ELEVENTH EUROPEAN ROTORCRAFT FORUM

PAPER No. 60

H/W AND S/W REDUNDANCY TECHNIQUES FOR 90's ROTORCRAFT COMPUTERS

A. DI GIOVANNI and P. GARRO
SELENIA
POMEZIA, ITALY

SEPTEMBER 10-13, 1985 — LONDON, ENGLAND

THE CITY UNIVERSITY, LONDON, EC1V OHB, ENGLAND

ABSTRACT

Future trends in aircraft and rotocraft designs is to utilize computers for flight critical operations.

In this way it is possible, for example, to improve the rotocraft performances by operating with greatly reduced stability margins. But computers for these applications must have a reliability comparable to the major structures of the aircraft. Therefore computers correctly operating also in the presence of faults is a matter of great interest for avionic designers.

The paper describes the most commonly used techniques for the implementation of real time fault tolerant computers and their application made by Selenia in two avionic systems.

The paper moreover describes the error detection and error handling techniques used in the SL/AYK 204 airborne computer.

1.0 BASIC CONCEPTS

A fault tolerant system is a system which remains operational after an internal fault has occurred or an external input peripheral gives incoherent data. Many authors, to indicate such a system, refer to "fail operational" or a "high integrity" system, which means that the system will deliver the correct control signals at the correct time even when a fault has occurred internally in the system.

To permit system operation, in the presence of fault, a certain degree of redundancy is introduced in the system. In case of a fault in a redundant unit of the sistem, its role is taken over by others units, where a unit may be both a hardware component of the system or/and a software component.

To obtain the redundancy, the hardware and software units of the system may be duplicated or triplicated: in general more copies (identical or not) with the same function inside the system may be used, so that a faulty unit may be immediatly substituited by its copy.

Depending on the system requirements a different granularity of the redundancy may be used: for example it may duplicate or triplicate the whole computer or only its memory or a particular peripheral unit, etc.

1.1 H/W Reliability

As mentioned above, redundancy can be applied at many levels, but perhaps high level rendundancy is more easily managed and is cheaper than that applied to LSI or VLSI circuitry level. For this reason this paper considers only high level computer hardware redundancy.

1.1.1 Stand-by computers -

In many applications not requiring the concept of "high integrity", stand-by computers can be used to continue the processing in case of a fault in the master unit. The switchover from the master to stand-by computer can be carried out manually or automatically. In such a redundant system, the master processor may produce incorrect data during the period in which the fault occurs and the time the switchover occurs. Such a computer system is not considered a "high integrity" system, and should only be used in critical applications where its peripherals can perform a sort of data-check on data coming from the failed computers (intelligent peripherals).

Moreover the system is limited to those applications in which the state of the system can be resumed by the stand-by computer: for example by reading sensors or other peripherals, and the state is independent on the history of the system evolution (combinatorial system).

In applications in which the state of the system depends from its history (sequential systems), the following architectural solutions can be used.

In the "warm standby" solution a backup computer is used but the application functions are carried out by the current "master" computer only, while the transactions caused on the system state are reported into the standby.

After a fault in the master processor, the stand-by computer has its internal state condition such to continue the computation that the master has interrupted.

For a system to have a "high integrity" the following conditions must be respected:

- the switchover must be performed very quickly compared with the system time evolution
- the detection capability into the master and slave must be very high

Another architecture which allows a faster switchover and a more consistent state between the two processors is the "hot stand-by" solution. In this architecture the stand-by system is used as above, but the two computers operate at the same time on the same inputs, while the peripheral units are driven only by the master unit.

1.2 Software reliability

In fault tolerant computers the same reliability provided by the hardware must be found in the software. At present there is no generally accepted method to quantitatively determine software reliability, although several mathematical models have been proposed. Moreover a number of factors contribute to system reliability and the software designer should give them due consideration during the development process; they include at least:

- developement methodology
- program structure
- programming language
- coding standards
- instruction set complexity
- testability
- error detection capabilities
- system redundancy
- file and program protection systems
- software maturity
- software production support tools

The traditional and most used method to remove errors from software is testing. The skill and method with which this testing is designed and planned, greatly determines the probability of removing errors from the software.

