IMTC 2004 - Instrumentation and Measurement
Technology Conference
Como, Italy, May 18-20, 2004

A Fault-Tolerant Solid State Mass Memory for Highly Reliable Instrumentation

G.C. Cardarilli, M. Ottavi, S. Pontarelli, M. Re, A. Salsano
Department of Electronic Engineering
University of Rome Tor Vergata
00133 Rome Italy Phone: (+39-06) 7259-7370,
email: marco.re@icee.org

Abstract *– Often, in space missions, a large amount of data from on board instrumentation must be stored in highly reliable mass memories. In this paper the implementation of a prototype of a Solid State Mass Memory (SSMM) for high reliability demanding applications is presented. A description of the SSMM architecture is given together with an in depth description of the prototype. The SSMM has been implemented by using a fast prototyping methodology. By using this technique, a flexible re-programmable test bed useful for the testing of both the conventional and faulty (using the fault-injection approach) functionality of the system has been obtained.*

Keywords *– SSMM, Mass Memory, Fault Tolerance*

I. INTRODUCTION

Frequently, satellite missions must acquire a lot of measurement data from on board instrumentation. The measurements must be performed in hostile environment, where a high number of errors is possible, and normally, a large amount of data must be collected for the transmission to the earth station. For example, data from on board instrumentation used in scientific experiments, or images coming from meteorological satellites (e.g. SAR images) can require a storing capacity of many GBytes and a large data transfer bandwidth. To satisfy these requirements, SSMM architectures are used. They are characterized by high reliability and high performances in terms of data transfer rate and capacity. In this paper a SSMM based on Commercial off The Shelf (COTS) components is described together with its implementation by using fast prototyping techniques. The paper is organized as follows: Section II illustrates the chosen design methodology while Section III shows the SSMM architecture. The description of the prototype is given in Section IV and the Conclusions are drawn in Section V.

II. DESIGN METHODOLOGY

The development of electronic systems suitable for space applications requires particular attention to obtain satisfactory levels of reliability being these systems embedded in an hostile environment in which mechanical stresses, ionizing radiations and critical thermal conditions are present. To face these problems, the typical design approach uses space qualified devices based on special and expensive technology processes. However, space qualified components are often not upgraded (due to the small market) and less performing (due to technological reasons). These disadvantages can be faced by using COTS components together with suitable system level design methodologies in order to match the severe reliability requirements of space applications. The SSMM architecture and design methodology presented in this paper, has been chosen in order to deal with the typical fault set defined for the space environment, i.e. *Single Event Upset* (SEU) faults, caused by ionizing particles, and *stuck-at faults*, related to the Total Ionizing Dose (TID) [3],[4].

Moreover, fault detection and dynamic system recovery fault tolerant techniques have been used for the SSMM design. These techniques has been chosen for the following reasons:

1. low hardware redundancy and low power consumption requirements
2. the SSMM can tolerate out-of-order time as it does not perform time-critical operations.

On line fault detection and reconfiguration techniques that imply the presence of a MTTR (Mean Time To Repair) but also low area and power consumption with respect to traditional NMR (N - Modular Redundancy) design have been chosen for the final architecture. A number of SpaceWire data links [5] accesses the memory banks through a cross-point switch matrix [1]. This solution is convenient with respect to a bus based architecture in terms of bandwidth, latency and reconfiguration capability as the failure of a connection does not compromise the entire connection of the network but only the access to a specific node. Moreover, in order to improve both fault tolerance and memory usage, we implemented a distributed file system. Most of the file system functions are hardware based and handled locally on each memory module.

