Fault Tolerant Solid State Mass Memory for Space Applications

G. C. CARDARILLI

M. OTTAVI, Member, IEEE

S. PONTARELLI

M. RE

A. SALSANO
University of Rome "Tor Vergata"
Italy

In this paper an innovative fault tolerant solid state mass memory (FTSSMM) architecture is described. Solid state mass memories (SSMMs) are particularly suitable for space applications and more in general for harsh environments such us, for example, nuclear accelerators or avionics. The presented FTSSMM design has been entirely based on commercial off the shelf (COTS) components. In fact, cost competitive and very high performance SSMMs cannot be easily implemented by using space qualified components, due the technological gap and very high cost characterizing these components. In order to match the severe reliability requirements of space applications a COTS-based apparatus must be designed by using suitable system level methodologies [1, 2]. In the proposed architecture error-correcting codes are used to strengthen the commercial dynamic random access memory (DRAM) chips, while the system controller has been designed by applying suitable fault tolerant design techniques. Different from other proposed solutions, our architecture fully exploits the reconfiguration capabilities of Reed-Solomon (RS) codes, discriminates between permanent and transient faults reducing the use of spare elements, and provides dynamic reconfiguration and graceful degradation capability, i.e., the FTSSMM performances are gracefully reduced in case of permanent faults, maintaining part of the system functionality. The papers shows the FTSSMM design methodology, the architecture, the reliability analysis, some simulation results, and a description of its implementation based on fast prototyping techniques.

Manuscript received May 7, 2004; revised March 25, 2005; released for publication May 20, 2005.

IEEE Log No. T-AES/41/4/860799.

Refereeing of this contribution was handled by M. Ruggieri.

Authors' current addresses: G. C. Cardarilli, S. Pontarelli, M. Re, A. Salsano, Dept. of Electronic Engineering, University of Rome "Tor Vergata," Via de Pontecnico 1, 00133, Rome, Italy, E-mail: (marco.re@ieee.org); M. Ottavi, Dept. of Electrical and Computer Engineering, Northeastern University, Boston, MA.

0018-9251/05/$17.00 © 2005 IEEE

I. INTRODUCTION

The designer of electronic systems for space applications should take into account a number of critical design constraints related to the harsh environment in which they operate. In fact, in the space environment electronic components are stressed by a number of physical phenomena, such as, for example, mechanical stresses, ionizing radiations, and critical thermal conditions. In order to deal with these issues the typical approach has been the development of space qualified electronic devices based on special and expensive technology processes. The use of such components implies some drawbacks such as high cost (due to the special technology and the low number of produced parts) and low performances compared with commercial off the shelf (COTS) components (due to limiting factors in radiation hard technology) [3]. On the other hand the use of COTS components push for a design based on suitable system level methodologies in order to match the severe reliability requirements of space applications. A typical case, where this approach is exploited, is the design of space-borne mass memories. In fact, the rapid growth in capacity of semiconductor memory devices permits the development of solid-state mass memories, which are competitive with respect to tape recorders due to higher reliability, comparable density, and better performances. Solid state mass memories (SSMMs) have no moving parts and their operational flexibility has made them suitable for many applications. Moreover, the requirements of low latency time, high throughput, and storage capabilities, cannot be satisfied by space qualified components and the choice of COTS is mandatory. The fault tolerant solid state mass memory (FTSSMM) presented here is based on COTS components. In [4] different coding schemes based on fixed Reed-Solomon (RS) codes, and hardware configurations to protect data stored in COTS-based memories are compared. However, in our architecture highly reconfigurability of RS codes is exploited to obtain a trade-off between reliability, data integrity, and overhead. The reconfigurability of RS codes is also used to achieve a graceful degradation of the system and the discrimination between permanent and transient fault allows to reduce the use of spare elements. Moreover, in our architecture a number of SpaceWire data links [5] accesses the memory banks through a crossbar switch matrix [6]. This solution has many advantages with respect to a bus-based architecture in terms of bandwidth, latency, and reconfiguration capability. In fact, the failure of a connection does not compromise the entire connection of the network but only the access to a specific node. In order to improve both the fault tolerance and the memory usage, a distributed file system has been implemented. Most of the functions performed by the file system are hardware based and handled locally on each memory module. This paper is organized as follows. Section II illustrates the used design methodology, Section III describes in details the FTSSMM architecture, while the reliability, data integrity, graceful degradation evaluation, and the used simulation methodology are reported, respectively, in Section IV and Section V. In Section VI a description of the prototype setup and a description of the used fast prototyping methodology is presented. Conclusions are drawn in Section VII.

II. DESIGN METHODOLOGY OF FAULT TOLERANT SYSTEMS

In this section the design methodology used for the implementation of the FTSSMM is presented.

The proposed design has been developed in order to cope with the typical fault set used to model the effects of radiations in space environment, i.e., the single event upset (SEU) faults, caused by ionizing particles, and stuck-at faults, related to the total ionizing dose (TID) [7, 8].

The fault tolerance design techniques applied, when COTS are used, are basically two: fault masking, e.g. triple modular redundancy (TMR) or, when the application needs low hardware overhead, fault detection, and dynamic system recovery techniques. The latter method satisfies the requirement of a lower hardware redundancy and lower power consumption, with respect to TMR technique, but, on the other hand, needs reconfiguration algorithms (software redundancy) and, when a fault occurs, implies an out-of-order time interval related to the mean time to repair (MTTR).

In order to choose the proper design strategies, each functional block has been evaluated in terms of the impact of its failure on the overall performance of the FTSSMM. In particular, the impact of permanent and transient faults after their detection have been evaluated. The functionalities of a block affected by a transient fault can be recovered, after its detection, simply reinitializing the hardware block. Therefore the impact of a transient fault is mainly related to the integrity of the stored data. On the other hand, permanent faults cause the unavailability of one of the implemented functions; therefore, their impact is mainly related to the performance assessment.

It is straightforward that the trade-off between reliability, power, and throughput is closely related to the requirements of the target application. In this design we pushed for obtaining the following objectives:

1) a scalable architecture which can be easily adapted to mission requirements,
2) high throughput achieved by choosing a highly parallel architecture,
3) capability to reduce the power consumption and failure rate of the memory by simply turning off the unused modules,

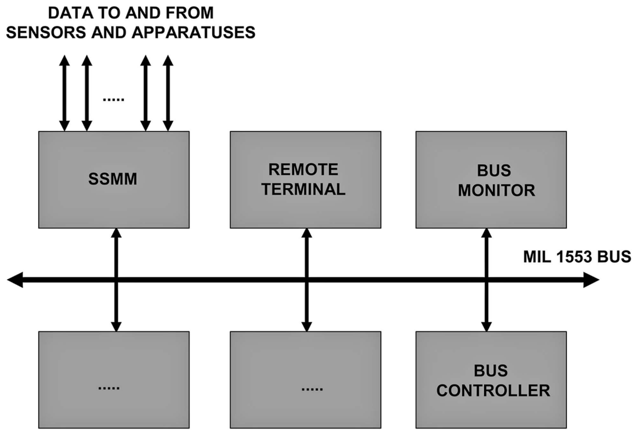


Fig. 1. External FTSSMM connections.

4) graceful degradation both in terms of functionalities and of availability of the service,
5) protection of the implementation of each function using different fault tolerant techniques.

Thus, a scalable architecture has been obtained by using the following techniques.

1) The error detection and correction (EDAC) codes used to protect the synchronized DRAM (SDRAM) chips are scalable RS codes; depending on the trade-off between reliability and latency requirements a different RS code can be used.
2) A different set of redundancy in the blocks can be used to obtain a trade-off between reliability versus power consumption and area overhead.
3) The use of a switching matrix to obtain dynamic routing capabilities allows to select the optimum number of I/O and memory modules to satisfy the throughput requirements.

Moreover, the use of dynamic reconfiguration algorithms allows a graceful degradation of the system. In fact, after the detection of an unrecoverable fault, the system can be modified to keep it working, even if its performances are reduced. The use of graceful degradation technique is suitable for the design of SSMMs for space applications, where the constraints on power and weight are quite relevant and the system can tolerate the presence of an out-of-order time when time-critical operations are not performed.

III. ARCHITECTURE DESCRIPTION

In this section, a detailed description of the FTSSMM architecture is presented. This architecture has been designed following the approach described in Section II.

At the top level, the FTSSMM can be viewed as a black box connected to different satellite apparatuses (Fig. 1). A number of bi-directional serial links are used for high-speed data exchange. For these links the

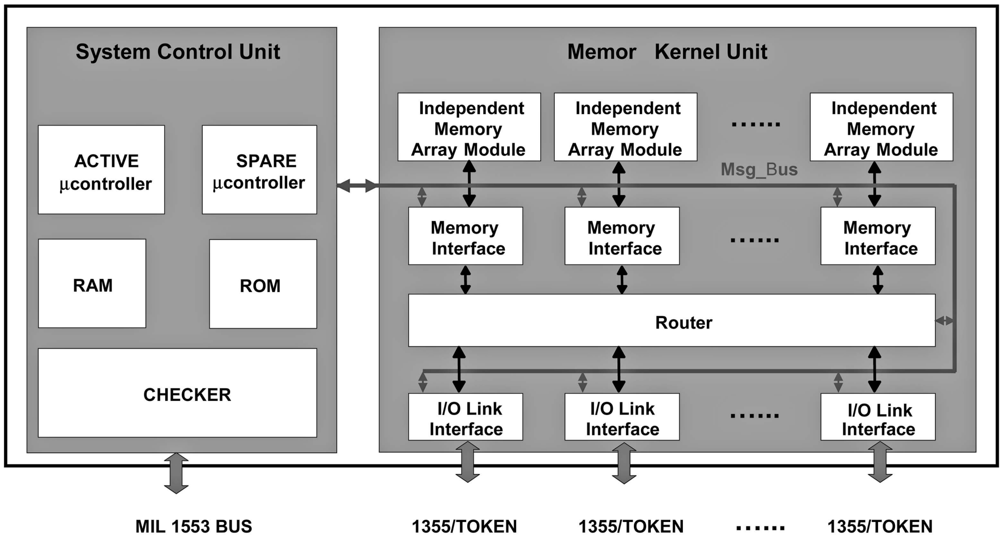


Fig. 2. FTSSMM architecture.