Another method of improving the probability of having an error free software, is to use standard software modules. For example an operating system is subject to additional testing in its normal operation so it can be expected that there would be a high probability that all normal modes of operation would have been exercised and any error thereby discovered.

In the environment of computer control, much of the software can be generalized and reused in many different applications. Hence taking into account that normal operation of a software package is an extension of testing, we can safely say that mature software is more error free than new software.

Recently some authors have used the term "fault tolerant software" referring to software techniques originally described under the name of "recovering block". The idea is to divide a software block into two parts:

— the first composed of two or more programs all performing the same functions

— the second composed of a different acceptance test for each of the previous computational program.

The program "version 1" is first executed, followed by carrying out an acceptance test. If the acceptance test shows that the output generated by the program is satisfactory, then the block is exited and next part of the overall program is run. If, however, the acceptance test fails, then the next version of the program is run, followed by running another different acceptance test. The objective is to permit that at least one of the program versions succeeds in passing the acceptance test.

Several considerations concerning such an approach must be carried out:

— the different version of the program should not produce "side effects", i.e. corrupt data structures used by others program versions.

— the test program must be reliable, i.e. must not reject correct results; and therefore should be kept relatively simple.

— this approach is usually designed to correct undetected software errors, but in certain cases can also detect and correct hardware faults.

Using this approach, the amount of software to be developed is larger than in the classical approach, thus causing memory size to increase. The overhead associated with providing fault tolerance is very low unless a fault actually occours, because only the running of the acceptance test is the normal overhead.

Hovewer it is in certain cases difficult to design an acceptance test that propely checks for the correct output of a program without itself being so complex and consequently so large to cause more overhead and to be itself subject to errors that would reject the whole approach.

