly be affected. For example, the cross-channel output monitor that should always detect the first diasagreement between the channel outputs may not detect it if the computer that performs the comparison is itself faulty. It is also possible that a detector will give an erroneous error indication when the units used to perform the detection are faulty. For example, a computer failure may cause the cross-channel sensor monitor to decide that the sensor is faulty when, in fact, it is the computer that is faulty.

For each fault-detection mechanism, one can list the set of units that they use. This can be expressed in a matrix form:

Units $\xrightarrow{M}$ Detectors

$M = [m_{ij}]$ = Detector implementation matrix

with $m_{ij} = 0$ if unit i is not used to implement the function of detector j,

with $m_{ij} = 1$ if unit i is used to implement the function of detector j.

In ARCS, because all three channels are identical, there will be only one such detector implementation matrix. For systems with dissimilar channels, one matrix per channel will be required.

Given the channel failure configuration, the detector implementation matrix makes it possible to list all the detectors which may either produce incorrect error indication or fail to generate an error indication.

IV.3.4 Set of All Possible Detection Vectors

Given the system initial failure configuration (the $\vec{F}, \vec{R}$ vector) and an occurrence of an individual failure (individual failure dF), one can find the set of all possible detection vectors. The first step is to find all the detectors that can detect the failure (using the detection matrix). The second step consists in finding whether some detectors achieve complete detection (using the g function). The third step is to find the set of faulty units that are used to implement the relevant detection functions (using the detector implementation matrix). These three steps yield the set of all detection vectors that can follow that particular failure occurrence:

$_A\vec{F} \xrightarrow{dF} \{ {}_A\vec{T} \}$ = set of detection vectors for channel A that correspond to failure dF (in state Ω),

$_B\vec{F} \xrightarrow{dF} \{ {}_B\vec{T} \}$ = set of detection vectors for channel B that correspond to failure dF (in state Ω),

$_C\vec{F} \xrightarrow{d\mathcal{F}} \{ {}_C\vec{T}\}$ = set of detection vectors for channel C that correspond to failure $d\mathcal{F}$ (in state Ω).

For simplicity, we will use $\vec{T}$ to refer to the complete detection:

$$\vec{T} = \langle {}_A\vec{T}, {}_B\vec{T}, {}_C\vec{T} \rangle.$$

IV.4 Diagnosis Process

IV.4.1 Diagnosis Mechanisms

Sometimes, when a fault-detection mechanism gives an error indication, each channel needs to perform some diagnosis to locate the unit where the failure has occurred. For example, any error indication provided by a cross-channel monitor requires finding which channel is faulty. This can be quite simple, for example in the case of cross-channel servo monitor Diagnosis consists simply in finding which of three servos disagrees with the other two. On the other hand, when a cross-channel output monitor detects an error, the diagnosis is more complex since it is required that two of the three channels reach a consensus. Some other error indications do not require follow-up diagnosis. For example, an error indication in provenance of the watchdog monitor or the sensor self-monitor does not trigger any diagnosis since these fault-detection mechanisms monitor only one unit each.

In ARCS, all the diagnoses are performed by the channel computers. However, in general, one can assume that each channel has more than one diagnosis mechanism (up to m). Each diagnosis mechanism looks at only some of the indications provided by the channel fault-detection mechanisms. For example, error indications given by the servo self-monitors are ignored. The set of the error indications that a diagnosis mechanism looks at will be called the input range. Each diagnosis mechanism can locate the failure only inside a subset of the channel units (for example determine whether the failure occurs in the

computer of either channel A, B or C). So, the diagnosis performed by a diagnosis mechanism is not a complete system diagnosis. One can call a diagnosis output range the set of units which can be distinguished by a diagnosis mechanism (cf. section IV.2.3).

For every diagnosis mechanism, one can define:

input range = subset of the channel error indications,
output range = subset of the channel units,
input = error indications from the input range,
output = diagnosis vector,
diagnosis algorithm = function relating the output to the input.

For every system, it should be possible to obtain the input and output ranges of every diagnosis mechanism. This is available from the description of the diagnosis algorithms with which the system are provided.

IV.4.2 Diagnosis Algorithm

The process of going from an error indication to a diagnosis vector is called the diagnosis algorithm. Each diagnosis mechanism has one such algorithm (that can be either software or implemented in hardware). The exact mathematical description of the diagnosis algorithms may be quite complex. However, it is available from the system description (the software specifications relative to diagnosis). In the following, one will assume that the diagnosis algorithms perform mappings from the

detector indication vectors to the diagnosis vectors that are dependent only upon the system state before the failure occurrence:

$$\langle \vec{F},\ {}_{A}\vec{T} \rangle \xrightarrow[\text{for diagnosis mechanism } j]{\text{detection algorithm}} {}_{A}\vec{D}_j$$

$$\langle \vec{F},\ {}_{B}\vec{T} \rangle \xrightarrow[\text{for diagnosis mechanism } j]{\text{detection algorithm}} {}_{B}\vec{D}_j$$

$$\langle \vec{F},\ {}_{C}\vec{T} \rangle \xrightarrow[\text{for diagnosis mechanism } j]{\text{detection algorithm}} {}_{C}\vec{D}_j$$

with

$${}_{A}\vec{D}_j = \langle {}_{A}d_1,\ {}_{A}d_2,\ \ldots,\ {}_{A}d_n \rangle$$

with ${}_{a}d_i = 0$ if diagnosis mechanism j diagnoses unit i of channel A as fault-free,

with ${}_{a}d_i = 1$ if diagnosis mechanism j diagnoses unit i of channel A as faulty,

with ${}_{a}d_i = 2$ if diagnosis mechanism j is not intended to diagnose unit i of channel A.