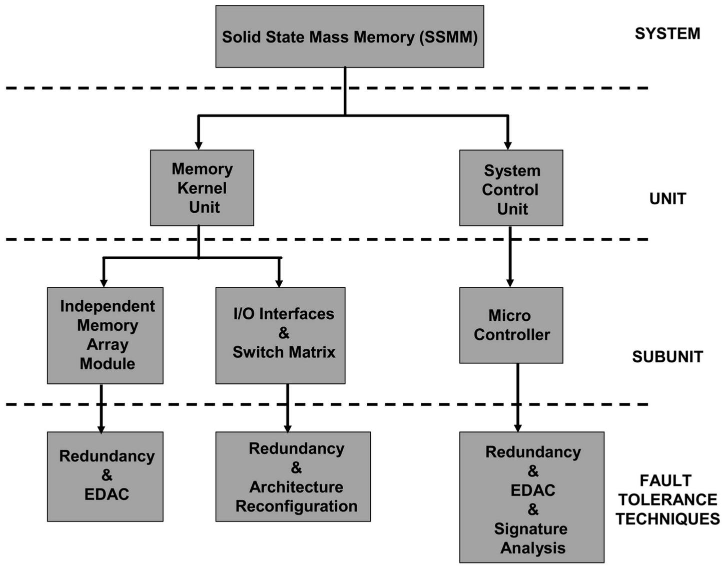

Solid State Mass Memory (SSMM)
SYSTEM
Memory Kernel Unit
System Control Unit
UNIT
Independent Memory Array Module
I/O Interfaces & Switch Matrix
Micro Controller
SUBUNIT
Redundancy & EDAC
Redundancy & Architecture Reconfiguration
Redundancy & EDAC & Signature Analysis
FAULT TOLERANCE TECHNIQUES

Fig. 3. Hierarchical view of FTSSMM and related fault tolerance techniques.

SpaceWire (IEEE 1355 DS-DE) protocol [9] has been chosen. In fact, SpaceWire is planned to become a European Space Agency (ESA) standard for on-board data handling in the near future and is expected to be widely used in future European missions [10, 5]. Each SpaceWire link is able to carry information (data or commands) at about 100 Mbit/s over distances of up to 10 m. Moreover, the FTSSMM can be connected to other apparatuses by a MIL 1553 interface, which is widely used in satellite platforms due to its physical redundancy (dual twisted pair bus structure) [11]. The FTSSMM is composed of the following two main units (Fig. 2).

1) The memory kernel unit (MKU) manages the bi-directional dataflow between users and memory chips

2) The system control unit (SCU) manages the memory resources and provides system level reconfiguration.

The required reliability of the FTSSMM system is achieved both by means of architectural redundancies and by introducing EDAC codes, granting the data integrity.

In Fig. 3, a hierarchical view of the FTSSMM is given. Each unit and the related subunits are indicated together with the adopted fault tolerant techniques. A simplified description of the FTSSMM operations can be drawn by analyzing the arrangement of the architecture shown in Fig. 4, where M data links (SpaceWire serial links) and N memory modules are connected by means of a crossbar switch matrix. The interfaces between the switch matrix and the N

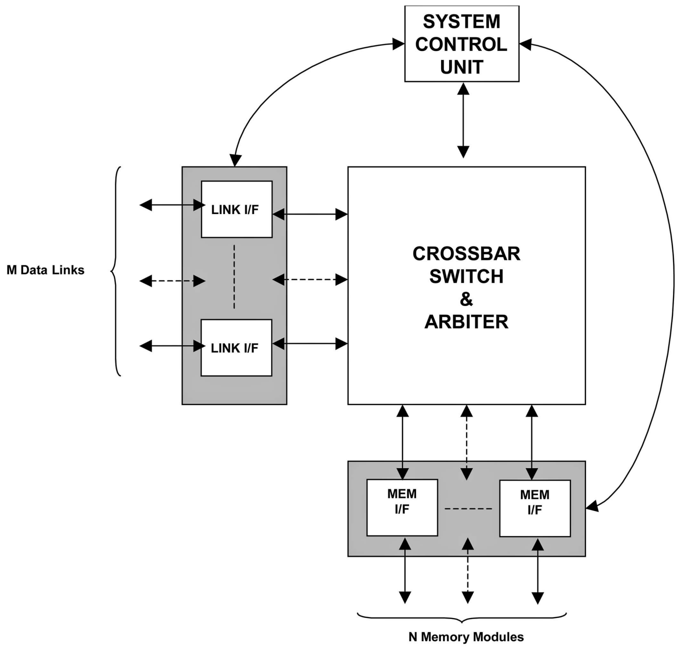


Fig. 4. Simplified FTSSMM architecture.

memory modules are called memory interfaces, while the interfaces toward the external M data links are called link interfaces.

All the interfaces and also the access arbiter, which handles the outputs contentions, are able to communicate with the SCU both for normal operations and for error handling. A system operation starts when a packet, that can contain either controls or data (see Fig. 5), arrives to the FTSSMM.

When it is a control packet (P(r)) that requires a read operation to a link interface it is forwarded to the microcontroller. The required memory modules are activated by using the commands (C_i(r)). As shown in Fig. 5, after a read operation request, the SCU generates C_1(r) in order to enable the first memory module. If the file is spread on different memory modules a chain of C_i(r) commands is generated by the memory modules itself. Instead, when a data packet P(w) arrives to a link interface (the arrival of a data packet means a write operation), the SCU is asked to allocate the destination module. The SCU answers to the link interface through the command C(w) to write its routing table. Thus the SCU can dynamically decide different file allocation strategies on different memory modules.

SCU also handles the error signals generated by the FTSSMM blocks. When an interface is affected by a fault, because of its self-checking implementation, the fault is detected and communicated to the SCU as shown in Fig. 6. The SCU can handle the error by using different policies depending on the faulty block and on the availability of spares. For instance a spare block can be used if available, thus implementing a typical duplex system, or a rerouting of data can be activated in order to obtain graceful degradation, if no cold spare is available.

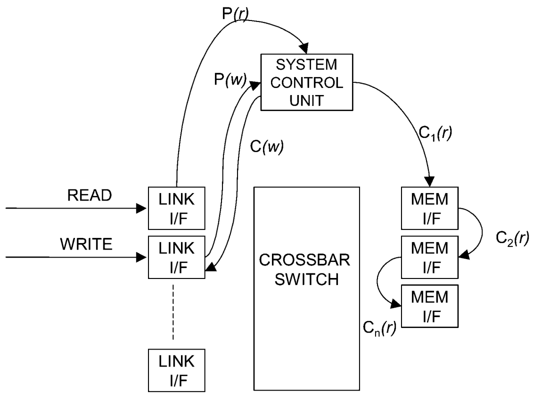


Fig. 5. Normal FTSSMM operations.

A. Memory Kernel Unit: General Description

As shown in Fig. 2 the MKU is composed of four functional modules:

1) independent memory array modules (IMAM) (see Section IIIA1),
2) routing module (see Section IIIA2),
3) I/O memory interfaces (see Section IIIA3),
4) I/O link interfaces (see Section IIIA4).

The MKU under the SCU control provides all the resources for the implementation of a file system on the set of SDRAM modules. The I/O interfaces are divided into two groups: I/O link interfaces and I/O memory interfaces. The I/O memory interfaces handle the IMAM file system, allowing basic operations like file read/write, delete,

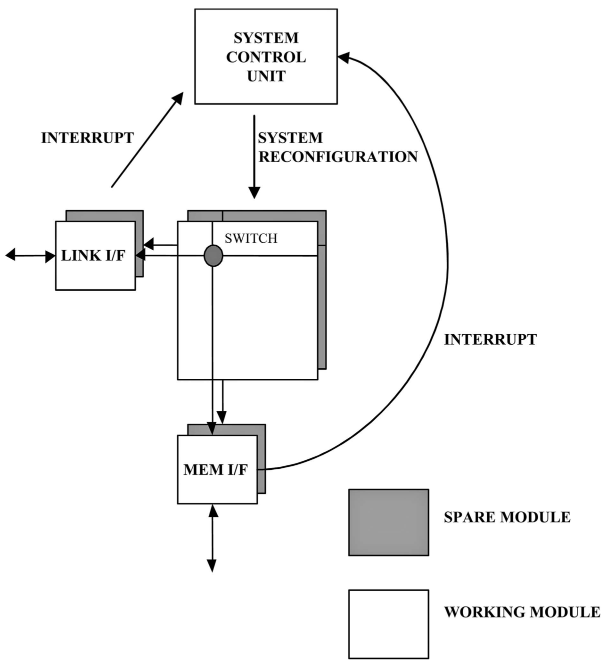


Fig. 6. FTSSMM reconfiguration.

format, etc. The I/O link interfaces are the front end of the system, providing a bi-directional transport of data and messages. The packet routing control and the dynamic reconfiguration of the system in case of faults are handled by exploiting the HW/SW interaction between these interfaces and the SCU. Once a connection between two interfaces is established, the data flow control is achieved through full handshake. The routing module is the central switch that interconnects the users (I/O link interface) with the memory modules. Each module has been developed by using different fault tolerant methodologies, depending on the final reliability requirements and the functionalities performed. These choices are described in the following subsections together with a detailed description of the modules architecture.

1) *Independent Memory Array Module (IMAM)*: The design of the memory array based on COTS RAM chips (dynamic random access memory (DRAM)) requires an accurate characterization of the used components in the environment in which they will operate. The effects of ionizing radiations on DRAM memory chips can be widely found in literature [3, 12, 13]. The usage of error-correcting codes strengthens the IMAM both in terms of reliability and data integrity.

Each IMAM module is composed of the following:

1) dynamic random access memory (SDRAM) bank (composed of several COTS chips or multi-chip modules (MCMs)),
2) control circuitry that interfaces the memory bank to the other components of the IMAM module,

TABLE I
Supported Codes

n,k	$n-k$
(18,16)	2
(36,32)	4
(72,64)	8
(144,128)	16

3) RS coder-decoder which adds redundancy to the data stored into the SDRAM.

The IMAM architecture is shown in Fig. 7. The SDRAM packages are arranged to implement either an RS code [14] with a maximum codeword length of 144 symbols or a code with a minimum codeword length of 18 symbols.

The most important features of the used RS codes are as follows.

1) small area due to the methodology proposed in [15] and "time- sharing" techniques that have been successfully applied to finite response filters [16],
2) optimization of the decode latency as illustrated in [17],
3) low cost reconfiguration: the codes can be reconfigured as RS(n,k), as shown in Table I, where k is the dataword length and n is the codeword length, i.e., $(n-k)$ redundant symbols are added to the original dataword.

The ratio k/n, for all the used codes, is 0.89, i.e., the check byte overhead is constant with respect to the selected code.

RS code reconfiguration is used when a permanent failure occurs in a memory package. For example, if the FTSSMM is initially configured with an RS(18,16) code and a permanent failure occurs in a memory package, the code can't correct any random error. Therefore we need to reconfigure the memory module with a new RS code. Permanent package failures can be easily detected by applying a reading and decoding procedure. The decoded (and corrected) data will be coded again with an RS code with higher correction capabilities. To perform this operation, a suitable buffer temporarily stores the codewords that will be grouped to form the larger one. Obviously, the use of a higher RS code involves the use of a higher number of symbols. On the other hand the fixed ratio k/n allows the use of higher RS code without adding symbols overhead.