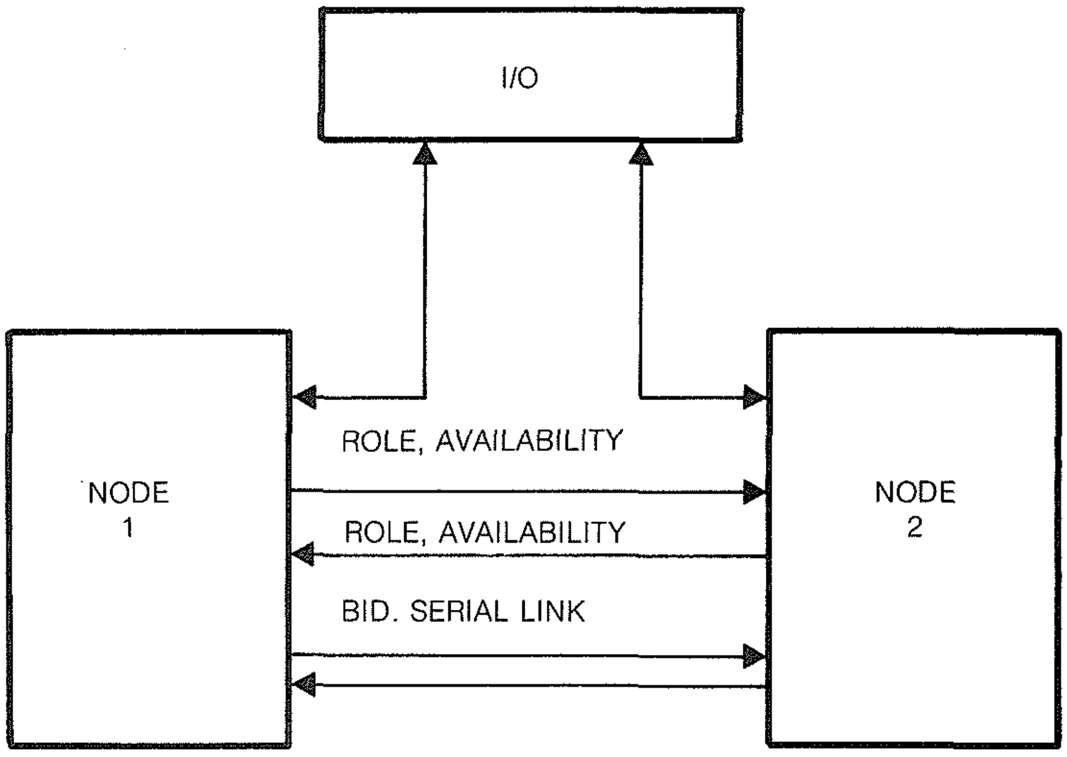
2.0 SELENIA APPROACH

This section summarizes the implementation of a highly reliable avionic computer based system developed by Selenia.

The implemented solution is the "stand-by" architecture based on a full duplication of the computer node. Fault detection is independently performed on both nodes. At every time only one of the two nodes, named the "master" controls the peripherals, while the other one, named the "slave" is always ready to substitute the Master in case of failure. Depending on the application requirements, the Slave node can be in "hot" stand-by mode or in a "warm" one. In both solutions the application software is supported by a run-time environment which makes transparent the duplicated configuration of the system to it.

2.1 The matched pair solution

In fig. 2 is depicted the Selenia implemented architecture for the stand-by solution, named "Matched-pair" configuration.



The two nodes of the pair are identical (in a Mono or Multiprocessor configuration) and are based on the "M.A.R.A." architecture. The two nodes communicate each other using a bidirectional serial data link, and using dedicated special lines to notify to the partner its own availability and role. Peripherals are independently connected to both nodes, that receive all incoming data, while only the currently selected master provides the output to the peripherals.

Each node has its own error detection hardware able to provide a high degree of error detection. Error detection is then separately performed on the nodes without making any matching between the computed results.

If the Master detects an error, an automatic role switching is performed. If the Slave detects an error, no role switching is performed and in presence of a second fault in the Master, the whole system is unavailabe due to its intrinsic capability to tolerate only the single error.

The main difficulty in the implementation of a matched-pair configuration is the role switching, after which the new Master must resume the peripheral output, starting from the point in which the old Master was interrupted. Therefore it needs to know exactly the point from which it is to continue the execution and part of the previous done computation: and hence the internal system state.

To obtain the state knowledge, Selenia implemented the two techniques above mentioned: namely the "warm" and "hot" stand-by methods. Before describing in more details the two solutions, some concepts should be made clear.

2.1.1 Stable Memory -

We call "Stable Memory" an ideal storage device for which the atomic operations "stable read" and "stable write" are defined. The two operations have the following characteristics:

1. if no error occurs during its execution, the operation is succesfully performed.
2. if an error occurs during its execution, there are two possibilities:
   - the operation is succesfully performed.
   - the operation is not executed at all.

2.1.2 Atomic Transaction -

The stable memory provides read and write atomic operations. To provide the application software to perform atomic read and write sequences, the basic software supports the following primitives, according to Lampson and Sturgis definitions:

- Begin transaction: this primitive must be used to start the atomic transaction;
- End transaction: this primitive must be used to request the succesful termination of all actions of the transaction;
- Abort transaction: this primitive must be used to abort all the actions of the transaction.

If a role switching occurs before the end transaction request, or if an abort transaction is called, the old states of the special objects updated by the transaction are restored.

If a role switching occurs after the end transaction request, all updates required by the transaction on the special objects are executed.

For "stable" special objects we mean the objects that are unaffected by the node failures, i.e. memory, special data structures, stable mailboxes, intelligent peripherals, ecc.

2.2 Warm Standby

In the warm standby solution it is the master that performs the application functions and updates, through the system state, the standby node that is performing only autodiagnosis.

The role switching is correctly performed by using a memory stable data base and atomic transactions.

