1. Publication Nº	2. Version	3. Date	5. Distribution
INPE-3244-PRE/590		August, 1984	☐ Internal ☒ External ☐ Restricted

4. Origin	Program
DCA/DEA	SUBORD/MECB

6. Key words - selected by the author(s)

FAULT TOLERANCE
MULTIPROCESSING
ERROR DETECTION
ON-BOARD COMPUTER

7. U.D.C.: 681.3.012:629.7.05-192

8. Title INPE-3244-PRE/590

A FAULT-TOLERANT MULTIPROCESSING UNIT FOR ON-BOARD SATELLITE SUPERVISION AND CONTROL

10. Nº of pages: 07

11. Last page: 06

12. Revised by

Eduardo W. Bergamini

9. Authorship Alderico R. de Paula Jr.
Ricardo C.O. Martins

13. Authorized by

Nelson de Jesus Parada
Director General

Responsible author

14. Abstract/Notes

This work presents a standard for fault-tolerant 16-bit multiprocessing systems to be utilized on board of satellites for their supervision and control. The systems, defined by the proposed standard, are characterized by their modularity, easy adaptation to different satellites and tolerance to single-point failures. The methodology for the development of such a standard is also oulined. This is based on the organization of the fault detection, analysis/confinement and recovery techniques according to the logical hierarchical layers of the system. The proposed multiprocessing system is constituted by a set of 16-bit multiprocessing units interconnected by redundant busses . The techniques for fault-tolerance in a tupical multiprocessing unit are presented stressing their structural interrelationship according to the following system layers: hardware, operating system and applicative processes.

15. Remarks

This paper will be presented in the "5º Congresso Brasileiro de Automática - 1º Congresso Latino-Americano de Automática", Campina Grande, PB, September 3-6, 1984

A FAULT-TOLERANT MULTIPROCESSING UNIT FOR ON-BOARD SATELLITE SUPERVISION AND CONTROL

Alderico Rodrigues de Paula Junior
Ricardo Corrêa de Oliveira Martins

Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq
Instituto de Pesquisas Espaciais - INPE
Departamento de Engenharia de Computação em Aplicações Espaciais - DCA
Caixa Postal 515 - São José dos Campos - SP - Brasil

Abstract

This work presents a standard for fault-tolerant 16-bit multiprocessing systems to be utilized on board of satellites for their supervision and control. The systems, defined by the proposed standard, are characterized by their modularity, easy adaptation to different satellites and tolerance to single-point failures. The methodology for the development of such a standard is also oulined. This is based on the organization of the fault detection, analysis/confinement and recovery techniques according to the logical hierarchical layers of the system. The proposed multiprocessing system is constituted by a set of 16-bit multiprocessing units interconnected by redundant busses. The techniques for fault-tolerance in a typical multiprocessing unit are presented stressing their structural interrelationship according to the following system layers: hardware, operating system and applicative processes.

Resumo

Este trabalho apresenta um padrão para sistemas de processamento distribuído tolerante a falhas, constituído de unidades de processamento de 16-bits interconectadas por barramentos seriais redundantes, a serem utilizados a bordo de satélites para sua supervisão e controle. O sistema definido pelo padrão proposto é caracterizado por sua modularidade, facilidade de adaptação em diferentes satélites e tolerantes a falhas simples. A metodologia utilizada para deteção, análise/confinamento e recuperação de erros é organizada em níveis lógicos hierárquicos. As técnicas de tolerância a falhas utilizadas em uma unidade típica de processamento são apresentadas, enfatizando-se o interrelacionamento entre as técnicas de tratamento de falhas a nível de circuitos, sistema operacional e processos aplicativos.

1. INTRODUCTION

Computers have been used more and more in satellites to replace analog circuits or non-programmable digital circuits. The main reason for this trend is that computers are flexible machines, in the sense that they can easily be adapted to different applications simply by changing the application programs and some interfaces.

Due to the development of VLSI technology, computer components are becoming more reliable, smaller, faster and the power consumption per gate is decreasing. Although the failure rate per gate has been decreasing, it is not expected that the failure rate will be negligible in the near future. Since computers have been used in critical parts of the satellites, a failure in the computer may disrupt the whole satellite. To avoid the loss of the satellite, the on-board computers should be designed to tolerate some class of faults.