III. ARCHITECTURE DESCRIPTION

At the top level, the SSMM can be considered as a black box connected to different satellite apparatuses. A number of bi-directional serial links are used for high-speed data exchange. For these links the Spacewire (IEEE 1355 DS-DE) protocol [5] has been chosen. Each Spacewire link is able to carry information (data or commands) at about 100 Mbit/sec over distances of up to 10 meters. Moreover, the SSMM is connected with a MIL 1553 Bus, which is widely used in satellite platforms due

0-7803-8248-X/04/$17.00 ©2004 IEEE

to its physical redundancy (dual twisted pair bus structure) [2]. Two main units compose the SSMM architecture (Fig. 1):

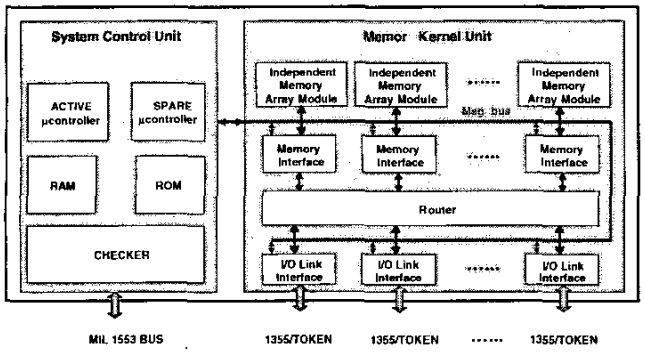

Fig. 1. SSMM Architecture

1. The Memory Kernel Unit (MKU) manages the bi-directional data-flow 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 SSMM system is achieved both by means of architectural redundancies, and by introducing Error Detection And Correction Codes (EDAC), granting data integrity.

In the following each block of the architecture will be briefly described.

A. Memory Kernel Unit: general description

As shown in Fig. 1 the Memory Kernel Unit, is composed of four functional modules:

1. Independent Memory Array Modules (IMAM)
2. Routing Module (Router)
3. I/O Link Interfaces
4. I/O Memory Interfaces.

The memory kernel unit 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, 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. All I/O Link Interfaces and Memory Interfaces modules are connected to the SCU through a message bus (**Msg_bus**) that allows communicating either the detection of a fault, or the necessary messages to operate the packet routing control (Fig. 1). Each module has been developed using different fault tolerant methodologies, depending on the final reliability requirements and the functionalities performed. These choices will be described in the following subsections.

A.1 **Independent Memory Array Module (IMAM)**

Each IMAM module is composed of:

- Dynamic Random Access Memory (SDRAM) bank (composed of several COTS chips or MCMs).
- Control circuitry that interfaces the memory bank to the other components of the IMAM module
- Reed-Solomon (RS) coder-decoder which adds redundancy to the data stored into the SDRAM.

The SDRAM packages are arranged on 4 rows per board side and each row is composed of 18 packages. Each package implements a 8 bit symbol. Using both the sides of the board, we are able to implement either a Reed Solomon (RS) code [6] with a maximum codeword length of 144 symbols ($2\ sides * 4\ row * 18\ column = 144$ SDRAM packages) or a code with a minimum codeword length of 18 symbols. The data word length depends on the reliability and data integrity constraints. IMAM is able to support either variable data and codeword word-length.

A.2 **Routing Module**

The routing system connects the I/O Memory Interfaces with the I/O Link Interfaces through a crossbar switch matrix. An arbiter provides the acknowledge signals to the I/O interfaces that send data through the crossbar. The main blocks composing the routing module are:

- A crossbar switching matrix
- An access arbiter.

With this interconnection method, multiple parallel connections between users and resources can be established 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 system functionality.

A.3 **I/O Link Interfaces**

The I/O Link Interfaces provide the transport of data and messages between users (i.e. data from the on board instrumentation or data from the on satellite control unit) and the memory modules. The structure of the packets exchanged with the Spacewire interface is the following:

End Of Packet	*Payload*	*Header*

The packet header is one byte long and indicates the ID of the packet while the payload is composed of a variable number of bytes terminated by the End Of Packet (EOP) marker. We assume that header values in the range from 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 or diagnostic communications sent or received from the memory. Thus the file system can handle 255 files and the memory can be controlled and monitored by using the same links carrying the data. All the interfaces access the switch matrix in full-duplex mode, and request arbitration through dedicated links. The internal shared bus **Msg_bus** interconnects all the I/O interfaces and the micro-controller to provide file system management and error detection. A generic I/O link interface is composed of two main functional blocks: the "Data Routing Block", that handles the data flow, and the "Interface Controller" block, that connects the interface with the System Control Unit and provides also local error handling with the "Error Handler" function.