Tha main features obtained by this solution are:

— the user writes an application program as a sequence of transactions:

— all the information requested to continue the execution after a role switching are contained in the stable data base;

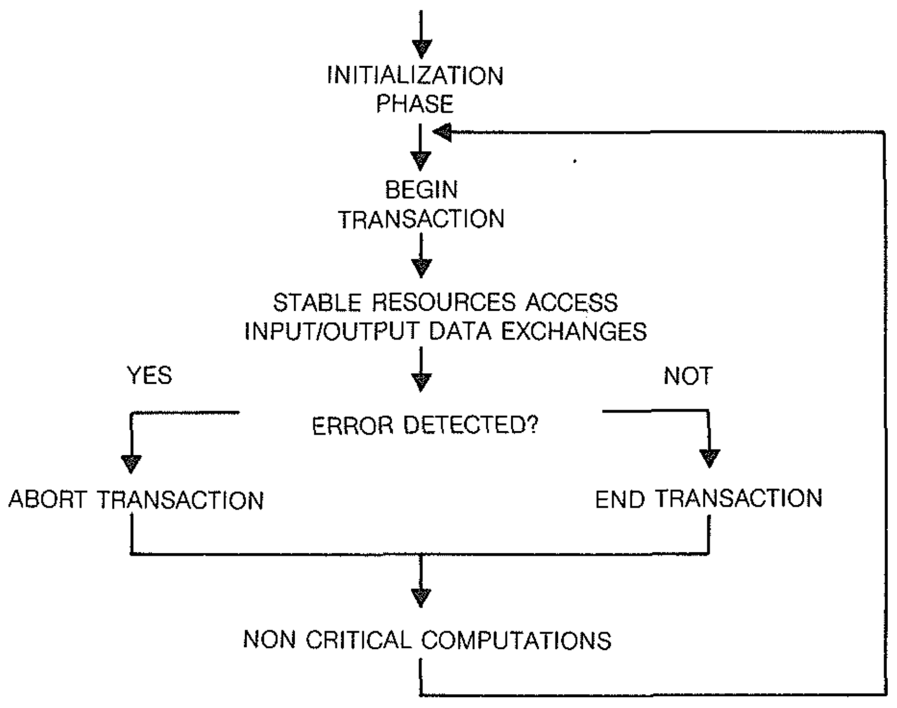
— during a transaction execution the user performs data base accesses, receives external inputs and sends output data to peripherals;

— data base changes and output data produced during a transaction will be definitively performed only after the end of a transaction request;

— if there is an abort transaction request, the data base is not updated so that the whole atomic transaction may be reexecuted;

— when a role switching occurs all the currently executing transactions are aborted to allow the new master to continue the system control by executing a new set of transactions starting from their initial point;

The following flow diagram shows the structure of a generic application transaction:



2.2.1 Peripherals Handling

Peripherals handling must be done to allow the repetition of the involved transactions.

2.2.1.1 Input Peripherals

If the data coming from the peripherals are stable i.e. a peripheral transmits continuously the block of data, a transaction may be repeated simply by restarting a read operation from the peripheral. The other way round if the peripheral sends data once, it is necessary to transform these data into stable data in order to allow the repetition of an aborted transaction in a faulty case. This can be achieved by using intelligent peripherals able to handle a repetition protocol (if requested), or by storing the incoming data directly into the stable memory: in this case, however, a less heavy handshaking protocol must be used between the node and the peripherals, because the data need not to be changed until written into the stable memory.

2.2.1.2 Output Peripherals -

Data produced by a transaction must be buffered and transmitted to the output peripherals only when the end transaction request is executed. This can be done by the peripheral handler.

2.3 Hot Standby

In the hot standby solution both the master and the standby node perform the application functions using the same input data and each node autonomously updates its own internal state, but only the master drives the peripherals.

Compared with the warm solution the hot standby has the following improvements:

- master faults have no side effects on the standby node. In fact in the warm solution a fault in the master node can corrupt the stable data base even if the probability can be kept at very low levels.

- in the switchover event it is not necessary to reinitialize the data structures because the old standby, being active, achieves a consistent state. For this reason the switchover can be done in a reduced time.

However to optimize the hot standby solution, the following problems must be solved.

2.3.1 Synchronization of the pair -

The application software can execute non deterministic algorithms, but to obtain the same computation from the master and the standby node is necessary that they make the same choices every time at the same time. This means a need of synchronization between the two nodes through a bidirectional serial link handled by the operating system of each node.