One widely used general rule is that the on-board computer system has to be design to survive any single-point failure, without affecting the behavior of the satellite. Due to the difficulty of impossibility to repair the on-board computer externally when in orbit, the on-board computer should be able to reconfigurate itself. Therefore, the computer should be design to detect faulty units and to replace them by a spare automatically or under control of a ground station, and then restart the operation.

Another important characteristic of the on-board computer is that hardware redundancy should be limited as much as possible due to the limitations of power, space, and weight of the satellite. Therefore, a careful analysis should be done to determine which fault-tolerance techniques minimize

the hardware redundancy.

The main objective of this paper is to present a standard for fault-tolerant on-board multiprocessing systems. In Section 2 the methodology for the development of such a standard is oulined.

2. METHODOLOGY OUTLINE

An elegant and precise approach to handle malfunction in digital systems is to organize the error handling techniques, according to the logical hierarchical layers of the system. An example of how a typical on-board system can be organized in logical hierarchical layers is presented in Figure 1.

In this organization the ground station is the highest layer. It receives telemetry from the on-board system, indicating the system status and sensor data, and sends telecommands to the on-board telemetry and telecommand unit. This unit decodifies the telecommands and actuates directly on the subsystems, or passes the telecommand to the supervisor processing units. Similarly, the supervisor processing units can actuate directly on the subsystems, or passes the information to the master processing unit that controls a specific subsystem. The master processing units control the slave modules of the subsystems. Each one of these hierarchical layers may be subdivided again into other hierarchical layers as, for example, applicative process, operating system and hardware layers.

Faults may be generated at any layer as, for example, bugs in the application programs or failure of a gate in the hardware.

When it is "expensive" to detect some class of errors in the layer where it was generated, the error may be left to be detected at a higher layer. However, the undetected class of errors of a specific layer must be detected in a higher layer, to avoid the failure of the whole system by an undetected error. When an undetected error propagates to higher layers, an increasing amount of states and data structure may be affected, and consequently the recovery procedures become more complex and a longer time is required to recover the affected data structures. This is even more critical in multiprocessor systems, where the existence several concurrently executing processes can multiply the ocurrences of errors, since nonfailed components and processes can take incorrect, decisions based on erroneous information.

When a layer does not have the capacity to recover from a detected error, some error contention mechanism must be implemented. Consequently, the hardware should be organized in such a way that faulty modules can be isolated from the system. In addition, information about the detected error must be passed on to higher layers. Therefore, error detection and recovery mechanisms for the same class of errors may be implemented in different layers. The distribution of error handling techniques throughout the different logical layers of a system denotes the existence of trade-offs between fault tolerant design objectives (such as reconfiguration boundaries and speed of recovery) and system constraints (such as volume, power consumption, overall system reliability and available technology).