The main functions of the "Data Routing Block" are the following:

- LVDS I/F. This block implements the electrical interface between the differential signals LVDS (Low Voltage Differential Signaling) and Data and Strobe single ended signals.
- SpaceWire (1355 DS-DE) I/F. This interface interprets the serial signal, implements the flow and the parity control following the procedures of the SpaceWire protocol, extracts the clock signal and pushes the extracted data into the FIFO in parallel mode. The parallelism of the data is 8 bit + 1 flag bit to separate a data/header token from an EOP marker.
- LINK I/F. Represents the core of I/O Link interfaces. It is composed of a master and a slave. The master block manages a local table of 255 elements containing the dynamically reconfigurable output association for each file. The entries of the table are written by the SCU through the "Message Handler" block. Once the output association is set the master negotiates the output connection with the arbiter.

A.4 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. The partition of the file system in every module reduces the amount of data that can be lost in case of an unrecoverable failure in the FAT. In fact, in case of failure we have only the loss of local stored information. In order to handle the file system the memory interface implements the following functions:

- *Delete function*: used to delete a file from the FAT
- *Fragment function*: used to add to the FAT the occurrence of more fragments of the same file
- *Read function:* used to read a file from the memory
- *Write function:* used to write a file in the memory
- *Format function:* used to set-up the FAT in the initialization phase.

Each function is implemented with a separate block and the "Operation Handler" inside the Interface Controller performs the activation of each function. Moreover, each functional block is self-checking and a spare block is used to obtain fault tolerance. In fact, since the occurrence of a failure on a single block can be detected, the "Error Handler" inside the Interface Controller can activate the spare module with a low time overhead.

Both the "Operation Handler" and the "Error Handler" communicate with the rest of the SSMM through the message handler that is the third block composing the Interface Controller. Therefore, through the message bus, the SCU can control the status of each I/O Memory Interface both in case of normal function and in case of fault occurrence.

B. *System Control Unit*

The System Control Unit manages the access of the users and the resources of memory. This module is connected with the rest of the SSMM system through the internal communication Bus **Msg_Bus**, used for the communications service, and through the selection signal Sel that regulates the modality of access to the **Msg_Bus**. This subsystem uses two 8051 micro-controllers that can be connected or isolated from the system through bypass blocks. 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 micro-controller accesses to a 2k ROM memory, which contains the executable program, and to a 1K RAM memory that is used for the data storage and management. Data written and read from the memory is Hamming coded in order to face soft errors (SEU) occurrence as bit flips in memory cells. Moreover, a signature calculation block controls the correctness of the operations performed by the micro-controller. This block reads the sequence of the addresses output by the micro-controller and verifies they are following a correct sequence. The evaluation of the correct execution is performed with a specific application of a well-know fault detection technique called signature analysis [7], [8]. The error management masks system faults. If a faulty behavior persists, the active micro-controller is substituted by the spare one.

IV. PROTOTYPE SETUP

The development of the prototype of the SSMM was intended to obtain a simple but still representative version of the proposed design. The SSMM prototype is composed of only two memory modules and two link interfaces. In this manner all the features of the of SSMM can be tested while reducing the complexity of the first implementation. Moreover the use of a fast prototyping methodology, based on re-programmable FPGAs, allows an incremental hardware testing approach. The

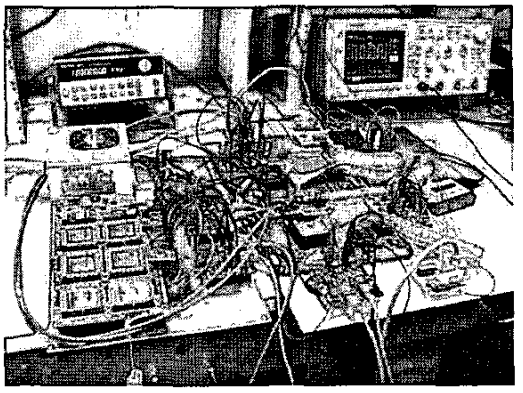

Fig. 3. Prototype Implementation

functional architecture of the prototype is shown in Fig. 1. This architecture has been partitioned and mapped on the following hardware as specified in Tab. I