Typical non deterministic sources in the application processes are the synchronization primitives inside each node, such as loocks, semaphores, messages exchange, ecc.

This problem can be solved by performing all the synchronization primitives inside the operating system which must wait for the same synchronization primitive executed on the other computer.

Another non deterministic source can be the commonly accessed memory areas between tasks without synchronization.

This kind of access cannot be controlled by the operating system and its use must be prohibited.

The non deterministic handling causes a decrease in synchronization efficiency owing to:

— communication handling between master and standby;

— delay increase on the internal synchronization primitives due to the synchronization with the partner.

It is to be noted that in the hot standby solution the application software can be more transparent to the matched pair configuration with respect to the warm standby, because it is not necessary to structure it into atomic transactions. However, all the data structures in common between tasks must be handled via the operating system.

2.3.2. Peripheral Handling -

2.3.2.1 Input Peripheral -

Input peripherals, during the normal system evolution, must provide the two nodes with the same data in the same order. A fault in a node must not cause side effects on the input peripherals, i.e. the correctly working node must be fed by correct data.

2.3.2.2 Output Peripherals -

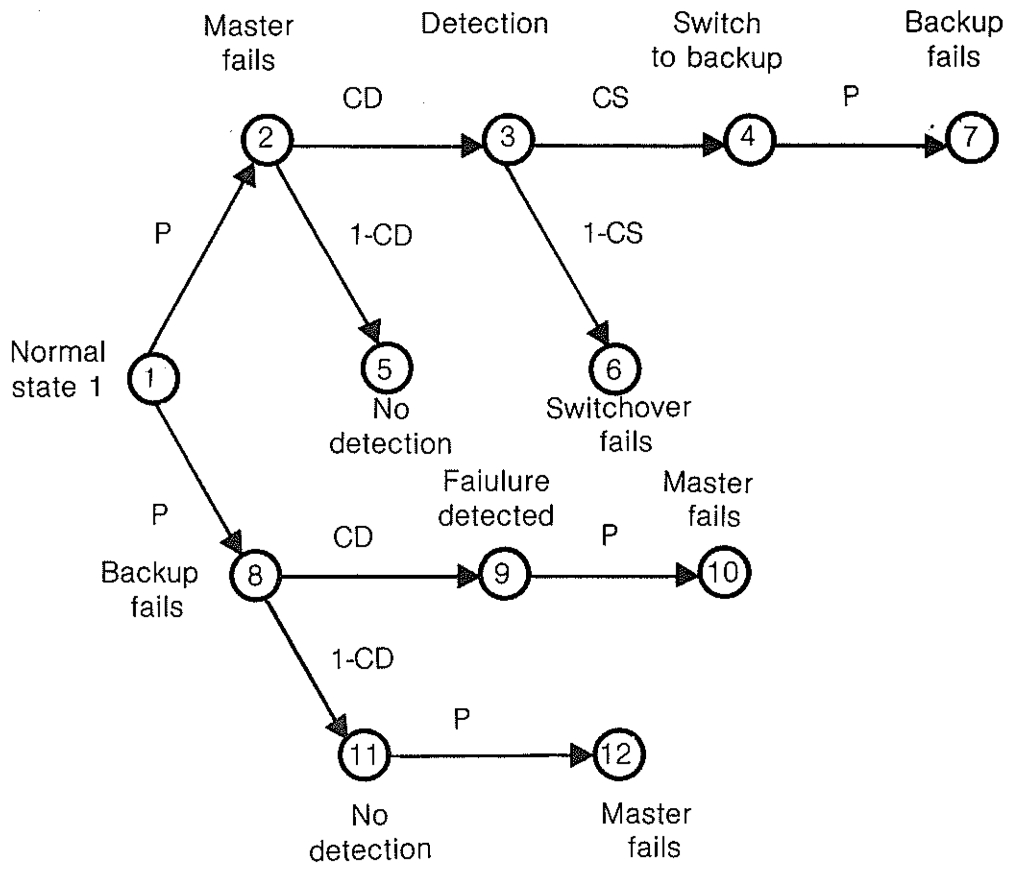
A failure in the master node, may interrupt the output data flow in any moment. Moreover the old standby can be asyncronous with the faulty master. Due to these reasons, to be sure that the new master resumes the output exactly from the point in which the old master was interrupted, it is necessary to structure the outputs to the peripherals into transactions, as above seen on the warm standby. This is, obviously, a transparency limitation of the application software for the matched pair.