IV.4.3 Implementation of the Diagnosis Algorithms

The diagnosis mechanisms can be faulty and, hence, the diagnosis vectors they produce may be different from those produced by fault-free mechanisms. For example a computer failure may cause the processor to decide that another channel is faulty when it is not so. Thus, one also needs to take into account the state (fault-free or faulty) of each diagnosis mechanisms to find all the possible diagnosis vectors that can be produced when an error indication is produced. This requires finding all the channel units used to run the diagnosis algorithms. For example

in ARCS, diagnosis is performed by the computers. In general, one can list all the units used by a given diagnosis mechanism to perform its function. Similarly to what was done for the fault-detection mechanisms, one can use a matrix notation:

Units ---M'---> Diagnosis Mechanisms

$M' = [m''_{ij}]$ = Diagnosis Implementation Matrix

with $m'_{ij} = 0$ if unit i is not used to implement the function of the diagnosis mechanism j,

with $m'_{ij} = 1$ if unit i is used in the implementation of the function of diagnosis mechanism j.

In ARCS, since all the channels are identical, there will be only one such diagnosis implementation matrix (which is extremely simple). For systems with dissimilar channels, one matrix per channel will be required.

IV.4.4 Set of All Possible Diagnosis Vectors

For each of the detection vectors that can be produced following a failure, (occurrence of failure dF while the system has a failure configuration given by $\vec{F}$), it is necessary to find all the possible diagnosis vectors that can result. This can be done in three steps:

find all the diagnosis mechanisms that look at the error indications of the detection vector,

for each one, find the diagnosis vector that would be produced should

the diagnosis mechanism be fault-free. This can be done using the description of the diagnosis algorithm.

find the diagnosis mechanisms that are affected by the failure configuration of the system (either by the failure that causes the diagnosis to be run or by all the previous failures). This is achieved by looking at the the system failure configuration and the diagnosis implementation matrix, M'. The diagnosis produced by these diagnosis mechanisms can be incorrect. Thus, one needs to consider that every single diagnosis vector can occur.

So, for each detector indication vector, one can find the set of all the possible diagnosis vectors that can result:

$$\langle \vec{F},\ {}_{A}\vec{T} \rangle \text{ -------> } \{ {}_{A}\vec{D}_1 \}, \{ {}_{A}\vec{D}_2 \}, \ldots, \{ {}_{A}\vec{D}_m \}$$
$$\langle \vec{F},\ {}_{B}\vec{T} \rangle \text{ -------> } \{ {}_{B}\vec{D}_1 \}, \{ {}_{B}\vec{D}_2 \}, \ldots, \{ {}_{B}\vec{D}_m \}$$
$$\langle \vec{F},\ {}_{C}\vec{T} \rangle \text{ -------> } \{ {}_{C}\vec{D}_1 \}, \{ {}_{C}\vec{D}_2 \}, \ldots, \{ {}_{C}\vec{D}_m \}.$$

For simplicity in the notations, one can say that a particular detector indication vector, $\vec{T}$, in a given system failure configuration, $\vec{F}$, yields a set of possible diagnosis vectors, $\{\vec{D}\}$:

$$\langle \vec{F}, \vec{T} \rangle \text{ -------> } \{\vec{D}\}.$$

IV.5 Switch Process

IV.5.1 Switch Control Mechanisms

As it was previously mentioned, some of the switches are controlled by the diagnosis mechanisms while others are under the control of the switch control mechanisms. The switch control mechanisms receive signals either from the fault-detection mechanisms or from the diagnosis mechanisms. In ARCS, all the signals received by the servo self-monitors have the same meaning: a request to turn off the actuator switch. If the servo monitor is fault-free, reception of any single signal will result in a command being sent to the actuator switch to disengage. However, one can conceive systems in which the switch control mechanisms are also provided with some intelligence to resolve possible inconsistencies between the incoming signals. In general, one can say that each switch control mechanism receives inputs from some fault-detection mechanisms (some components of the detector indication vectors) and from the diagnosis mechanisms (some components of the diagnosis vectors) and controls the switch according to some function of the input signal (for example reception of any signal results in a command to the switch to turn off):

For every switch control mechanism, one has:

inputs = subset of the detector indication vectors
+ subset of the diagnosis vectors,

outputs = {stay on, turn off} commands to the switches

function = relation between the inputs and outputs.