As an example we show the reconfiguration from the RS(36,32) to the RS(144,128). Starting from an RS(36,32) coding scheme, after a certain period of time the check procedure detects three permanent package failures (three erasures). All the data stored in the memory module are converted from RS(36,32) to RS(144,128): i.e., four 36 byte codewords are read and decoded and the 128 data bytes are coded into a codeword of 144 bytes.

This procedure allows preserving the data stored in the memory. However, if the number of erasures

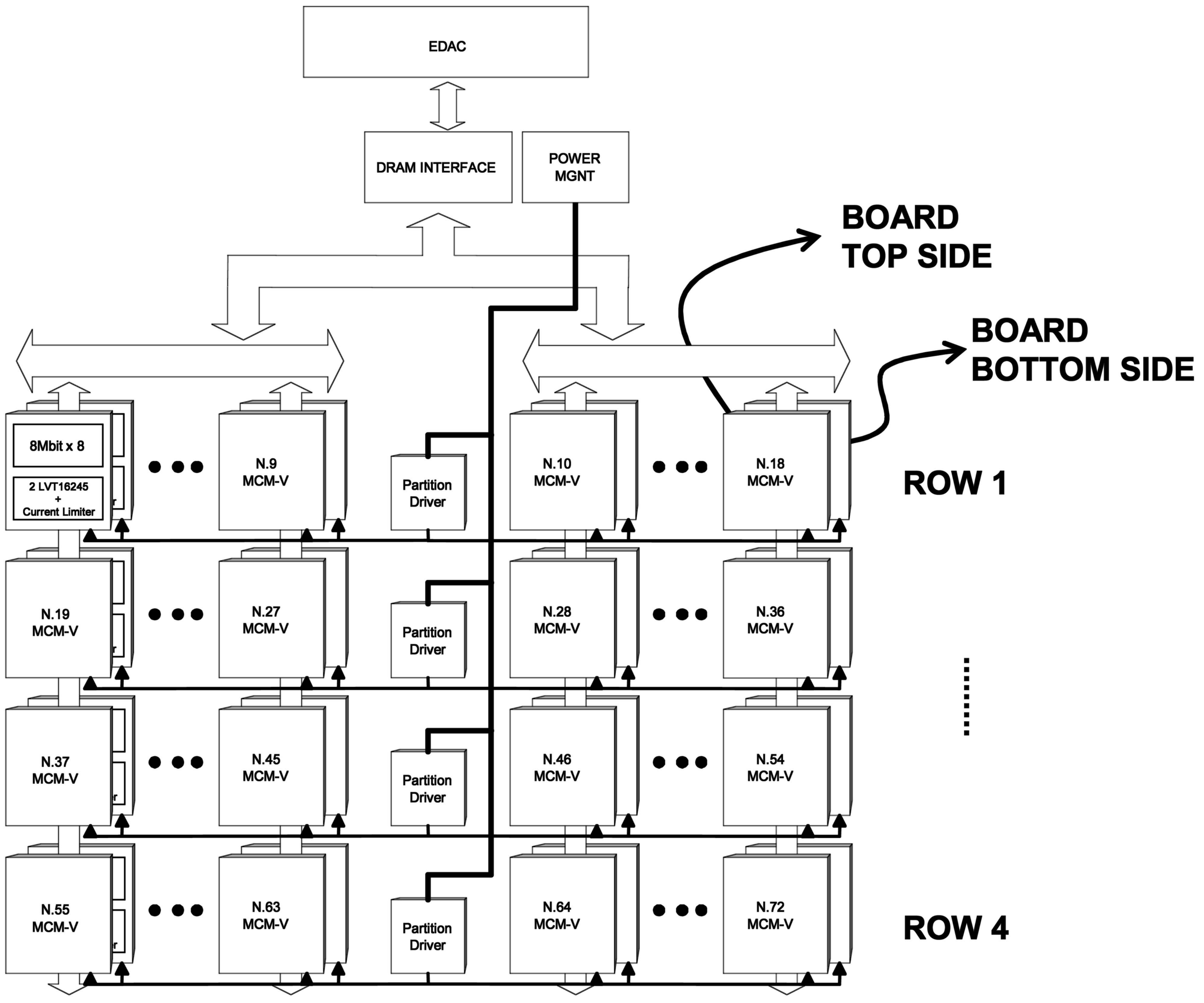


Fig. 7. Memory array architecture.

is greater than the error correction capability of the active code, (e.g. 5 erasures for the RS(36,32) code), the data stored in the codeword are unrecoverable, but the functionality of the memory element can be restored using a code with greater error correction capability. The use of longer codewords improves the integrity of the stored data degrading the performance of the IMAM in terms of latency. In fact, the decode latency depends on the codeword length.

2) *Routing Module*: The routing system connects the I/O memory interfaces with the I/O link interfaces through a crossbar switch matrix. The interconnection is performed in nonblocking mode. An arbiter provides the acknowledge signals to the I/O interfaces that send data through the crossbar. With this interconnection method multiple parallel connections between users and resources can be established, by increasing the overall throughput. Latencies can be reduced choosing appropriate arbitrating policies. Moreover, the intrinsic redundancy of such architecture increases the reliability of the system. The failure of a connection, due to a fault in an I/O interface or in a switch, implies only a partial loss of the system functionality.

a) *Crossbar switch matrix*: This component allows the physical interconnection among α I/O link interfaces and β I/O memory interfaces (see Fig. 2). The number of the possible connections, and thus the number of switches and wires necessary in a crossbar switch matrix, can be defined by introducing an $(n \times n)$ interconnection matrix where $n = \alpha + \beta$. In this matrix the (i, j) element is equal to 1 if there is a connection between the input i and the output j, 0 if there is no connection. In our implementation we consider two subsets of the set N of I/O interfaces: A and B of α and β elements, respectively, being $\alpha + \beta = n$ $A \cup B = N$ and $A \cap B = \emptyset$. Connections are only present between elements belonging to different subsets, requiring $2 \cdot \alpha \cdot \beta$ switches and $2n$ wires.

b) *Arbiters*: The arbiters added to the routing matrix, handle the interconnections conflicts between the I/O interfaces (I/O link interfaces and I/O memory

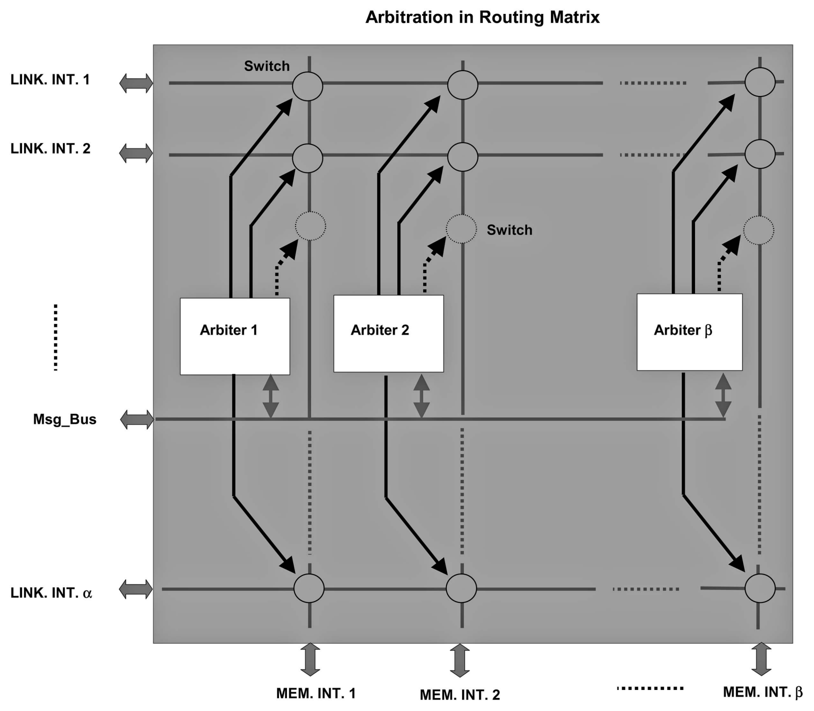


Fig. 8. Distributed arbiter.

interfaces). They are able to manage write conflicts (different I/O link interfaces trying to write on the same IMAM) and read conflicts (different memory interfaces trying to access to the same I/O link interface during file read operations). In Fig. 8 the switch matrix functional architecture is shown. For the sake of simplicity only write arbiters have been indicated. The use of an arbiter for each output (write and read mode) avoid setting different arbitrating policies for each output (write and read mode).

The available arbitrating policies are priority based or time sharing. For instance, a priority-based policy can be used during the window of visibility of a Low Earth Orbit (LEO) satellite, when the I/O memory interfaces that are downloading the data to the Earth station should be assigned more bandwidth. Instead, with a time-sharing policy the bandwidth is almost equally shared between all the interfaces requesting the same output. The different arbitrating policies are programmed by the SCU through the Msg_Bus.

3) *I/O Memory Interfaces*: These interfaces handle the file system. Each I/O memory interface has a local file allocation table (FAT) stored in the controlled memory module (IMAM). The partition of the file system in the different memory modules reduces the amount of data that can be lost in case of a FAT unrecoverable failure. In fact, in this case only the local stored information will be lost.

The internal architecture of the I/O memory interface, is composed by a number of subblocks as shown in Fig. 9.

The file system handling functions, implemented by dedicated blocks are the following.

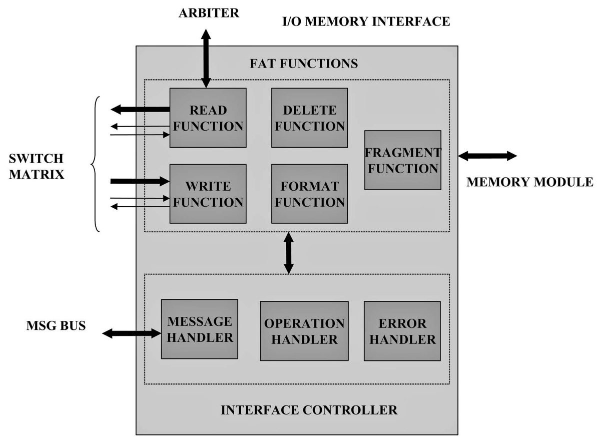


Fig. 9. I/O memory interface.

1) Delete function: used to delete a file from the FAT.

2) Fragment function: used to add to the FAT the occurrence of more fragments of the same file.

3) Read function: used to read a file from the memory.

4) Write function: used to write a file in the memory.

5) Format function: used to setup the FAT in the initialization phase.

Each function is implemented with a separate block and the "operation handler" (interface controller subblock) activates the different functions. Moreover, each block is self-checking and a spare block is used to obtain the fault tolerance. In fact, when the occurrence of a failure (single block failures) is

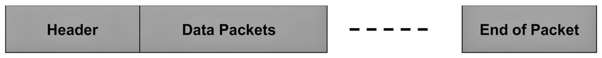


Fig. 10. SpaceWire packets.

detected the "error handler" activates the spare module with a low time overhead.