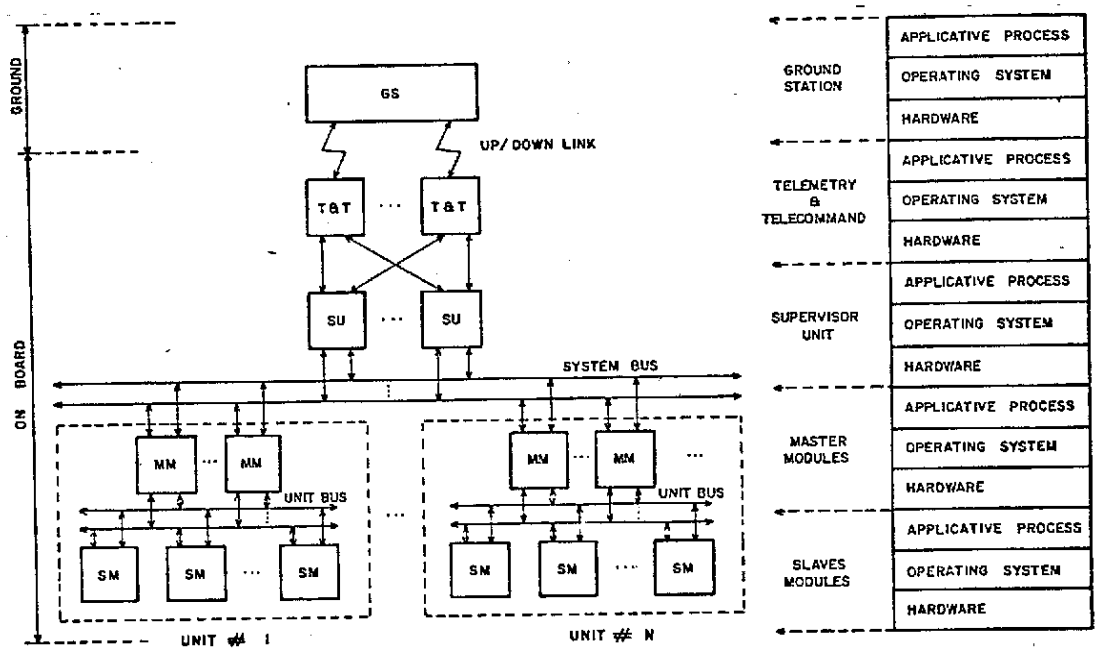

Fig. 1 - Organization of an on-board computer system at hierarchical layers.

For each error (or class of errors) in a given logical layer of a particular processing unit of the system, a coherent strategy for the design of the corresponding error handling mechanisms must include the following steps:

- Definition of the error detection mechanism at the same layer in which the error is originated or at an upper layer, based on the class of the task (critical, semi critical or non-critical) executed by the unit.
- Definition of the boundaries for error confinement and isolation. Once these are defined, the reconfiguration/replacement regions are defined, partitioning the unit into "modules". Each module (be it hardware or software) has a limited number (usually only one) of tightly controlled interfaces, to maximize the effectiveness of the isolation and recovery techniques.
- Definition of the recovery techniques to be used at the same layer in which the error is detected or at an upper layer. In the later case, mechanisms for reporting the error to upper layers must be designed.

3. THE HARDWARE LAYER

The proposed multiprocessing system is composed of a set of processing units interconnected by a set of redundant busses (system busses). The processing units are organized in two hyerarchical functional levels. The high-level units supervise the whole on-board system and controls the communication with the ground stations, while the low-level units are dedicated to specific satellite subsystems and perform data acquisition and control.

3.1 - THE MULTIPROCESSING UNIT ARCHITECTURE

A typical multiprocessing unit is partitioned into modules as depicted in Figure 2. The modules are interconnected by redundant busses (unit busses), each module having an unique address on the bus.

Each multiprocessing unit has a set of identical CPU modules, each CPU module being able to perform the functions of any other CPU module. In this way, redundancy and multiprocessing capabilities can be implemented, making it also possible to optimize the number of spare CPU modules for redundancy, since the function of any CPU module is determined only by the software it is running at the moment.

The management electronic module controls the power switching of the modules on the Unit bus. It receives power-on/off commands from the CPU module and keeps a record of the previously failed modules and of the presently powered-on modules, for recovery and reconfiguration procedures. The unit, as a whole, receives commands an transmits telemetry to the on-board system master (be it another computer or not) through the system bus interface.

The 16-bit CPU module (Figure 3) is designed to be connected to the Unit Bus through which all I/O operations are done. The CPU module is constituted by four submodules: Processor, Memory, Unit Bus Interface, and Error Handling. These submodules are interconnected by three parallel busses:

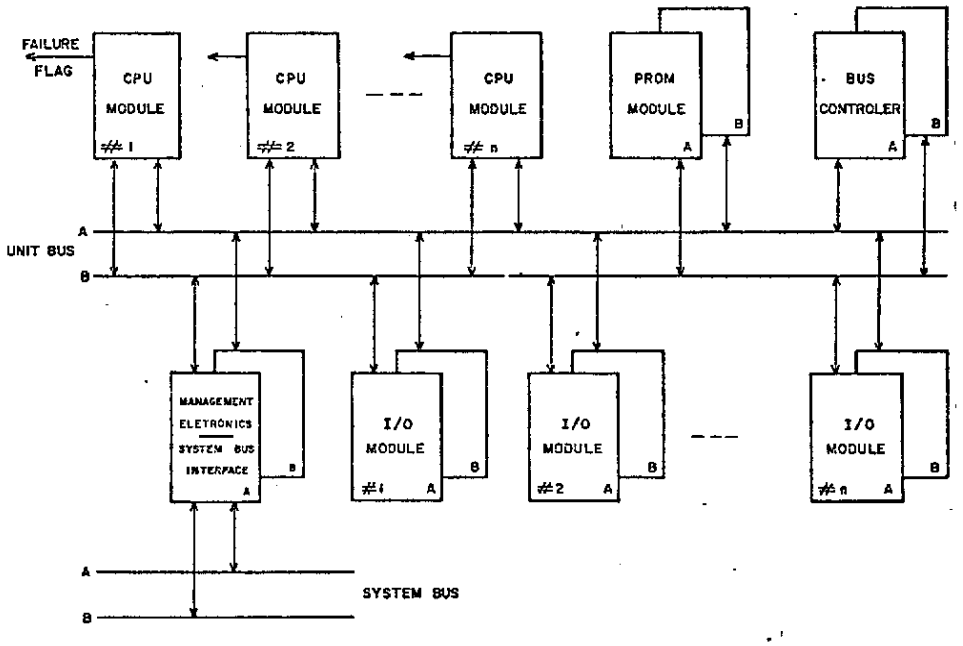

Fig. 2 - Multiprocessing Unit Organization.

Data Bus, Address Bus, and Control Bus.

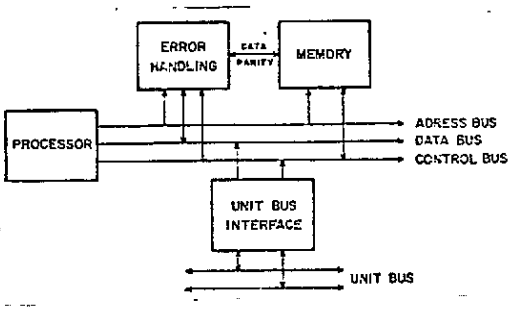

Fig. 3 - 16-bit CPU unit block diagram.

3.2 - ERROR HANDLING TECHNIQUES AT HARDWARE LAYER

The 16-bit multiprocessing unit and its operating system were design to detect the most probable classes of errors and to correct some of them automatically. Since the memory is the submodule most prone to be affected by cosmic particles, it is the submodule in which most of the error handling mechanisms were implemented at the hardware layer.

The RAM block is constituted by 22 4k x 1 CMOS RAM chips. The first 16 chips contain the processor word that is formed by two bytes, while the last 6 chips contain the parity bits necessary for detection of double errors and correction of single errors. Based on this organization, each RAM chip contributes to a single bit of the RAM word (22 bits). Therefore, any failure internal to a single RAM chip causes a single error in the RAM word. In order to detect a double error and to correct a single error, the modified Hamming code was used. The implementation of the Hamming code was accomplished by using an EDC chip.

During the RAM write cycle, the EDC chip generates the 6-bit modified Hamming code parity bits which are stored at the memory into the same address as the processor word. During the RAM read-cyclé, both the processor word and the parity bits are read from the memory and loaded into the EDC chip. Using the processor word, received from the memory, the EDC chip computes the new modified Hamming code parity bits and compares them with the parity bits received from the memory.

If a single error is detected, the EDC corrects this error and sends the correct word to the processor. At the same time, the address of the word that had an error and error syndrome bits generated by the EDC chip are loaded into the RAM error register, to be used by the error analysis/recovery process. Then, the interrupt request line is activated to inform to the operating system the occurrence of a single error at the memory.

The memory block can be formed by RAM chips or PROM chips, or it can remain free. Each of these blocks can be protected against a write access, by setting to zero the corresponding bit of the memory protection register. Before a process is taken to running state, the operating system defines the memory blocks that the process is allowed to write by loading the memory write-protection byte into the memory protection register. The PROM blocks and the free blocks that are defined in the beginning of the operation must be protected all the time.

During the memory write operation, the memory block enable signal is compared with the block protection bit. If they disagree, the failure flag flip-flop is set, the processor is taken to the reset state and the unit bus I/O reset line is activated.

The Error Handling Submodule has the function to handle the error detected at the hardware layer correcting some of them and informing the higher layer error handling mechanism the occurrence of errors. This module comunicates with the operating system layer of the processing unit by activating an interrupt request line and with external modules by raising the failure flag. This submodule contains a watch-dog timer which is designed to detect faults in the real time clock, or faults in the processing module hardcore (processor, or data and bus interfaces) that disturbs the CPU module in such a way that it is not able to execute the routine that reset the watch-dog timer. The watch-dog timer is discussed with more detail in the next section.