For every system, it is possible to list all the switch control mechanisms and to find their inputs and the function they implement. This is available from the hardware and software description of the channels. So, for every detector indication and diagnosis vectors, it is possible to find what should be the commands received by all the switches should the switch control mechanisms be fault-free:

$$\langle \vec{T}, \vec{\mathcal{D}} \rangle \text{ -------> } \langle {}_A\vec{S}, {}_B\vec{S}, {}_C\vec{S} \rangle = \ .$$

IV.5.2 Implementation of the Switch Control Mechanisms

For a complete study, one needs to find which channel units are used to implement the function of the switch control mechanisms. Similarly to what was done for the fault-detection mechanisms and the diagnosis mechanisms, one can use a matrix notation to represent which units are used by the switch control mechanisms:

Units $\xrightarrow{M''}$ Switch switch control mechanisms

$M'' = [m''_{ij}]$ = Switch Control Implementation Matrix

with $m''_{ij} = 0$ if unit i is not used in the implementation of the switch control mechanism j,

with $m''_{ij} = 1$ if unit i is used in the implementation of the switch control mechanism j.

IV.5.3 Set of All Possible Switch Vectors

For each combination of detection and diagnosis vectors, it is necessary to find all the possible combinations of commands that are received by the switch (this in order to find all the possible system configurations). This can be performed in two steps:

find all the switches that are controlled directly by the diagnosis (or detection) mechanisms. For each one, find the switch command that corresponds to the particular diagnosis (detection) vector.

find all the switches that are controlled through a switch control mechanism. Find the switch control mechanisms that are affected by the failures the system has suffered. This can be obtained from the system configuration vector, $\vec{R}$, by using the matrix M".

So, for every combination of a detection vector, $\vec{T}$, and a diagnosis vector, $\vec{D}$, one can find the set of all the possible commands that are issued to the switches:

$$<\vec{F},\ {}_A\vec{T},\ {}_A\vec{D}> \text{-------->}\ \{{}_A\vec{S}\},$$
$$<\vec{F},\ {}_B\vec{T},\ {}_B\vec{D}> \text{-------->}\ \{{}_B\vec{S}\},$$
$$<\vec{F},\ {}_C\vec{T},\ {}_C\vec{D}> \text{-------->}\ \{{}_C\vec{S}\}.$$

For simplicity in the notations, one will write:

$$<\vec{F},\ \vec{T},\ \vec{D}> \text{-------->}\ \{\vec{S}\}.$$

IV.6 Reconfiguration Process

IV.6.1 Final System State

The commands issued to the switches along with the state of the switches dictate completely the system configuration (the $\vec{R}$ vector). Switches can fail in such a way that either they ignore the commands or they turn off without being issued any command. The first kind of failure can be called unsafe failure since it is far more dangerous for a faulty unit to be enabled than for a fault-free unit to be accidentally disabled.

Given the switch vectors and the knowledge of the switch states, one can find the system configuration:

$$\langle {}_A\vec{F},\ {}_A\vec{S}\rangle \text{ --------> } {}_A\vec{R},$$

$$\langle {}_B\vec{F},\ {}_B\vec{S}\rangle \text{ --------> } {}_B\vec{R},$$

$$\langle {}_C\vec{F},\ {}_C\vec{S}\rangle \text{ --------> } {}_C\vec{R}.$$

which can also be noted as:

$$\langle \vec{F},\ \vec{S}\rangle \text{ --------> } \vec{R}.$$

IV.6.2 Consistent Static System States

Section III presented a method to list all the possible static system states with a given criticality. However, as it was mentioned, some of these states could never occur in the actual system. This present section gives a method to find which static states can actually occur.

Furthermore, and even more importantly, it also characterizes the fault-patterns that can drive the system in these dangerous states.

Fault-patterns were defined as ordering of individual failure occurrences:

$$\vec{F}_0 \xrightarrow{dF_1} \vec{F}_1 \xrightarrow{dF_2} \vec{F}_2 \xrightarrow{dF_3} \ldots \xrightarrow{dF_x} \vec{F}_x = \vec{F}.$$

For each individual failure in the fault-pattern, it was hown how to find all the detection vectors that could possibly be produced:

$$\langle \vec{F}, dF \rangle \text{ -------> } \{\vec{T}\}.$$

Then, for each detection vector, one can find all the possible diagnosis vectors that are produced:

$$\langle \vec{F}, \vec{T} \rangle \text{ -------> } \{\vec{D}\}.$$

For each combination of a detection vector and a diagnosis vector, one can find all the possible combinations of commands that are issued to the switches:

$$\langle \vec{F}, \vec{T}, \vec{D} \rangle \text{ -------> } \{\vec{S}\}.$$

Then, it is quite simple to find the system configuration from the switch vector and the failure states of the switches:

$$\langle \vec{F}, \vec{S} \rangle \text{ -------> } \vec{R}.$$

Thus, one can find all the possible configuration that a failure pattern can induce on the system. The condition for a static state to be

consistent is that the system configuration, the $\vec{R}$ vector, can be the result of at least one fault-pattern.