Both the operation handler and the error handler communicate with the rest of the FTSSMM through the message handler. Therefore, through the message bus, the SCU is able to control the status of each I/O memory interface both in the case of normal operations or in the case of the occurrence of a fault. In order to obtain single point of failure avoidance, the interface controller has been implemented by using the TMR technique.

4) *I/O Link Interfaces*: The I/O link interfaces allow the exchange of data (through the routing matrix) from the input channels to the memory modules and commands from the I/O link interfaces, to the SCU.

The packets organization for the SpaceWire interface is shown in Fig. 10.

The one byte header is the packet ID while the payload is composed of a variable number of bytes terminated by the end of packet (EOP) marker. We assume that the header values in the range 1 to 255 indicate that the packet is part of a file whose ID number is the header value. Header value 0 indicates a special packet containing commands to the SCU sent through the Msg_Bus. Thus the file system can handle 255 files and the memory can be controlled and monitored by using the same links carrying the data.

In the FTSSMM, most of the I/O link interfaces are used in unidirectional mode (links carrying measurement data from the on-board instrumentation). Just a small number of I/O link interfaces require reading and writing of the memories (connection to the satellite CPU or its telemetry).

An internal shared bus interconnects all the I/O interfaces, the SCU, and the routing module to provide file system management and error detection (Msg_Bus).

B. System Control Unit

The SCU (Fig. 11), manages the high level tasks of the FTSSM. This module is connected to the MKU through the Msg_Bus. The main functions of the SCU are closely related to the high level file system operations and can be summarized as follows.

1) Create file request (from I/O link): when a packet must be written in a file that does not exist, the file must be created. The file create operation is performed by the SCU, which has the control of the FAT.

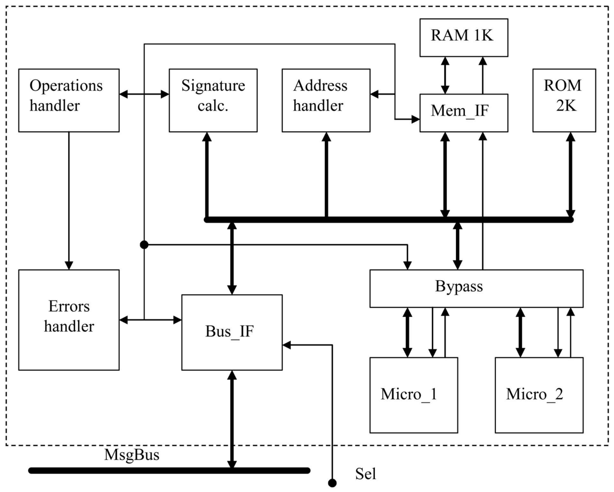


Fig. 11. System control unit.

2) Command execution request (from I/O link): the packets with header 0 represent the command for the FTSSSM. All these commands (read file request, erase file request, and diagnosis and debug commands) are managed by the SCU.

3) Page allocation (from IMAM): when the page currently used by the IMAM is full, the IMAM sends to the SCU a request for a new empty page.

The Bus_IF interface handles the exchange of messages between the controller and the rest of the FTSSMM system through the Msg_Bus. The system uses two microcontrollers Intel 8051 that can be connected or isolated from the system through the block bypass. Normally only a single processor is active and connected to the system, while the other one is in stand-by and electrically isolated. The active microcontroller accesses a 2k ROM program memory and a 1k RAM data memory.

The Mem_IF block supplies the coding of the data the microcontroller writes in the RAM. The data read from the memory is decoded from the same block. The operation performed by the Mem_IF is transparent to the microcontroller. A Hamming code capable of correcting single errors has been used.

The "address handler" block handles the connection between the microcontroller and the Mem_IF and Bus_IF creating the suitable switches, transparently to the microcontroller, to achieve the required connections.

The "signature computation" block checks the correctness of the operations performed by the microcontroller. This block reads the sequence of the addresses at the output of the microcontroller and checks if they are following a correct sequence. The evaluation of the correct execution is performed by using signature analysis [18, 19].

The operations handler block manages the phases of the elaboration of a message: if some phases are not executed correctly, this system supplies the relative error signaling.

TABLE II
Reliability Evaluation of FTSSMM

	RS(36,32)	RS(72,64)
ROUTER + SCU RELIABILITY @ 2 years (Cold redundancy—1 out 2)	0.9996 (23 FIT @ 2 y)	0.9996 (23 FIT @ 2 y)
IMAM RELIABILITY @ 2 years (R_{MM})	0.9984 (91 FIT @ 2 y)	0.9984 (91 FIT @ 2 y)
—MEMORY CONTROL/EDAC/INTERFACE @ 2 years	0.99845 (89 FIT @ 2 y)	0.9986 (80 FIT @ 2 y)
—MEMORY ARRAY RELIABILITY @ 2 years	0.99995 (3 FIT @ 2 y)	0.9998 (12 FIT @ 2 y)
MEMORY STACK RELIABILITY @ 2 years (Cold redundancy—5 out 6)	0.99997 (1.7 FIT @ 2 y)	0.99997 (1.7 FIT @ 2 y)
FTSSMM MEMORY RELIABILITY @ 2 years	0.9996 (23 FIT @ 2 y)	0.9996 (23 FIT @ 2 y)

The error handler block receives error signals from all the blocks of the system and handles the error exceptions. It operates in a transparent way and assumes the control of the system only if an error is found. The error management is aimed to mask the system faults. If the error persists the manager can substitute the active microcontroller with the spare one.

IV. RELIABILITY, DATA INTEGRITY, AND GRACEFUL DEGRADATION EVALUATIONS

In this section some evaluations of reliability, data integrity, and graceful degradation capabilities of the FTSSMM are shown. The IMAM dominates the complexity of the system. In fact, to obtain a storage capability of several Gigabytes, a high number of SDRAM chips must be used. Therefore, the reliability of this subsystem must be accurately studied. The use of RS codes to grant a high level of data integrity allows also increasing the reliability of the IMAM. In fact, the erroneous data of a failed SDRAM chip can be faced as particular data errors, called erasure, and corrected by the RS codes. This improvement of the reliability depends on the codeword length and on the memory scrubbing frequency. Moreover, the choice of RS codes with a number of bit per symbol of 8 is useful also in case of multiple bit upset (MBU) with a limited number of corrupted bits. In fact, an MBU of up to 8 bits corrupt one or two symbols, and due to the ability of RS code to correct symbols instead of bits, these kinds of faults can be easily managed by RS codes with a sufficient number of check symbols. Finally, the modularity of the FTSSMM allows to face the occurrence of unrecoverable faults preserving a reduced functionality set and/or reducing the amount of available memory storage capability.

A. Reliability Evaluation

The allocation of reliability of a system involves solving the following inequality

$$f(R_1,R_2,\ldots,R_n) > R^*$$

where R_i is the allocation reliability parameter for the ith subsystem, R^* is the system reliability requirement parameter, and $f()$ is the functional relationship between subsystem and system reliability.

For a simple series of systems, in which the Rs represent the probability of survival at end of life (EOL), we get

$$R_1R_2\ldots R_n > R^*.$$

The above equation has an infinite number of solutions and a procedure that yields a unique or limited number of solutions must be used. For this purpose we use a proprietary optimization tool [20] based on the minimization of an effort function, as described in [21]. For the reliability evaluation of the FTSSMM a suitable reliability model is used [22]. It must be noticed that theoretically the results of the reliability evaluations depend on the chosen RS code, because different codes correspond to different reliability models (see [22] for a detailed description). In Table II the results of this evaluation are reported, for both the subsystem and the overall FTSSMM with two RS code configurations.

The above results show that the architecture has a reliability level matching the requirements of a 2 year long satellite mission. It has to be noticed that the overall results are related to the hypothesis of full functionality of the FTSSMM after 2 years. However, the graceful degradation capability of our system allows different levels of acceptable performances, i.e., the FTSSMM can be reconfigured to work also with less I/O link interfaces and/or memory modules. Therefore the above evaluations can be considered as a worst case analysis of the system, while the probability that the FTSSMM keeps working at the end of mission with reduced performances is higher as explained in Section IVC. Analyzing the results in detail, we can notice that the reliability of the memory control for the RS(36,32) configuration is better than RS(72,64), however the latter can tolerate a larger number of memory package failures. Therefore, the overall reliability of the RS(72,64) configuration is almost the same of the other case. The modification of RS coding scheme improves the data integrity as we show in the next subsection.

B. Data Integrity Evaluation

To evaluate the data integrity of the system we use the notation adopted in [23] and [24] and reported in Table III.

TABLE III
Notation

λ	Upset bit rate
N_{error}	Number of corrupted bits
m	Number of bits per symbol
$N_{\text{total},W}$	Memory size in number of bits and words
k	Number of symbol per data word
MTTDL	Mean Time To Data Loss
MTTF	Mean Time To Failure
c	Number of check symbols in a codeword
$r(\tau)$	Probability @ time τ of codeword error free
n	Number of symbol per code word
$R(\tau)$	Probability @ time τ of memory system error free
t	Correctable random error
T	Interval of scrubbing with deterministic mechanism
LT	Latency Time of stored data
j	Number of intervals of deterministic scrubbing

TABLE IV
BER at Storage Period of 48 Hours and $T_{\text{scrubbing}} = 1000$ s

Reed-Solomon	Error	BER @ Minimum SEU Rate	BER @ Maximum SEU Rate
RS(36,32)	2 re, 0 er	$1.7 \cdot 10^{-20}$	$1.7 \cdot 10^{-14}$
RS(72,64)	4 re, 0 er	$2.5 \cdot 10^{-33}$	$2.5 \cdot 10^{-23}$

We assume the following.

1) Transient faults occur with a Poisson distribution.
2) Bit failures are statistically independent and thus linearly uncorrelated.
3) The control, correction, and interface circuitry in the memory system are fault tolerant.
4) There is a dominant memory bit cell failure mode. This with assumption 3 provides an upper bound on system reliability.

In interplanetary space a background rate of $7.3 \cdot 10^{-7}$ errors/bit/day can be assumed, which occasionally increases up to $1.7 \cdot 10^{-5}$ errors/bit/day during solar flares. The following tables summarize the data integrity evaluation in terms of bit error rate (BER) from the minimum SEU rate to the maximum SEU rate, even if some package failures occur.

Different situations in which the FTSSMM can operate are summarized in Table II. Table IV shows the data integrity evaluations computed assuming the following hypotheses.

1) The data are downloaded from the satellite to the Earth station every two days.
2) A scrubbing operation is performed with a period of 1000 s. The scrubbing operation is performed reading a data stored in the memory and rewriting it in the same memory location. This operation allows to correct errors that can occur in the data word, preventing the situation in which different errors accumulate in the same word leading to an unrecoverable erroneous word.