3.0 RELIABILITY MODEL FOR THE PAIR

The described fault tolerant computer architecture pre-supposes that failures on the master can be detected, and that reliable switching to the backup is possible.

In practice, there are limitations on the probability that reliable fault detection and subsequent switchover can be accomplished.

In this chapter we define a finite state reliability model for the matched pair as reported by J.H. Wensley:



where:

— P is the rate (probability per time unit) at which one provessor will fail

— CD is the probability to detect a failure in a processor

— CS is the probability that a switchover is correctly performed

We can define the various states of the model:

1. both master and standby computers are correctly functioning
2. this represents the state when the master unit has failed. This is a transient state. Erroneous results can be produced by the system.
3. this is the state after which the master has detected, (with CD probability) that it failed. It has not yet switched to the standby. During this state erroneous results can be produced by the system.
4. this state represents a correct switchover to the standby computer which is now controlling the system. In this state correct control is achieved.
5. the system enters this state if the master does not detect the failure. In this case the total system failure can occur.
6. if the failed master detects the failure but switchover fails the whole system fails.
7. following state 4 if the standby fails the system enters state 7. The whole system is corrupted.
8. the matched pair enters state 8 if while in state 1 the standby fails. In this case the master continues to work correctly.
9. is the state in which the failed standby has detected its failure. Correct operation for the matched pair is mantained.
10. while the matched pair is in state 9 the master can fail. This causes a failure of the system.
11. the pair enters in this state if the standby fails but the failure is undetected. The system continues to function correctly.
12. while the matched pair is in state 11 the master may fail. This cause a failure of the system.

From this state diagram it is possible to observe the following characteristics:

— the system tolerates the single fault. in fact states 7,10,12 are terminal states.

— states 9 and 11 are states in which the system tolerates a failure of the standby computer.

— states 5 and 6 are critical for the system.

— the reliability performance of the system can be greatly increase by having a CD (probability of error detection inside a processor) very close to 1.

— the reliability performance of the system can be greatly increased by having a CS (probability of a correct switchover)very close to 1.

This analysis shows that great care must be taken when:

* implementing a powerful fault detection mechanism

* design the switchover mechanism.

4.0 SELENIA ERROR DETECTION AND SWITCHOVER MECHANISM

As seen in the above reliability model it is possible to increase the "pair" reliability, by having a powerful error detection and switchover mechanism.

4.1 Error Detection

Error detection in a SELENIA computer node is implemented by means of control hardware which monitors the correct functioning of that node and by periodic on-line diagnostic procedures.

Each application task has its own separate virtual address space and runs in a "protected" environment. Protection is required to prevent programs from improperly using unauthorized code or data. Each code or data segment has a particular "privilege level" assigned to it which determines which procedure can access the segment.

This protected enviroment is primarily oriented to the immediate detection of a hardware fault or of a software error which can corrupt the system. Minimization of the error detection time is very useful for the system integrity: in facts, as seen in the reliability model, during the transient states 2 and 3 erroneous results can be produced by the system.

The error detection circuits are placed so as to achieve a high coverage in the hardware which has the highest failure rate such as memories, and in areas which have high usage such as data paths.

All connections between interface cards and standard peripherals include some forms of redundancy in the information flow, i.e. parity bit.

Three standard error types are related to the bus use; these are: protection violation, bus timeout and slave error, and the hardware which detects them is normally enabled during every bus cycle.

A protection violation arises when a processor attempts an illegal access. The processor itself is notified of the event and access is aborted.