IV.6.3 Characterization of the Critical Failure-Patterns

A critical fault-pattern is one that leaves the system in a critical static state. One can characterize fault-patterns by the detection, diagnosisa nd switch vectors associated with each individual failures. This provides all the information relating to the detection, diagnosis and isolation of every individual failure in the pattern. In general, a critical fault-pattern will be listed as:

$$\langle \vec{F}_0, \vec{R}_0 \rangle \xrightarrow[\substack{\vec{T}_1 \\ \vec{D}_1 \\ \vec{S}_1}]{dF_1} \langle \vec{F}_1, \vec{R}_1 \rangle \xrightarrow[\substack{\vec{T}_2 \\ \vec{D}_2 \\ \vec{S}_2}]{dF_2} \cdots \xrightarrow[\substack{\vec{T}_x \\ \vec{D}_x \\ \vec{S}_x}]{dF_x} \langle \vec{F}_x, \vec{R}_x \rangle = \langle \vec{F}, \vec{R} \rangle$$

IV.7 A Method to List All the Failure-Patterns with a Given Criticality

Starting from the enumeration of the critical states with a given criticality, the enumeration of the corresponding critical fault-patterns can implemented as a computer program.

IV.7.1 First Step

The first step consists in listing all the possible fault-patterns that correspond to a given failure configuration. The failure configuration, $\vec{F}$, that should be considered first are those which are part of the most critical static states. The list of fault-patterns that correspond to a given failure configuration $\vec{F}$, depends upon what is considered an individual failure (whether individual failures affect only one unit or more). The fault-patterns that correspond to a given failure configuration will be called potentially critical.

IV.7.2 Second Step

The first part is to find all the fault-detection mechanisms (whether hardware or software).

For each fault-detection mechanism, one needs to list all the units for which detection is possible. This gives the matrix N.

It is also required to find whether there are detection mechanisms that can achieve perfect detection. In general, only the mechanisms that use cross-channel comparison can achieve perfect detection. For each one of them, one should also find under which conditions perfect detection is

achieved.

For each fault detection mechanism, one needs to find all the channel units that are used to perform the detection (the matrix M).

Then, for each individual failure in a potentially critical fault-pattern, one needs to list all the possible detection vectors. This produces the list of detection vectors associated with a potentially critical failure.

IV.7.3 Third Step

The first part is to find all the mechanisms that are used to perform diagnosis (either local or global).

For each diagnosis mechanism, one needs to find which error indications trigger the diagnosis process and which is the diagnosis algorithm.

For each diagnosis mechanism, it is necessary to find all the system units that are used to implement this mechanism. This gives the matrix M'.

For each diagnosis mechanism, one needs to find all the possible diagnosis vectors that can be produced as a result of the detection vectors found in step 2.

IV.7.4 Fourth Step

The first part is to find how switches are controlled. For the switches that are controlled directly by the diagnosis mechanisms, one must find what commands are issued for each diagnosis vector found in step 3.

For the switches that are controlled through control mechanism, one needs to find the inputs, outputs and function of such mechanisms.

For every switch control mechanism, it is necessary to find which units are used to implement their function. This will give the matrix M".

For every combination of detection and diagnosis vectors that is produced by step 2 and 3, one needs to find all the possible combinations of commands sent to the switches. This, with the first part of step 4, will give all the possible switch vectors.

IV.7.5 Fifth Step

Given the switch vector and the system failure configuration, one can find the system configuration. If a potentially critical failure pattern (along with its detection, diagnosis and switch vectors) does not drive the system in a critical static state, then this fault-pattern is not critical (consistency operation).

This fifth step yields all the fault-patterns with their detection, diagnosis and switch vectors that drive the system from an initial fault-free state to a critical static state (cf. Fig. 10).

One should note that this procedure to enumerate the critical fault-patterns seems to involve much computation. However, since we are principally interested in the system tolerance to the first few failures, the total number of all the possible combinations of detection, diagnosis and switch vectors will be quite manageable.