Subsystem name	Board Name
IMAM	Memory Board
Routing Module	DINI BOARD
System Control Unit	DINI BOARD
I/O Memory Interfaces	DINI BOARD
I/O Link Interfaces	DINI BOARD

TABLE I
SSMM BLOCKS PARTITIONING

The prototype has been tested implementing two emulators of remote terminals accessing to it. These emulators have been implemented on two computers interfaced to the SSMM through Spacewire links. The overall prototype setup including the testing elements is shown in Fig. 2. As can be seen, the hardware blocks implementing the system are:

1. Two Computers
2. Two Virtex II Prototyping Boards
3. One DN2000k10 Fast Prototyping Board
4. Two PC2AFX Boards
5. Two Memory Boards
6. Four National LVDS47/48EVK Boards.

In the following, each of the composing HW blocks will be described and the partitioning of the design will be shown. Fig. 3 shows the implemented prototype.

A. Computers

On the two computers has been developed 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. From the functional standpoint, the software is composed of two functional parts

Routine name	Routine function
BT	Bitmap to ASCII file image translation and vice versa
GC	SSMM commands formatting
INP	Translation from ASCII data to IEEE 1355 packets and vice versa
INTERFACE	Parallel port bidirectional management

TABLE II
TEST SOFTWARE DEVELOPED

Mod. Name	Mod. 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 IP I/O
SpaceWire	block implementing the SpaceWire protocol

TABLE III
LOGIC MODULES IMPLEMENTED ON THE XC2V3000

- Formatting of the data to be exchanged with the SSMM in a special format compatible with the SSMM. Creation of special packets to send commands to the SSMM (read, write delete commands for example).
- Bidirectional interface through the parallel port (in the next release the USB port will be used)

In Table II, the list of the implemented routines is reported

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

B. PC2AFX Boards

These boards have been developed in order to implement voltage translation from the TTL levels of the parallel port to the LVTTL levels of the HW-AFX Xilinx Board. Two buffers have been used. A bidirectional buffer is used for the data bus. A directional buffer for control signals. These boards have been mounted on the two HW-AFX boards (Fig. 2).

C. Virtex II Prototyping Boards

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

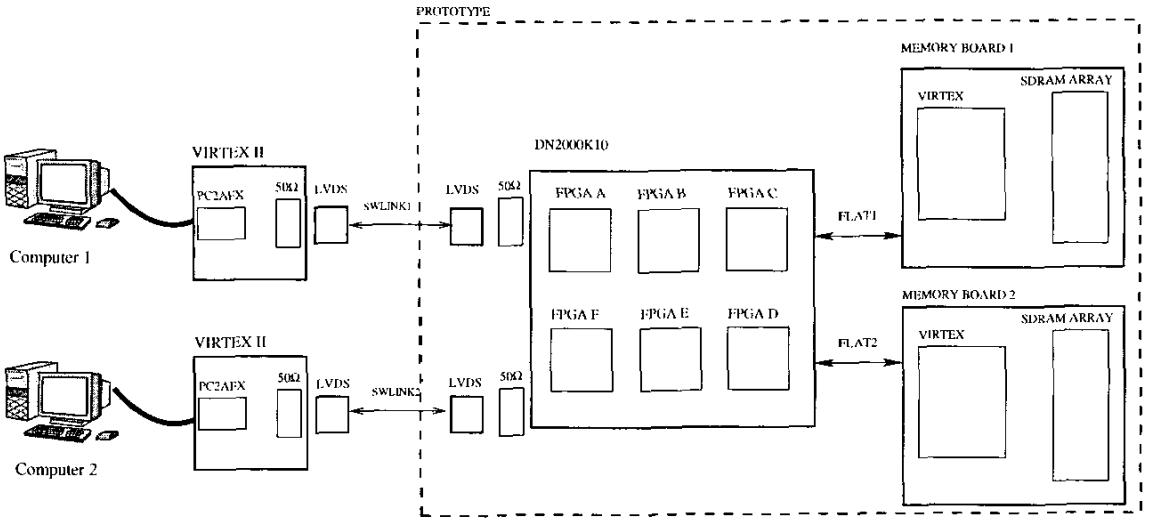

Fig. 2. Prototype Setup

D. 50 Ω Buffer Board

The input-output signals from the SpaceWire link are four. Two inputs (data in and strobe in) and two outputs (data out and strobe out). These signals are connected to a couple of boards used to for the conversion from the LVTTL levels to the LVDS standard levels (see the next section). The LVDS board are connected to the FPGA board by using a 50 Ω coaxial cable. The LVTTL outputs of the FPGA are not able to drive a 50Ω transmission line. In order to solve this problem a small board with a 50Ω buffer has been developed.