TABLE V
$T_{\text{scrubbing}}$@ $T_{\text{storage}} = 2$ day and BER $= 10^{-12}$
(0 Memory Package Failure)

Reed-Solomon	Error	$T_{\text{scrubbing}}$ @ Minimum SEU Rate	$T_{\text{scrubbing}}$ @ Maximum SEU Rate
RS(36,32)	2 re, 0 er	> 2 days	$7.67 \cdot 10^{3}$ s (~ 2 h) < 2 days
RS(72,64)	4 re, 0 er	> 2 days	> 2 days

TABLE VI
$T_{\text{scrubbing}}$@ $T_{\text{storage}} = 2$ day and BER $= 10^{-12}$
(2 Memory Package Failure)

Reed-Solomon	Error	$T_{\text{scrubbing}}$ @ Minimum SEU Rate	$T_{\text{scrubbing}}$ @ Maximum SEU Rate
RS(36,32)	1 re, 2 er	$6 \cdot 10^{3}$ s (1.8 h < 2 days)	$6 \cdot 10^{-1}$ s ($\ll 2$ days)
RS(72,64)	3 re, 2 er	> 2 days	$8 \cdot 10^{4}$ s (~ 1.8 h < 2 days)

These results show that the FTSSMM is able to tolerate the occurrence of transient faults giving very high data integrity levels. Table V shows the frequency of the scrubbing operation needed to achieve a requested BER level of 10^{-12}. The table shows that the scrubbing operation is needed only for high SEU rates and for an RS(36,32) code. In the other cases the scrubbing operation is not necessary, and a reduction in terms of power consumption can be obtained. Finally, in Table VI, the BER levels in the presence of permanent faults in two memory packages are reported. The table shows that, also in presence of permanent faults, the FTSSMM remain able to provide high reliability levels. It can be noticed that, also in presence of permanent faults, there are some combinations of SEU rate and coding scheme in which the scrubbing technique is not necessary. The above reported results show that the FTSSMM is able to tolerate a high number of permanent and transient faults occurring in the IMAM exploiting the RS coding reconfiguration. The reconfigurability of the RS code allows, given an expected SEU rate and a BER requirement, to operate both on the codeword length and/or the scrubbing period to obtain the requested memory performances. The choice of the codeword length and of the scrubbing period is the result of a trade-off. In fact the use of long codewords increases the time for decoding the data-word, while the use of short scrubbing periods increases the time in which the memory can't be accessed by the user.

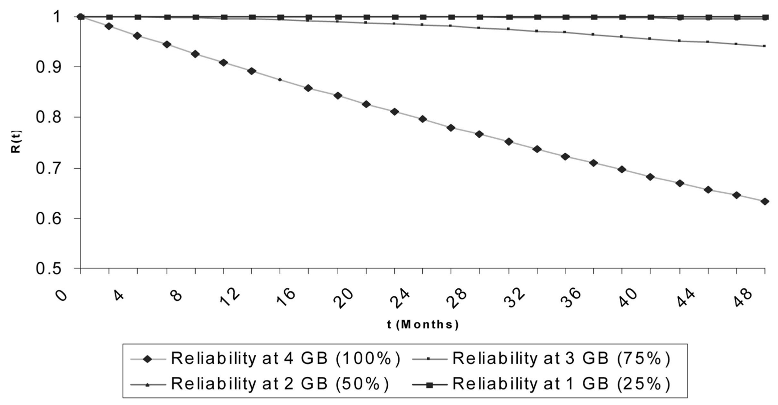


Fig. 12. Reliability curves for different performances.

TABLE VII
Characteristic of Set A

	Write	Read	Delete
Detection Technique	CED	Signature analysis	CED
Redundancy	None	None	None
# CLB	1015	390	265

TABLE VIII
Characteristic of Set B

	Write	Read	Delete
Detection Technique	CED	—	CED
Redundancy	Cold Spare	TMR	Cold Spare
# CLB	1630	803	445

C. Graceful Degradation Capabilities Evaluation

The graceful degradation capabilities of the system [25] can be evaluated by calculating the probability that the system works correctly at different levels of performance. This approach is quite similar to the performability defined in [26]. In this section we focus our attention on the IMAM block, because the reliability of this block is dominant with respect to the other blocks (see Table II). As an example, let us report two possible sets of fault tolerant techniques that can be applied to the file system functions. In Table VII (set A) concurrent error detection (CED) techniques are used to check the write and delete functions reducing the latency of the transient and permanent fault detection that could lead to irreparable FAT incongruities. This set of solutions has a limited area overhead because no spares are introduced. The drawback of this choice is that when a permanent fault is detected, the function can't be recovered and the performance of the mass memory must be degraded.

In Table VIII (set B) the concurrent detection technique is applied to both write and delete functions while TMR is applied to read function to provide better reliability and lower read latency.

Lower read latency is a requirement for a LEO satellite. In fact, the short window of visibility requires the implementation of fast downloads with no repetition. This set of solutions has a higher area overhead with respect to the previous one. On the other hand, when a permanent fault is detected, the functionality can be recovered using the redundant blocks and the performances of the mass memory are not degraded. Depending on the selected set of solutions different levels of performance degradation can be obtained. In fact, for a field programmable gate array (FPGA) implementation of the various file system functional blocks the reliability of each of these blocks can be estimated as follows. Assume that the reliability of each block is the reliability of the series of the complex logic blocks (CLBs) needed to implement the function. Therefore, given a certain failure rate λ of a single CLB, the failure rate of the functions can be expressed as

$$\lambda_{\mathrm{W}} = 652 * \lambda, \qquad \lambda_{\mathrm{R}} = 260 * \lambda, \qquad \lambda_{\mathrm{D}} = 178 * \lambda$$

where λ_{W}, λ_{R}, λ_{D} are the failure rates of the write, read, and delete blocks, respectively. In a configuration with only fault detection capability (set A), inside the memory interface, the reliability can be assumed as the reliability of the series of the function blocks. The failure rate λ_{MI} of a single memory interface is

$$\lambda_{\mathrm{MI}} = \lambda_{\mathrm{W}} + \lambda_{\mathrm{R}} + \lambda_{\mathrm{D}}.$$

Using this failure rate can estimate the reliability of the overall set of memory interfaces at different levels of performance for a 4 Gigabytes FTSSMM composed of four 1 Gigabyte memory modules. The possible levels of performance are defined as the amount of memory available. With n interfaces, the reliability with r interfaces functioning is

$$R_s = \sum_{i=r}^{n} \binom{n}{i} [R(t)]^i [1 - R(t)]^{n-i}$$

where $R(t)$ is the reliability of a single interface. If an unrecoverable fault is detected, the FTSSMM is reconfigured reducing the available storing capability. In Fig. 12 the reliability curves are drawn. It can be noticed that the reliability to a typical end of mission

TABLE IX
Reliability Comparison at EOM

$R(t)$ $t = 3$ years	4 GB	3 GB	2 GB	1 GB
Set A	0,7097	0,9638	0,9979	0,9999
Set B	0,9893	0,9999	0,9999	0,9999

(EOM) of 36 months is about 70% if complete availability is required (4 GB), while, in the cases of 75%, 50%, and 25% storage capability the reliabilities are above 95%. The use of redundancy inside the memory interface (set B) provides better reliability for each memory interface.

In fact, with this configuration and the same failure rates for each hardware block (λ_W, λ_R, λ_D) the formulas given in [27] can be applied to calculate the reliabilities of a TMR configuration and cold spare configuration. Using the obtained reliabilities of the hardware blocks the reliability curves of Fig. 12 can be recalculated. In Table IX is shown the improvement of the reliability for each performance configuration at the EOM.

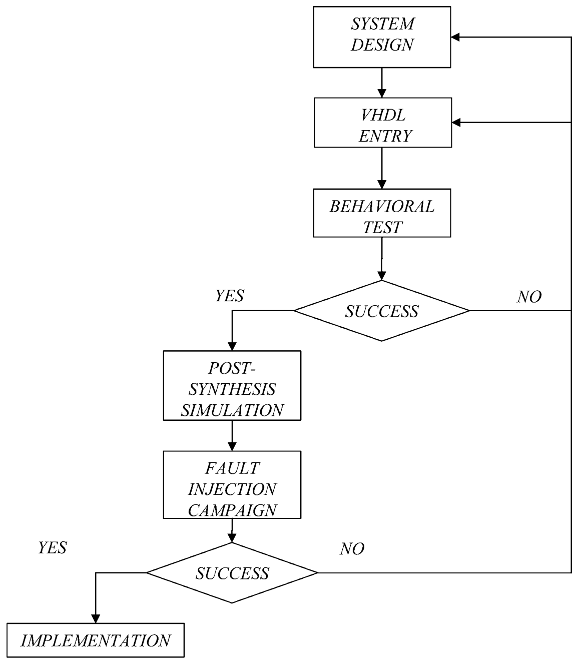


Fig. 13. FTSSMM design flow.

V. FTSSMM SIMULATIONS

The simulation of the FTSSMM has been performed with the purpose of evaluating both its performances and its fault tolerant capabilities. In particular, during the design phase the simulations have been carried out using the VHDL hardware description language. To have a closer emulation of real physical faults, we injected faults in postsynthesis structural VHDL [28, 29]. The results obtained by behavioral and postsynthesis with fault injection simulations of the system provided a feedback in the design flow (Fig. 13).

Different from a typical design flow, the verification phase of fault tolerant systems requires not only the verification of the correctness of normal functionalities, but also of the functionalities related to the operations which must be performed to face the occurrence of a fault in the system.

The simulation results are mandatory to confirm the fault tolerance and graceful degradation capabilities of the FTSSMM. The use of VHDL descriptions of the FTSSMM (both structural and behavioral) allowed us to perform a fault injection campaign using a method called the "saboteurs" method [30].

The validation of the FTSSMM has been performed by splitting the system into the two main blocks MKU and SCU. In both subsystems transient and permanent faults have been simulated. Regarding the validation of the MKU we focused our attention on the occurrence of permanent faults in a memory module or in an I/O memory interface in order to validate the graceful degradation capabilities of the system. In fact, to face the occurrence of permanent faults the gracefully degradable MKU must switch the data path from the faulty memory module to another one. Due to the system symmetry, the same reconfiguration can be done also if a failure occurs in an I/O link interface. This last feature exploits the capability of the system to read the data stored in the memories from any I/O link interface. The reconfiguration of the routing allows a graceful degradation of the system in terms of throughput but keeps intact the basic functionality of the storage system. In fact, the shared transfer of the files on the same module is more time consuming than the time needed for parallel transfer to different modules, but we obtained the result that the system can still store and read files from at least a memory module. The same consideration can be done also if some link interfaces fail.

For the validation of the SCU it must be noticed that the fault simulation campaign implies a specific approach with respect to the fault simulation in the MKU block. In fact, as in the final release of the FTSSMM where two actual microcontrollers will be used, a prediction of the error rates and of the behavior of the microcontroller in case of faults must be performed.