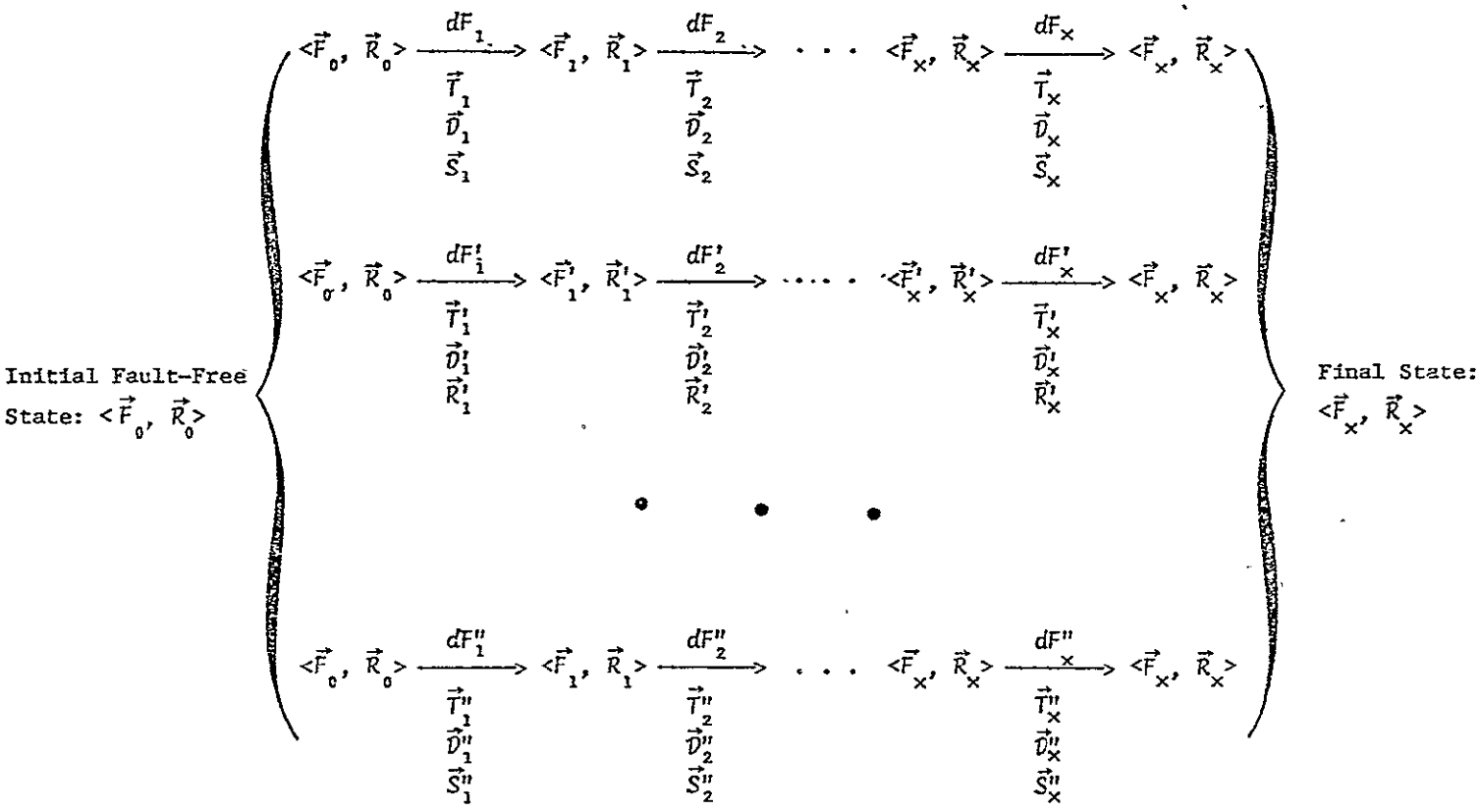


Fig. 10. Critical fault-patterns for a critical state.

V. CURRENT AND FUTURE RESEARCH

V.1. Computer Program for Enumerating the Critical Fault-Patterns

We are presently looking at the problems posed by a computer implementation of sections III. and IV. One of the major criteria is generality. Such a program should be applicable to any highly redundant computer system. For this reason, an interactive approach seems more appropriate.

The problem of computation complexity needs also to be carefully considered. Enumeration of the static states with a given criticality will be a fairly simple task since we are principally interested with the system tolerance to the first few failures. It is highly likely that most of the computations will take place when listing all the possible detection, diagnosis and switch vectors for each of the individual failures that are part of the potentially critical fault-patterns. Some reduction in the computation can be obtained by using the symmetry between channels. Another method to reduce the computation complexity is to eliminate some of the potentially critical static states before analysing each of the corresponding fault-patterns. For example, it is possible to eliminate some potentially critical system states by noting that some failures can never induce certain reconfiguration actions. Thus, a preliminary gross analysis of the range of possible effects for failures will significantly reduces the number of fault-patterns that need to be analyzed.

Another problem that is currently under study relates to the diagnosis algorithms. Such diagnosis algorithms are substantial pieces of software. In the case of ARCS, the diagnosis process following an error indication in provenance of the cross-channel output monitors is fairly complex since it may lasts several time frames and involves cross-channel communication. One possibility is to allow the program for critical fault-pattern enumeration to use the actual system diagnosis software as a subroutine. This way, the need to model the diagnosis process (and the corresponding loss of accuracy) is bypassed completely. However, unless the diagnosis software is available in a high level language, this will require to write a simulator.

The end goal of this enumeration program is to list all the fault-patterns, with their detection, diagnosis and switch vectors, that take the system from an initial fault-free state to a critical state (cf. Fig. 10). This will already provide a simple way to relate the system coverage to the first few failures to the efficiency of the fault-detection mechanisms, since we can list which of the undetected failures will cause a system crash. However, far more accurate system evaluation can be made if one can map the critical fault-patterns (as described by their detection, diagnosis and switch vectors) onto the physical hardware failures (for example line x stuck-at-one, ...).

V.2 Mapping Between Critical Fault-Patterns and Hardware Failures

We are presently investigating how to find all the hardware failures (described for example as line x stuck-at-one) which correspond to a given critical fault-pattern.

V.2.1 First Approach: Simulation

The first approach under consideration is to use a fault simulator. The enumeration program will provide the list of the most critical fault-patterns. Each individual failure in these critical fault-patterns is characterized by its location (at the unit level), detection, diagnosis and switch vectors. Thus, one can easily list the subset of the system units where a failure can correspond to the first individual failure of one of the most critical fault-patterns. Then, one can simulate each one of these units. The simulation should include the possibility to establish whether failures are detected. The simulation can be either a gate level simulation, if one wishes to characterize the faults down to the gate inputs and outputs, or a higher level simulation (for example at the I.C. chip level). Many such fault-simulators have been developed in relation to test generation and some are commercially available (H.P. TESTAID system for example).