4 - THE OPERATING SYSTEM AND APPLICATIVE PROCESS LAYERS

The first and foremost goal of the operating system was to provide a highly modular structure for a typical fault-tolerant multiprocessing unit of the modular on-board microcomputer system. This was translated into the following requirements:

a) Error detection, analysis and recovery by software should be configurable (modular) depending on the mission. In the best case, an external master unit would be able to provide a great deal of the error recovery/reconfiguration. In the worst case, the multiprocessing unit would be expected to operate in a totally autonomous manner (such as in the case of a deep space probe).

b) The operating system should support any possible hardware configuration of the multiprocessing unit, ranging from a single CPU-module to a true multiprocesssing unit with several CPU-modules, part of them active simultaneously, and part of them as spare modules (cold standby redundancy).

c) The operating system should create and maintain a (software) processing environ

ment in which the CPU-module and the unit bus interface characteristics are transparent at the applicative process layer, i.e., processor and interface transparency should be implemented at that layer. Therefore, the applicative processes could be programmed independently of the specific CPU-module in which they will run. The set of applicative processes for a multiprocessing unit, which may contain an (a priori) unknown of active CPU-modules, should be designed so that it can be partitioned appropriately among those processors without extensive reprogramming.

d) A the operating system should support in-flight reprogramming of any applicative process.

4.1 - THE OPERATING SYSTEM STRUCTURE

To satisfy the requirements above, the operating system is composed of a set of identical nuclei, one nucleus for each CPU-module of the multiprocessing unit. In a multiprocessing unit, all the CPU-modules are also identical at the hardware layer (see the previous section).

The nucleus supports two primary abstractions, namely:

- Processes - which basically do the work, performing to the processing unit.
- Messages - which allow interprocess communication. These are also the unique mechanisms for interprocess synchronization. Some of the reasons for this decision are the requirements "b" and "c" specified in the previous section.

The nucleus is composed of:

- a set of interrupt service routines, which handle the interrupts generated at the hardware layer;
- a set of nucleus primitives for interprocess communication/synchronization, long-term scheduling, error-handling and processing and switching;
- the nucleus data structure, a set of queues, tables and specific variables that are managed only by the interrupt service routines and the nucleus primitives.

The handling of interrupts is an aspect of any operating system design that tends to destroy attempts to satisfy the requirements for hardware transparency (processor and interface transparency), such as the requirements presented above. The known solutions are almost always compromises in which the objectives may be violated, since the processes which interact with interrupt service routines often become bound to specific processors (CPU-modules in this report). To overcome this limitation, processor and interface transparency are not only implemented by the nucleus, but by the nucleus jointly with a set of "system processes" which interact directly with the interrupt service routines. These processes are device drivers that, jointly with the corresponding interrupt service routines, convert the device interrupts and register contents into messages. Thus, any applicative process receives data from the sensors or commands from the ground stations and sends telemetry and actuator commands out only by receiving or sending messages.

The nucleus supports two processing modes, namely:

- the normal mode, in which all the applicative processes that do the usual tasks assigned to the unit mus run;
- the privileged mode, in which only special tasks are executed, such as error analysis/recovery, reprogramming and long-term scheduling.

Mechanisms specifically for mode switching are provided by the nucleus through its primitives and interrupt service routines, as follows:

- some interrupt service routines, when detecting an error, do a mode switching and force the execution of a specific process (which runs only in privileged mode) for further analysis and/or recovery of the error;
- an applicative process, running in normal mode, can call a specific nucleus primitive for mode switching and execution of another process, which implements the long-term scheduling or error analysis/recovery techniques.

In the privileged mode, the interrupts are tightly controlled and, in some cases, part of them are masked most of the time. Some processes, such as the system processes cited above, can run in both processing modes. However, all the applicative processes that run only in the normal mode are not scheduled for execution, while the system is in the privileged mode. Therefore, they are kept "frozen" at their current state jointly with the corresponding nucleus data structure. In this way, while the system is in the privileged mode, an error analysis/recovery process can reconstruct a system state believed to be error free, and then can resume the normal system operation.

The nucleus supports two forms of process scheduling: the long-term and the short-term scheduling. The long-term scheduling must be carried out by a mission-dependent process, here called "scheduler". The scheduler implements the different mission phases by activating the applicative processes specific to each phase of the mission.

The short-term scheduling is carried out by the interrupt service routines jointly with the communication/synchronization primitives. This is a priority scheduling in which the CPU-module is allocated to the

process in the ready state with the highest priority.