The fault simulation has been done on a behavioral VHDL description of the 8051 microcontroller, and the correctness of the results has been validated comparing the obtained data with the ones obtained by a radiation ground testing campaign performed running two programs used as test bench applications [31]. Radiation testing experiments, in which the 80C51 processor was exposed to beams of several heavy-ion species, were performed by the Cyclone

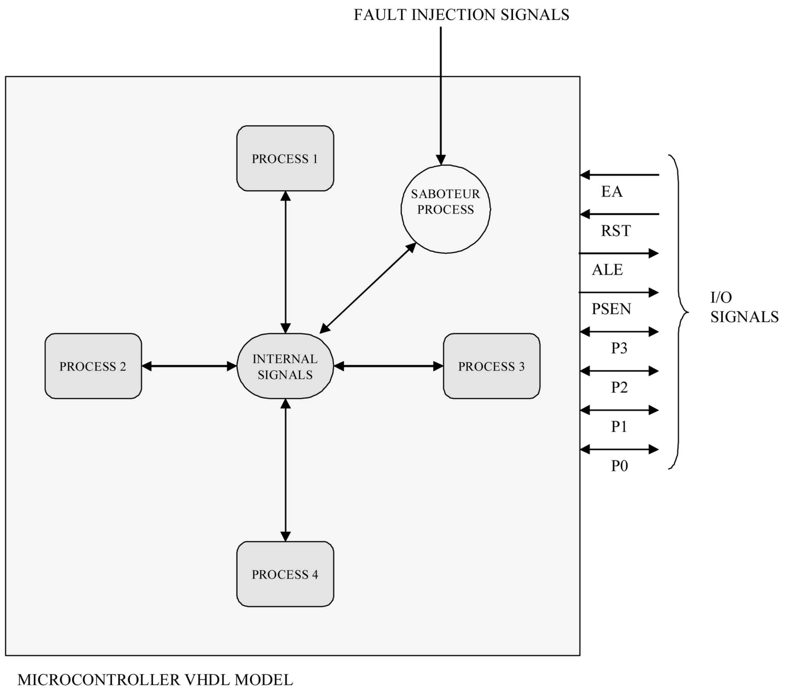


Fig. 14. "Saboteur" process.

cyclotron available at the UCL (Université Catholique de Louvain-la-Neuve, Belgium). Further details about the Cyclone facility and the main characteristics of the heavy ions, to which the studied processor was exposed are provided in [32]. These experiments of ground testing allowed measuring the static SEU cross section of the 8051 microcontroller by executing a memory-like test pattern (static strategy) while exposing the circuit to the heavy ion beams. Such an experiment provides statistical evaluations of the number of particles needed to flip a bit of a given memory element. The resulting cross section curve gives for each of the used particle species (identified by the energy they deposit in silicon measured by linear energy transfer (LET)) the number of detected bit flips normalized by the number of hitting particles

$$\sigma_{\mathrm{SEU}} = \text{number detected errors/particle fluency}. \quad (1)$$

The considered faults are limited to changes in the content of internal memory elements. The VHDL model uses an array of 8 bit vectors in order to simulate all the 128 internal RAM bytes and special function registers (SFRs) included in the 8051 architecture. A test campaign has been performed in which faults were randomly injected both in location of the affected bits inside this array and in time of its occurrence. Note that injected faults did not target the memory bits of program code to be executed by the microcontroller, the fault injection being performed by means of suitable modifications added to the VHDL signals within the emulated 8051.

The setup of the experiment was therefore a simulation setup. In fact, we generated a VHDL schematic with the instantiation of the studied microcontroller and other needed blocks. Main blocks of the setup were
8051,
SRAM 64k,
ROM 4k.

These blocks were simulated using a commercial VHDL simulator [33]. The only modification made to the VHDL model was to add a saboteur process capable of injecting faults inside the registers within the 8051. The VHDL simulator, concurrent with the normal processes emulating the 8051 microcontroller, executes the saboteur process. Therefore the activation of fault injection, performed by means of the two extra signals IND and BIT, is totally independent and asynchronous to the state of the 8051. Fig. 14 shows a schematic description of the fault injection strategy in a general case. The VHDL behavioral description of the 8051 can be seen as a set of concurrent processes, each one implementing a function of the microcontroller (ALU, PC incrementer, Watchdog, etc.) and the communication between these processes is provided by a set of internal signals visible to all the processes. Some of these signals have physical meanings like SFR, RAM, PC, and others. The saboteur is a special process that runs concurrently to the other processes and is activated, in our case, by two external commands IND and BIT. When normal operation (without fault injection) is carried out the saboteur is in a stand-by mode; when the fault injection is activated, the saboteur modifies an internal signal inverting its value. In this way

TABLE X
Results of Fault Injection Campaigns

	# Runs	# Result Errors	# Detected Errors	# Lost of sequence	# Detected Lost of Sequence	Result Errors coverage	Lost of Sequence coverage
First	40000	6078	3110	4969	4908	51,16%	98,77%
Second	40000	5948	3066	4864	4797	51,54%	98,62%
Third	40000	5692	2964	4785	4746	52,07%	99,18%
Average	40000	5912,67	3046,67	4872,67	4817	51,53%	98,86%

the saboteur provides an asynchronous SEU injection on any internal signal of the VHDL description.

It has to be noticed that, in order to have a realistic behavior of the microcontroller, the injection must be performed only on those signals representing physical registers. Assuming that the SEUs mainly affect the internal registers rather than combinatorial logic, we isolated the signals representing these registers and made them our SEU injection target.

The fault injection technique is composed of the following steps. First, the time width of injection zone relative to the program that must be tested is defined, second, the number of logical targets is defined that must be used in order to inject error in all the internal registers (in this case 152). Once both these ranges are determined, the macro routine is generated for executing a test using a suitable C++ program. The C++ program is based on a recursive algorithm, the first step of each cycle is the generation of time variable for error injection (into the predefine injection zone interval time), the second step is the generation of logical targets of injection, both byte and bit addresses. Finally the C++ program writes the correct macro file commands for fault injection.

Aiming at comparing the validity of prediction based on fault injection sessions radiation ground testing was performed with the UCL cyclotron. In Fig. 15 curves corresponding to both the measured and predicted error rates are depicted. The comparison of predicted and measured error rate curves, put in evidence the excellent correlation obtained from the prediction technique based on fault injection. These results show clearly the excellent efficiency of this technique to predict error rates, at least for the simple studied processor (the 8051).

Starting from the validation of the fault injection methodology, a set of simulations of the SCU behavior in the presence of fault has been done [34]. The simulation has been performed to validate the self-checking capabilities of the SCU provided by the signature computation and the operations handler blocks. We performed a large number of fault injection runs in three campaigns and the results are reported in Table X.

We focused our attention on the lost of sequence detection. We expected that the checker would have a high coverage on these errors that affect the control path of the microcontroller and the results provided

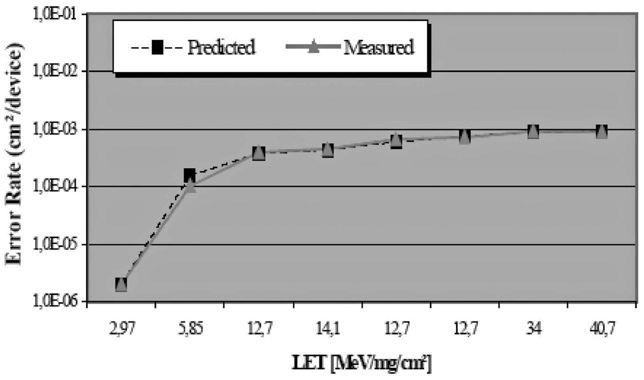


Fig. 15. Predicted and measured error rates under different heavy ions for 80C51.

an average of 98.86% of coverage. We assumed that a loss of sequence occurred; if the program did not terminate in a time 150% of the normal execution time then we checked if the signature analysis checker detected the error. The high coverage allows protecting the microcontroller from freezes that would isolate it from the rest of the system. We also checked the correlation between the control flow errors detected by the checker and the origination of errors in the results. For the vector-sorting algorithm we found a 50% correlation. Even if this correlation could improve the quality of the results, it is worth noticing that the main result is the high coverage on the loss of sequences.

VI. PROTOTYPE SETUP

The development of the prototype of the FTSSMM was intended to obtain a simple but still representative version of the design. The FTSSMM prototype is based on two memory modules and two link interfaces. This kind of setup permits testing of all the features of the FTSSMM while reducing the complexity of the implementation. Moreover, the use of a fast prototyping methodology based on reprogrammable FPGAs, allows an incremental testing approach. The functional structure of the prototype is shown in Fig. 2. It has been partitioned and mapped on the hardware as specified in Table XI.

Each part of the hardware apparatuses shown in the table are described in the following sections. The prototype has been tested by realizing two emulators

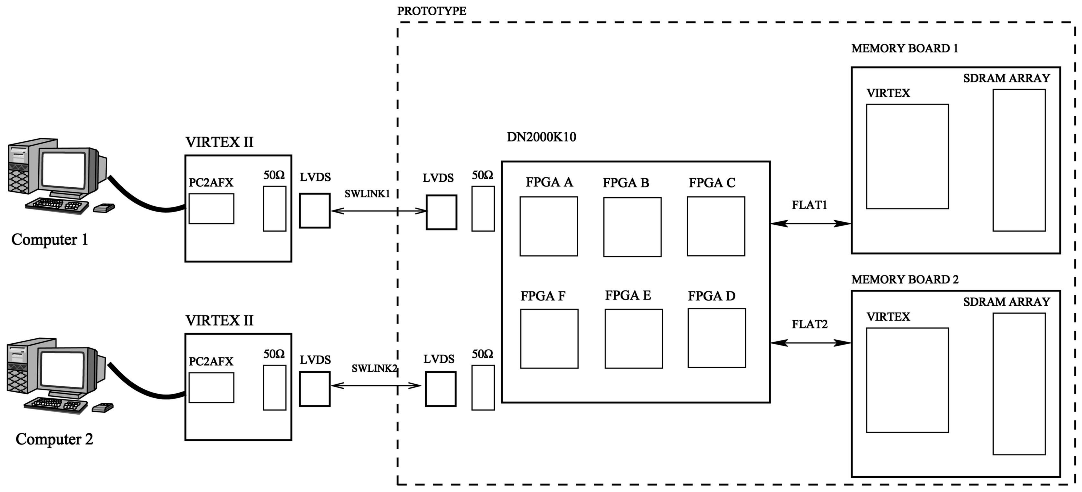


Fig. 16. Prototype setup.

TABLE XI
FTSSMM Blocks Partitioning

Subsystem Name	Board Name
IMAM	Memory Board
Routing Module	DN2000K10
System Control Unit	DN2000K10
I/O memory interfaces	DN2000K10
I/O link interfaces	DN2000K10

of remote terminals accessing to the FTSSMM. The emulators have been implemented on two computers interfaced to the FTSSMM through SpaceWire links (SWLINK1, SWLINK2). The overall prototype setup including the emulators of remote terminals is shown in Fig. 16. As can be seen, the hardware blocks implementing the overall test bed are as follows.