If the unit that is simulated is provided with some hardware fault-detection mechanism, the simulation will also provide the list of failures that are not detected. If the unit is tested throught software tests (for example the processor is tested by the test routines), then,

we can run the softare tests on the simulated unit. This way, one can get a list of all the failures that will have a detection vector that matches the detection vector specified in the characteristics of the critical fault-patterns.

If a failure has the detection vector specified in the fault-pattern characteristics, then one needs to look at the diagnosis and switch control processes. If the unit that performs the diagnosis is fault-free, then one can run the diagnosis program for this particular detection vector and find out whether the failure is correctly diagnosed. If the diagnosis mechanism is itself affected by the failure (for example if the failure is in the processor), then, one needs to run the diagnosis program on the faulty machine. This can also be done with the simulator. Similarly, one can also simulate the switch control process and decide whether a particular failure has diagnosis and switch vectors that match those specified in the fault-pattern characteristic. This will yield the list of all the failures in a particular unit that correspond to the first individual failure of a critical fault-pattern. Then, one can use the same method to find all the failures that constitute the second individual failure of the fault-pattern.

This method may require a substantial amount of computation since it involves using a fault simulator. It is commonly assumed that a software implemented simulator runs about a thousand time slower than the actual hardware. However, it is believed that this approach is still a valid one. First, one will have to simulate only some of the units, and never the complete system (at the most, it will require

simulating one channel). Secondly, since the systems under consideration are highly reliable, it is likely that there will be only a few failures that will satisfy the conditions specified in the fault-pattern characteristics. Thus, when looking for the second individual failures of the fault-patterns, one will have only a very limited number of first failures to consider. One should also note that this approach is far more efficient that a straightforward method that would ignore completely the enumeration of the critical fault-patterns and simulate directly the system behavior for every single, double, and even triple failures.

V.2.2 Second Approach: Analysis of the Detection Process

One of the major problems with simulation is that it is necessary to simulate all the faults in order to find which ones are not detected. A different approach, that is currently under study, will avoid such a complete enumeration of all the failures.

This approach is based upon the observation that detection can be achieved either through hardware mechanisms (parity, encoding) or throught test sets. A hardware detection fault-detection mechanism will yield an error indication if the circuit ouputs invalidate a certain condition (fig. 11). For example, a parity checker will trip if the values on the circuit outputs do not sum up to zero (mod. 2). Similarly, a test set will indicate a failure if the circuit output values, for the inputs specified in the test set, do not match the correct values. So, in both cases, an error indication will be generated if some boolean conditions are not satisfied. Failures escape detection if the faulty circuits still satisfy the conditions but yet, they produce erroneous outputs. For example, one can state the boolean conditions that must be satisfied for a failure to escape detection by the parity checker for the circuit of Fig. 11:

let X represent the input vector,

let z_1, z_2, z_3 and z_4 be the circuit outputs when it is fault-free,

let z'_1, z'_2, z'_3 and z'_4 be the circuit outputs when it is faulty,

then, the conditions for a failure to go undetected are:

$z'_1 \oplus z'_2 \oplus z'_3 \oplus z'_4 = 0$ for every value of X,

and there exist X such that

$$(z_1 \oplus z'_1) + (z_2 \oplus z'_2) + (z_3 \oplus z'_3) + (z_4 \oplus z'_4) = 1.$$

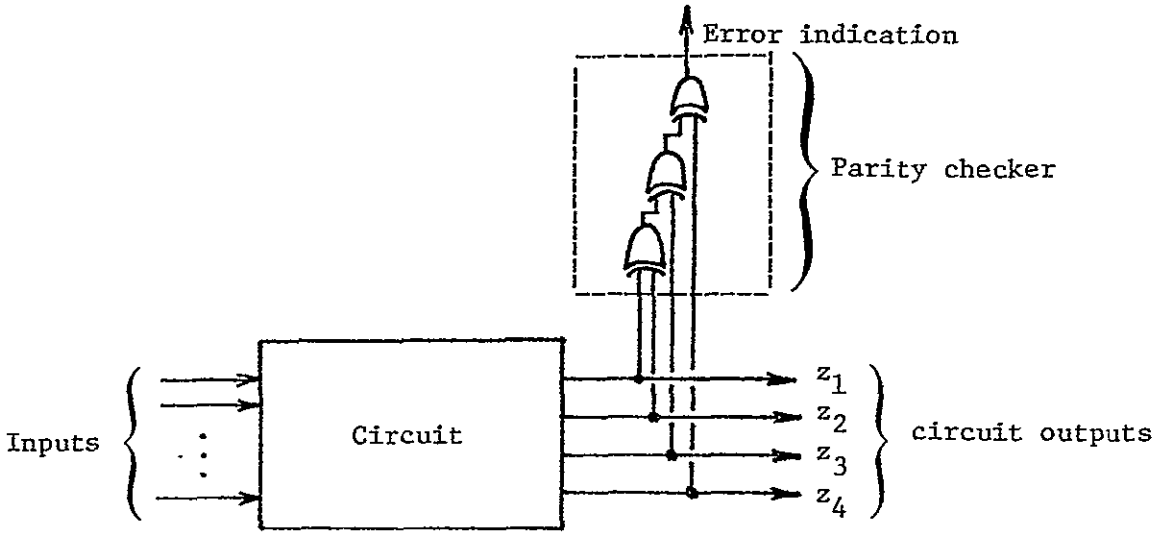


Fig. 11. Example of a circuit with parity checker.