A bus timeout occurs when an agent attemps an access to a slave peripheral and no slave peripheral activates the ready signal within a given time from the beginning of the bus cycle.

A slave error occurs when a slave is unable to perform the request read or write access correctly.

Moreover inside the processor module there are other particular circuits for error detection such as:

— power failure alarm signal which is activated by the power supply when a power failure occurs.

— processor inactivity time-out which is activated when the intercycle time of the processor is too high.

— non maskable interrupt service time-out which is activated when the time to service a non maskable interrupt is too higt.

— watch-dog time-out which is based on a software retriggerable timer, to test if the global activity of the processor is correct.

Those classes of error which are not detectable by using the above mentioned hardware circuitry are covered by on-line diagnostic procedures which continuosly run inside the processor.

4.2 THE AVAILABILITY CONCEPT

One of the basic requirements of modern control systems is the automatic isolation of faulty units to avoid error propagation and to have faster error recovery, when the adequate redundancy is available.

This concept is extensively implemented in SL/AYK computer nodes as well as in any environment based on SL/AYK nodes.

Within a node each processor unit contains availability logic based on:

— state flag

— error flag

A processor in the on-line state performs all functions in the node. The loss of its availability state can be caused by:

— its own fatal error

— software actions as a consequence of error previously detected by the above mentioned dedicated hardware

— software actions as a consequence of error detected by diagnostics.

All these actions cause the error flag to be set. Once the error flag has been fired the processor looses any access right to the node, so that no wrong action can be propagated in the common resources.

Two strategies can be implemented by the system designer, and SELENIA computer architecture offers adequate support for both of them through a special line on the bus, called the availabilities line:

— the first strategy does not attempt any dinamic recovery action within a multiprocessor node. The availability line is the logical AND of all the processor availabilities within a multiprocessor node.

— a second strategy, after one or more errors resulting in the unavailability of some modules, allows the remaining "healty" modules to decide whether to continue in a degraded environment. In this case the availability line is the logical OR of the availabilities for each of the processors of a node.

Independently from the adopted strategy, if a node has lost its availability, it is automatically disconnected from the enviroment with the hardware passivation of all critical outputs.

4.3 Switchover Mechanism

The foundation hardware of a node declares its state with two external discrete lines:

— the node availability state

— the "ROLE" master/standby state

and can receive the homologous signals from a partner node.

Direct coupling of two "partner" nodes is possible as well as indirect coupling via a control panel.

The active state of these signals corresponds to the availability and the master role in the matched pair configuration. The loss of the availability of the role switching of the master (master-standby) causes in the partner node the recovery action to gain the master state.

5.0 SELENIA IMPLEMENTED FAULT TOLERANT SYSTEMS

SELENIA has developed both warm standby and hot standby solutions to implement fault tolerant systems.

The warm standby solution, based on atomic transactions was implemented for a civil application using a redundant disk unit as stable memory.

A Disk File Management System, a Memory File Management System, a Mailbox Management System and a Peripheral Management System has been implemented, all supporting the above explained atomic transaction mechanism.

In the military avionic field a Stores Management System (SMS) for the AMX aircraft has been implemented and the main computer unit (MCU) for the EH101 helicopter is under development, both based on SL/AYK dual redundant computer architecture.

These are hot standby redundant systems because both applications require a fast switchover time and a shortest time in which the output data may be manteined inconsistent.

REFERENCE:

1. J.H. Wensley, A review of techniques for achieving high reliability in hardware and software. Paper presented AT ISA conference, Philadelphia october 1982.

2. J.H. Wensley, Reliability in batch control processes. Paper presented at ISA conference, Philadelphia october 1982.

3. P. Ciompi et al., A highly available multiprocessor system for real time applications. SAFE COMP. 83 Cambridge 20-22/9/1983.

4. P. Ciompi et al., A rendundant distributed system supporting atomic transactions for real time control. Arlington 6-8 dicember 1983.

5. J. Goldberg et al., Development and evaluation of a software implemented fault-tolerant (SIFT) computer: Sift operating system. SRI International Technical Report, april 1980.