1) two computers,
2) two PC2AFX boards,
3) two Virtex II prototyping boards,
4) four national LVDS47/48EVK boards,
5) one DN2000k10 fast prototyping board,
6) two memory boards.

In the following, each of the composing hardware blocks will be described and the partitioning of the design will be shown. Fig. 17 and Fig. 18 show the prototype setup.

A. Computers

They are used to implement the remote terminals. A software implementing both high level functions such as file read, write, and delete and low level functions such as single packet send or receive useful for debug purposes has been developed. From the functional standpoint, the software is composed of two parts:

1) Formatting of data and command packets. The data to be exchanged with the FTSSMM must be

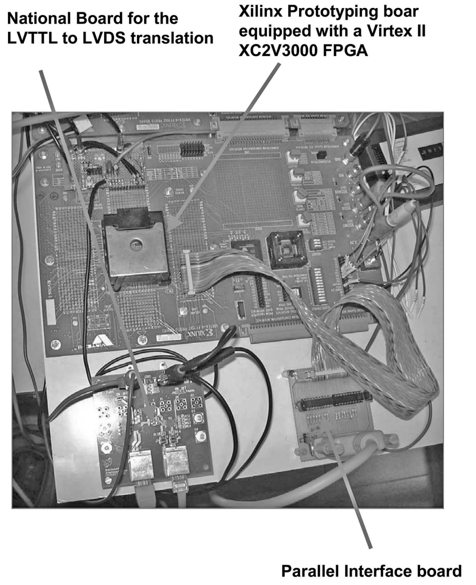

National Board for the LVTTL to LVDS translation
Xilinx Prototyping boar equipped with a Virtex II XC2V3000 FPGA
Parallel Interface board

Fig. 17. Prototype setup: Virtex II prototyping board.

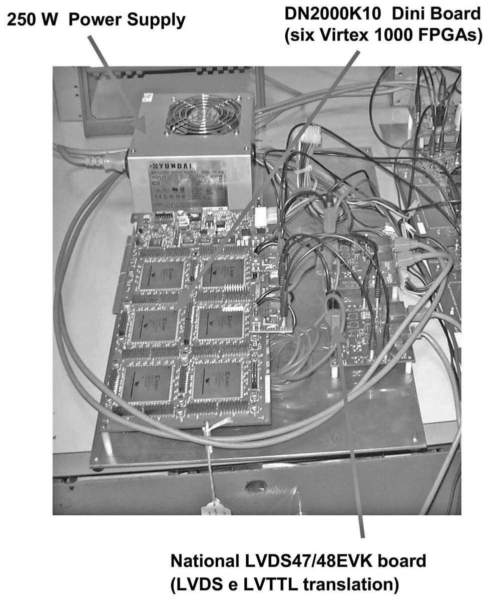

250 W Power Supply
DN2000K10 Dini Board (six Virtex 1000 FPGAs)
National LVDS47/48EVK board (LVDS e LVTTL translation)

Fig. 18. Prototype setup: Dini board.

TABLE XII
Test Software Developed

Routine Name	Routine Function
BT	Bitmap to ASCII file image translation and vice versa
GC	FTSSMM commands formatting
INP	Translation from ASCII data to IEEE 1355 packets and vice versa
INTERFACE	Parallel port bi-directional management

TABLE XIII
Logic Modules Implemented on XC2V3000

Module Name	Module Function
Parallel	Parallel port Handshake management handles parallel port half duplex communication between PC and FPGA
ParHandler	Data codification from 8 bit data of the parallel port to the 9 bit data of the SpaceWire interface
SpaceWire	block implementing the SpaceWire protocol

formatted in data packets in order to be compatible with the FTSSMM itself. The commands to be sent to the FTSSMM correspond to special command packets (read, write, delete commands for example).

2) Bi-directional interface through the parallel port (in the next release an HS USB port will be used).

The tests have been realized by using the FTSSMM to memorize and retrieve very big data files containing high resolution images. In Table XII, the list of the implemented routines is reported.

All the developed code has been written in C while the GUI has been implemented by using the Tcl and Tk graphic toolkits.

B. PC2AFX Boards

These boards have been developed in order to implement voltage translation from the TTL levels of the parallel ports to the low voltage TTL (LVTTL) levels of the HW-AFX Xilinx Boards. Two buffers have been used. A bi-directional buffer is used for the data bus. A directional buffer is used for control signals (Fig. 16).

C. Virtex II Prototiping Boards

These Xilinx prototyping boards host a Xilinx Virtex II XC2V3000 FPGA, a PROM for the configuration bitstream and some control logic, and have been used for the implementation of the SpaceWire protocol and its interface to the parallel ports of the two remote terminal emulators. In Table XIII the main blocks implemented on the XC2V3000 FPGA are described.

TABLE XIV
Logic Blocks Partition on DN2000K10 Board

Block Name	Block Function
FPGA A	SpaceWire Interface 1
FPGA B	SpaceWire Interface 2
FPGA C	Routing Module
	I/O link interfaces
	I/O memory interfaces
FPGA D	System Control Unit

D. National LVDS 47/48 EVK Boards

These boards have been used for the voltage level translation from the single ended LVTTL standard to the low voltage differential signaling (LVDS) differential standard adopted in the SpaceWire protocol. The boards host a two channel differential line driver and a two channel differential receiver. The inputs to the board are two SMB female connectors; the differential pairs in input and output are available on two RJ45 CAT5 connectors [35]. The LVDS boards are connected to the FPGA board by using a 50 Ω coaxial cable. Because of the inability of the FPGA LVTTL outputs to drive a 50 Ω transmission line [36], a small board with a 50 Ω buffer has been developed.

E. DN2000k10 Fast Prototyping Board

This is a very complex board for fast ASIC prototyping based on six Xilinx Virtex XCV1000-4 FPGAs [37]. The board hosts 8 Mbyte of Flash memory to memorize the FPGAs configurations, canned oscillators, and low skew clock drivers. A very large number of headers both on the top and bottom of the board allows a simple board interfacing. As shown in Table XI the DN2000k10 board has been used in order to map the core logic functions of the FTSSMM, in particular in Table XIV the partition of the design on the FPGAs provided by the board is shown.

The interconnections from the DN2000k10 board and the memory boards has been implemented by using flat cables (see Fig. 16).

F. Memory Boards

The memory boards have been implemented on two PCBs as can be seen in Fig. 19.

1) HOST PCB: ad-hoc developed backplane housing SDRAM arrays (4 GB) and a Virtex XCV1000 fast prototyping board (1).

2) HW-AFX-BG560-100 fast prototyping board housing a Virtex XCV1000 device implementing the logic for the IMAM handling.

The logic functions implemented on each VIRTEX XCV1000 are related to the SDRAM control, RS

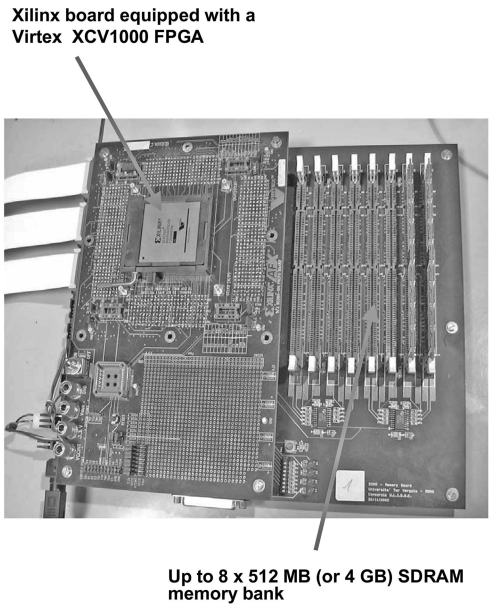

Xilinx board equipped with a Virtex XCV1000 FPGA
Up to 8 x 512 MB (or 4 GB) SDRAM memory bank

Fig. 19. Memory module.

TABLE XV
Logic Blocks on VIRTEX XCV 1000 Devices

Block Name	Block Function
MMC	Memory module controller: handles the handshake of the data I/O with the rest of the system (FTSSMM dynamic router)
EDAC	Error Detection and Correction performs the Reed-Solomon coding/decoding of the data stored in the memory
MAC	Memory Address Controller: handles the data I/O on the memory chips performing the necessary handshaking of control signals for accessing DRAM arrays

coding, and system interfacing with the rest of the FTSSMM. In Table XV the logic blocks are sketched.

VII. CONCLUSIONS

In this paper the system design methodology, the architecture, and the performance evaluations of an FTSSMM for satellite applications has been shown. The behavior, architecture, and implementation of its building blocks have been described in detail both for their normal functionality and fault tolerant capabilities. A detailed analysis of the system reliability and data integrity is reported. The graceful degradation capability of our system allows different levels of acceptable performances, in terms of active I/O link interfaces and storage capability. Different from other proposed solutions, that are based on fixed RS codes, our architecture uses reconfigurable RS codes, showing that the overall reliability of the FTSSMM is almost the same using different RS coding schemes allowing a dynamic reconfiguration of the coding to reduce the latency (shorter codewords) or to improve the data integrity (longer codewords). The use of a scrubbing technique can be useful if a high SEU rate is expected, or if the data must be stored for a long period in the FTSSMM. The reported simulations show the behavior of the FTSSMM in the presence of permanent and transient faults. In fact, we show that the SCU is able to recover from transient faults. On the other hand, using a spare microcontroller, hard faults can be tolerated. The original approach of using a distributed file system confines the unrecoverable fault effects only in a single I/O interface. In this way, the FTSSMM maintains its capability to store and read data. The proposed system allows obtaining FTSSMM characterized by high reliability and high speed due to the intrinsic parallelism of the switching matrix compared with other systems based on a bus architecture.

REFERENCES

[1] Kluth, M. P., Simon, F., Le Gall, J. Y., and Muller, E.
Design of a fault tolerant 100 gbits solid-state mass memory for satellites.
In *Proceedings of 14th VLSI Test Symposium*, 1996, 281–286.

[2] Fichna, T., Gartner, M., Gliem, F., and Rombeck, F.
Fault-tolerance of spacebome semiconductor mass memories.
In *Twenty-Eighth Annual International Symposium on Fault-Tolerant Computing, Digest of Papers*, 1998, 408–413.

[3] Fox, J., Abare, W. E., and Ross, A.
Suitability of cots ibm 64mb dram in space.
In *Proceedings of Fourth European Conference on Radiation and Its Effects on Components and Systems* (RADECS 97), 1997, 240–244.

[4] Underwood, C. I., and Oldfield, M. K.
Observations on the reliability of cots-device-based solid state data recorders operating in low-Earth orbit.
IEEE Transactions on Nuclear Science, **47**, 4 (June 2000), 647–653.

[5] Parkes, S. M.
Spacewire: The standard.
In *DASIA'99*, 1999, vol. (ESA SP-447) (ISBN 92 9092 788 7), 111–116.