For combinational circuits, one can express the circuit faulty behavior (the output functions) as a function of the fault-free functions and the faults if one replaces every line by an AND and OR gates to take into account the fact that the line can be fault-free, stuck-at-one or stuck-at-zero. Fig. 12 gives an example of such a transformation. Thus, the functions z' can be expressed as functions of the fault:

$z' = z'(X, F)$

with $F = \langle f_1, f_2, \ldots, f_k \rangle$,

$f_i = 00$ if line i is fault-free,

$f_i = 01$ if line i is stuck-at-one,

$f_i = 10$ if line i is stuck-at-zero.

Such a description of the faults allows the use of boolean manipulations to find the faults that escape detection (for which the circuit does not satisfy the detection conditions). The problem is analogous to solving a boolean expression, since failures can be represented as extra inputs to the circuit.

We are presently studying the relations between the faults with respect to detection. This will allows to reduce significantly the computations. We are also investigating methods to scan circuits for the faults that escape detection. The end goal is to be able to list all the failures that escape detection with a single scanning of the circuit. This involves studying the propagation of the detection conditions throught the circuit.

In a parallel effort, we are trying to obtain a very fine partitionning of the units in terms of subunits which would be quite independent with

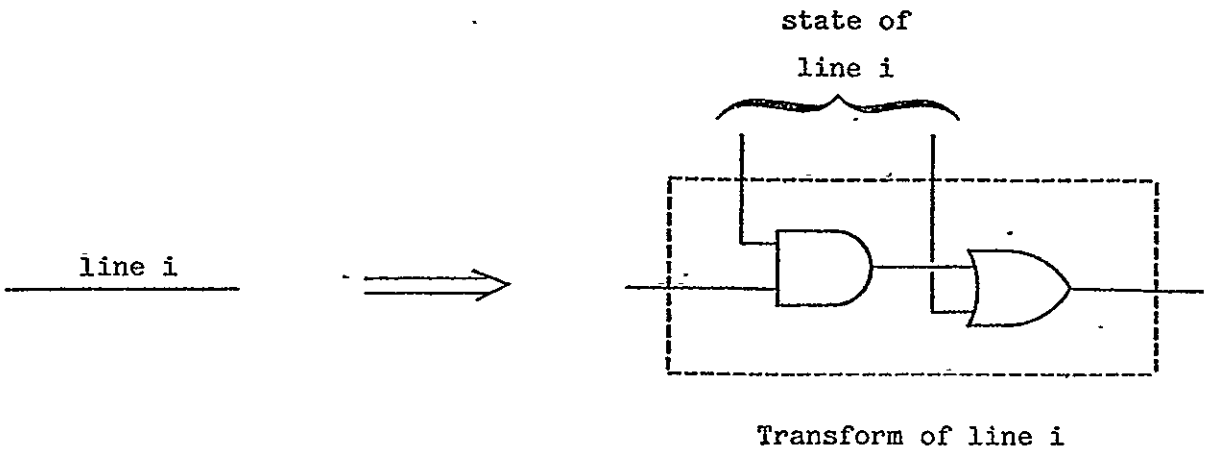


Fig. 12-a. Transformation of a line to reflect possible line failures.

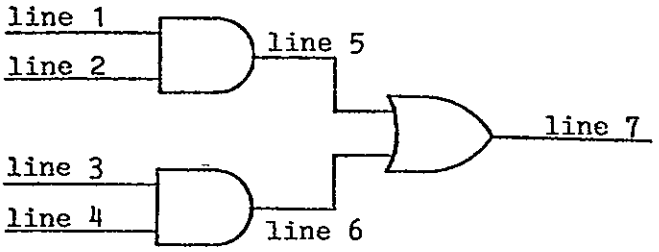


Fig. 12-b. Example of a circuit.