The layered software structure outlined above, implemented and supported by the nucleus, is depicted in Figure 4. This layered organization is also stressed by the error handling mechanisms to be described in the next section.

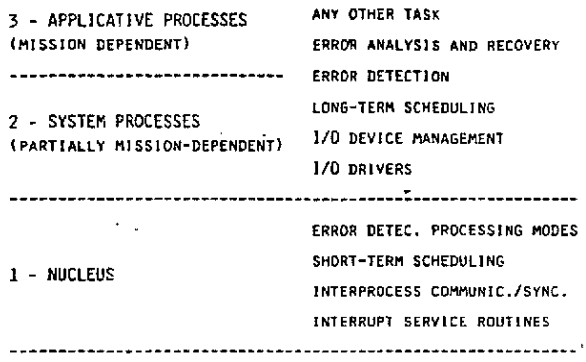

Fig. 4 - Operating System Layered Structure.

4.2 - ERROR HANDLING MECHANISMS

A form of consistency and destination checking, as proposed by Martins and De Paula (1983), is provided by some of the nucleus primitives and interrupt service, which implement and control the flow of messages among active processes. Each processes has a "buffering limit" as part of its control block, which determines the maximum number of message-buffers the process can have at a time. This limit as defined at the process design phase. Any attempt to exceed that limit at run time is detected as an error (process unexpected behavior) by the interprocess communication primitives or by the time interrupt routines, that serve the real time clock and unit-bus interrupts.

These are: no built-in deadline mechanisms in the nucleus synchronization and communication primitives. However, as the short-term scheduling implements a control cycle equal to the real-time clock interval, any process scheduled for execution (i.e., that reached the ready state) in a control cycle must complete its tasks before the arrival of the next real-time clock interrupt. The routine, that services this interrupt, executes a completion checking on all those processes. In this way, a global deadline mechanism is always active, checking the execution of any "ready" process.

Autodiagnosis tests are embedded in at least two "system processes", namely the "Unit Bus Manager" and the "RAM-Analyzer". Depending on the application, other autodiagnosis processes can be added to the system simply as applicative processes. The Unit Bus Manager performs a loop check in every I/O operation with a unit bus device, for autodiagnosis. The RAM Analyzer (triggered by the RAM single-error interrupt service routine) executes an diagnosis procedure and exercises the memory modules for permanent failure isolation.

The system may continue the operation after a permanent failure is detected in a RAM chip, by masking the single error interrupt request line. The degraded operation is possible until the occurrence of a double error. When the EDC circuit detects a RAM double error, it sets the failure flag flip-flop that takes the processor to the reset state until an external action is taken. To avoid the occurrence of double-point error in the RAM memory during the normal operation, the whole memory must be exerciced periodically.

Underlying all the error detection mechanisms implemented at the nucleus or at any process, there exists an error reporting and confinement mechanism provided by the nucleus through some of its routines. Error reporting mechanisms are provided by the interrupt routines that serve error generated interrupts (for errors detected at the hardware layer), and by the nucleus primitive that switches operation modes for errors detected at the applicative process layer. The appropriate error analysis/recovery process can then be executed in privileged mode. In this processing mode, some selected interrupt routines can be executed and selected system processes can be scheduled for execution, exercising and checking continuously the CPU-module and the nucleus. Such a simple scheme provides the necessary modularity for the implementation of configurable (depending on the mission) error recovery techniques.

Resource Control is provided through the nucleus routines that control the access to the common message buffer pool; and through the Unit Bus Manager, which controls the access to the unit bus.

5 - CONCLUSION

The standards for on-board supervision and control described in this paper is being utilized by SPAR Aerospace of Canada in the development of a 16-bit fault-tolerant on-board microcomputer system. The fault-tolerant methodology outlined in Section 2, and its structural organization of mechanisms for fault-tolerance are also being utilized at INPE, in the development of the so-called INPE Standard for On Board Supervision-(PISB), which is also based on 16-bit processing units. The latter system will be implemented in the Brazilian satellites being developed by INPE for the MECB mission. Both systems are now at conclusion phase of their first prototype.

6 - REFERENCES

Martins, R.C.O. & De Paula, A.R., (1983). "A Fault-Tolerant 16-Bits Multiprocessing Unit for On-Board Satellite Applications", SPAR Technical Report.