[6] Cardarilli, G. C., Marinucci, P., Ottavi, M., and Salsano, A.
A fault-tolerant 176 gbit solid state mass memory architecture.
In *Proceedings of the International Symposium on Defect and Fault Tolerance in VLSI Systems* (DFT '00), 2000, 173–180.

[7] Oldham, T. R., Bennett, K. W., Beaucour, J., Carriere, T., Polvey, C., and Garnier, P.
Total dose failures in advanced electronics from single ions.
IEEE Transactions on Nuclear Science, **40**, 6 (Dec. 1993), 1820–1830.

[8] Johnston, A. H.
Radiation effects in advanced microelectronics technologies.
IEEE Transactions on Nuclear Science, **45**, 3 (June 1998), 1339–1354.

[9] European Space Agency
SpaceWire Homepage:
http://www.estec.esa.nl/tech/spacewire/index.html.

[10] Maeusli, D., Teston, F., Vuilleumier, P., and Harboe-Sorensen, R.
ESA developments in solid sate mass memories.
Preparing for the Future, (ESA Publication Division), **5**, 2 (June 1995).

[11] MIL-STD-1553.

[12] Ziegler, J. F., and Nelson, M. E., et al.
Cosmic ray soft error rates of 16-mb dram memory chips.
IEEE Journal of Solid-State Circuits, **33**, 2 (Feb. 1998).

[13] Bertazzoni, S., Cardarilli, G. C., Di Giovenale, D., Grande, G. C., Piergentili, D., Salmeri, M., Salsano, A., and Sperandei, S.
Failure tests on 64mb sdram in radiation environment.
In *Proceedings of the International Symposium on Defect and Fault Tolerance in VLSI Systems*, (DFT '99), 1999, 158–164.

[14] Blahut, R. E.
Theory and Practice of Error Control Codes.
Reading, MA: Addison-Wesley, 1983.

[15] Paar, C., and Rosner, M.
Comparison of arithmetic architectures for reed-solomon decoders in reconfigurable hardware.
In *Proceedings of the Symposium on Field-Programmable Custom Computing Machines*, Apr. 1997, 219–225.

[16] Wilhelm, W.
A new scalable VLSI architecture for Reed-Solomon decoders.
IEEE Journal of Solid-State Circuits, **34** (Mar. 1999), 388–396.

[17] Kwon, S., and ShinDept, H.
An area-efficient VLSI architecture of a Reed-Solomon decoder/encoder for digital VCRs.
IEEE Transactions on Consumer Electronics, **43** (Nov. 1997), 1019–1027.

[18] Mahmood, A., and McCluskey, E. J.
Concurrent error detection using watchdog processors—A survey.
IEEE Transactions on Computers, **37**, 2 (Feb. 1988), 160–174.

[19] Saxena, N. R., and McCluskey, E. J.
Parallel signature analysis design with bounds on aliasing.
IEEE Transactions on Computers, **46**, 4 (Apr. 1997), 425–438.

[20] Cardarilli, G. C., Marinucci, P., and Salsano, A.
Development of an evaluation model for the design of fault-tolerant solid state mass memory.
In *Proceedings of the IEEE International Symposium on Circuits and Systems* (ISCAS2000), vol. 2, May 2000, 673–676.

[21] MIL-HDBK 338 B-6.3.5.

[22] Cardarilli, G. C., Leandri, A., Marinucci, P., Ottavi, M., Pontarelli, S., Re, M., and Salsano, A.
Design of a fault tolerant solid state mass memory.
IEEE Transactions on Reliability, **52**, 4 (Dec. 2003), 476–491.

[23] Labeau, F., Desset, C., Macq, B., and Vandendorpe, L.
Approximating the protection offered by a channel code in terms of bit error rate.
In *Proceedings of the European Signal Processing Conference*, Rhodes, Greece, 1999.

[24] Saleh, A. M., Serrano, J. J., and Patel, J. H.
Reliability of scrubbing recovery-techniques for memory systems.
IEEE Transactions on Reliability, **39** (Apr. 1990), 114–122.

[25] Cardarilli, G. C., Ottavi, M., Pontarelli, S., and Salsano, A.
A fault tolerant hardware based file system manager for solid state mass memory.
In *Proceedings of the 2003 International Symposium on Circuits and Systems* (ISCAS 2003), vol. 5, V-649–V-652.

[26] Meyer, J. F.
On evaluating the performability of degradable computing systems.
IEEE Transactions on Computers, **C-29** (Aug. 1980), 720–731.

[27] Lala, P. K.
Fault Tolerant and Fault Testable Hardware Design.
Englewood Cliffs, NJ: Prentice-Hall, 1985.

[28] Gracia, J., Baraza, J. C., Gil, D., and Gil, P. J.
Comparison and application of different vhdl-based fault injection techniques.
In *Proceedings of International Symposium on Defect and Fault Tolerance in VLSI Systems*, 2001, 233–241.

[29] Parrotta, B., Rebaudengo, M., Reorda, M. S., and Violante, M.
New techniques for accelerating fault injection in VHDL descriptions.
In *Proceedings of 6th IEEE International On-Line Testing Workshop*, 2000, 61–66.

[30] Jenn, E., Arlat, J., Rimen, M., Ohlsson, J., and Karlsson, J.
Fault injection into VHDL models: The mefisto tool.
In *Proceedings of 24th International Fault Tolerant Computing Symposium* (FTCS-24), Austin, TX, June 1994, 66–75.

[31] Cardarilli, G. C., Kaddour, F., Leandri, A., Ottavi, M., Pontarelli, S., and Velazco, R.
Bit flip injection in processor-based architectures: A case study.
In *Proceedings of 8th IEEE International On-Line Testing Workshop* (IOLTW02), Isle of Bendor, France, July 2002.

[32] Berger, G., Ryckewaert, G., Harboe-Sorensen, R., and Adams, L.
Cyclone—A multipurpose heavy ion, proton and neutron see test site.
In RADECS Radiation and its effects on Components and Systems.

[33] Aldec, Inc.
Active-HDL, VHDL Reference Guide, Apr. 2001.

[34] Ottavi, M., Pontarelli, S., Re, M., Salsano, A., Cardarilli, G. C., and Leandri, A.
A methodology for program-flow checking in microcontrollers: Checker design and performance evaluation.
Internal report, available upon request.

[35] National Semiconductors Inc.
Homepage: http://www.national.com.

[36] Xilinx Inc.
Homepage: http://www.xilinx.com.

[37] The Dini Group
Homepage: http://www.dinigroup.com.

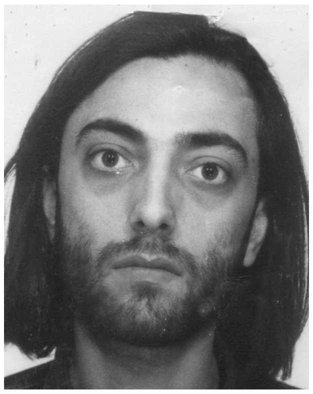
Gian Carlo Cardarilli received the Laurea (summa cum laude) in 1981 from the University of Rome "La Sapienza."

He has been with the University of Rome "Tor Vergata" since 1984, where he is currently full professor of digital electronics and electronics for communication systems. During the years 1992–1994 he worked for the University of L'Aquila. From 1987–1988 he worked for the Circuits and Systems team at EPFL of Lausanne, Switzerland. His interests are in the area of VLSI architectures for signal processing and IC design. In particular, he works in the filed of computer arithmetic and its application to the design of fast signal digital processor. he has also developed mixed-signal neural network architectures implementing them in silicon technology. Recently, he also proposed different new solutions for the implementation of fault-tolerant architectures.

He has published over 140 papers in international journals and conferences. He has also worked in cooperation with companies like Alenia Aerospazio, Rome, Italy; STM, Agrate Brianza, Italy; Micron, Avezzano, Italy; Ericsson Lab, Rome Italy and with a lot of SMEs.

Marco Ottavi (M'04) received the Laurea degree in electronic engineering from the University of Rome "La Sapienza" in 1999 and the Ph.D. in microelectronics and telecommunications from the University of Rome "Tor Vergata" in 2004.

In 2000 he was with ULISSE Consortium, Rome, as a designer of digital systems for space applications. In 2003 he joined the Department of Electrical and Computer Engineering of Northeastern University, Boston, as a visiting research assistant. He is currently postdoctoral research associate in the same department. His research interests include yield and reliability modeling, fault-tolerant architectures, on-line testing and design of nano scale circuits and systems.



Salvatore Pontarelli received the Laurea degree in electronic engineering from the University of Bologna in 1999 and the Ph.D. in microelectronics and telecommunications engineering from the University of Rome "Tor Vergata" in 2003.

Currently he has a postdoctoral fellowship with the Department of Electronic Engineering of the University of Rome "Tor Vergata." His research mainly focuses on fault tolerance, on-line testing, and reconfigurable digital architectures.

Marco Re received the Laurea degree in electronic engineering from the University of Rome "La Sapienza" in 1991 and the Ph.D. in microelectronics and telecommunications engineering from the University of Rome "Tor Vergata" in 1996.

In 1998 he joined the Department of Electronic Engineering of the University of Rome "Tor Vergata" as a researcher. He was awarded two one-year NATO fellowships with the University of California at Berkeley in 1997 and 1998. His main interests and activities are in the area of DSP algorithms, fast DSP architectures, fuzzy logic hardware architectures, hardware-software codesign, number theory with particular emphasis on residue number system, computer arithmetic and CAD tools for DSP, fault tolerant and self-checking circuits.

Dr. Re has authored and coauthored more than eighty papers.

Adelio Salsano was born in Rome on December 26, 1941 and is currently full professor of microelectronics at the University of Rome "Tor Vergata" where he teaches the courses of microelectronics and electronic programmable systems. His present research work focuses on the techniques for the design of VLSI circuits, considering both the CAD problems and the architectures for ASIC design. In particular, of relevant interest are the research activities on fault tolerant/fail safe systems for critical environments as space, automotive etc., on low power systems considering the circuit and architectural points of view, and on fuzzy and neural syste4ms for pattern recognition.

Dr. Salsano has an international patent and has written or presented more than 90 papers. At present he is the president of the national consortium named U.L.I.S.S.E., between ten universities, three polytechnics, and several of the biggest national industries, such as STMicorelectronics, ESAOTE, FINMECCANICA. He is responsible for contracts with the ASI (Italian Space Agency), for the evaluation and use in space environment of COTS circuits and for the definition of new suitable architectures for space applications. professor Salsano is also involved in professional activities in the fields of information technology and is also consultant to many public authoriti8es for specific problems. In particular he is consultant to the Departments of Research and of Industry, of IMI, and of other authorities for the evaluation of industrial public and private research projects. he was a member of the Consulting Committee for Engineering Sciences of the CNR (National Research Council) from 1981 to 1994 and participated in the design of public research programs in the field of telematics, telemedicine, office automation, telecommunication and recently, microelectronics and bioelectronics.