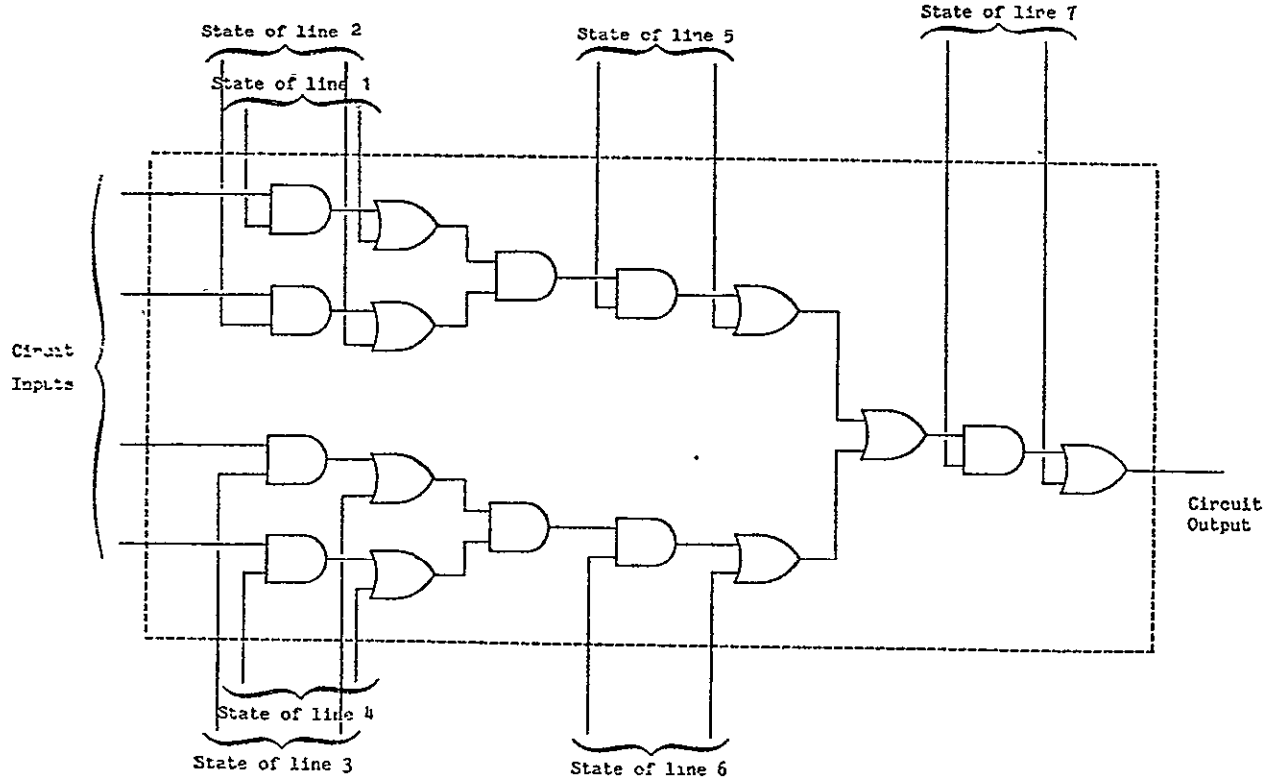


Fig. 12-c. Transformed circuit.

Fig. 12. Circuit transformation to take into effect possible failures.

respect to failure detection. This will decrease significantly the complexity of the circuits for which one needs to find the undetected failures.

VI. REFERENCES

[Bjurman, 1976] Bjurman, B.E., G.M. Jenkins, C.J. Masreliez, K.L. McClellan, and J.E. Templeman, "Airborne Advanced Reconfigurable Computer System (ARCS)," NASA Contractor Report 145024, 1976.

[Boute, 1972] Boute, R., "Equivalence and Dominance Relations between Output Faults in Sequential Machines," Tech. Rep. No. 38, SU-SEL-72-052, Nov. 1972, Stanford University, Stanford, California.

[Siewiorek, 1971] Siewiorek, D.P., "An Improved Reliability Model for NMR," Tech. Rep. No. 24, Digital Systems Laboratory, Stanford University, Stanford, California, Dec. 1971.

[von Neumann, 1976] von Neumann, J., "Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components," Automata Studies (Annals of Mathematical Studies), C.E. Shannon and J. McCarthy, Ed., Princeton Univ. Press, Princeton, NJ, 1956, pp. 43-98.

1. Report No.	2 Government Accession No	3 Recipient's Catalog No
NASA CR-145352		

4 Title and Subtitle	5 Report Date
Critical Fault Patterns Determination in Fault-Tolerant Computer Systems	January 1978
	6 Performing Organization Code
7 Author(s) Prof. E. J. McCluskey - Principal Inves. J. Losq - Project leader	8 Performing Organization Report No SEL Project 24-77
	10 Work Unit No
9 Performing Organization Name and Address Stanford University Digital Systems Laboratory Stanford, CA 94025	11 Contract or Grant No NSG-1410
	13 Type of Report and Period Covered Contractor Report
12 Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, DC 20546	14 Sponsoring Agency Code

15 Supplementary Notes

NASA Technical Manager, S. J. Bavuso, FED, NASA Langley

16 Abstract

A method is proposed which attempts to enumerate all the critical fault-patterns (successive occurrences of failures) without analyzing every single possible fault.

The intent of this study is to develop a technique for predicting or measuring coverage parameters for fault-tolerant computing systems.

17 Key Words (Suggested by Author(s))	18 Distribution Statement
Coverage Fault Analysis Reliability	Unclassified - Unlimited

19 Security Classif (of this report)	20 Security Classif (of this page)	21 No of Pages	22 Price*
unclassified	unclassified	78	$6.00

* For sale by the National Technical Information Service, Springfield Virginia 22161