E. National LVDS 47/48 EVK Boards

These boards have been used for the voltage level translation from the single ended LVTTL standard to the LVDS differential standard adopted in the SpaceWire protocol. The boards host one two channels differential line driver and one two channels differential receiver. The inputs to the board are two SMB female connectors, the input and output differential pairs are available on two RJ45 CAT5 connectors.

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

TABLE IV
LOGIC BLOCKS PARTITION ON DN2000K10

Block Name	Block Function
MMC	memory module controller: handles the handshake of the data I/O with the rest of the system (SSMM 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.

TABLE V
LOGIC BLOCKS IMPLEMENTED ON THE VIRTEX XCV1000 DEVICE

F. DN2000k10 Fast Prototyping Board

The DN2000k10 [9] is a very complex board for fast ASIC prototyping based on six Xilinx Virtex XCV1000-4 FPGAs. 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 I the DN2000k10 board has been used in order to map the SSMM core functions, in particular in Table IV the partition of the design on the FPGAs provided by the board is shown. The interconnections from the DN2000k10 board and the memory banks has been implemented by using flat cables.

G. Memory Boards

The Memory Boards are a compound subsystem of the SSMM. The subsystem is implemented on two boards:

1. HOST PCB :ad-hoc developed PCB housing SDRAM arrays (4 GB) and a VIRTEX XCV1000 fast prototyping board.
2. HW-AFX-BG560-100 fast prototyping board housing a Virtex XCV1000 device implementing the logic of IMAM handling.

The logic functions implemented on each VIRTEX XCV1000 are related to the SDRAM control, Reed Solomon Coding and System Interfacing with the rest of the SSMM. In Table V the logic blocks are reported and described.

V. CONCLUSIONS

In this paper the fast prototyping based implementation of a Solid State Mass Memory has been presented. At present, the measurements on the SSMM are in progress. The basic operations performed on the prototype are both functional testing, in order to evaluate the performances of the system, and fault injection in order to test the system in emulated fault occurrences.

First results show a good behavior of the system, also in the case of injected faults. We are now testing the effects of the faults on the SSMM performance, in order to validate the behavior of the mechanisms introduced for obtaining graceful degradation. In particular, due to the complexity of the interactions among the different elements involved in the above mechanisms, we observe that the use of rapid-prototyping based implementation, if compared to the simulative analysis, greatly improves the accuracy of performance estimation of faulty SSMM.

REFERENCES

[1] G.C. Cardarilli, P. Marinucci, M. Ottavi, A. Salsano: 'A Fault-tolerant 176 GBit Solid State Mass Memory Architecture', Defect and Fault Tolerance in VLSI Systems, 2000. DFT '00. International Symposium on , 2000 Page(s): 173 -180

[2] MIL-STD-1553

[3] T.R Oldham, K.W Bennett, J. Beaucour, T. Carriere, C. Polvey, P. Garnier: "Total dose failures in advanced electronics from single ions", Nuclear Science, IEEE Transactions on, Vol. 40 Issue: 6 Part: 1-2 , Dec. 1993 Page(s): 1820 -1830

[4] A.H. Johnston: "Radiation effects in advanced microelectronics technologies" Nuclear Science, IEEE Transactions on , Volume: 45 Issue: 3 Part: 3 , June 1998 Page(s): 1339 –1354

[5] S.M. Parkes, "Spacewire: The Standard", DASIA'99

[6] R. E. Blahut. Theory and Practice of Error Control Codes. Addison-Wesley Publishing Company, 1983.

[7] Mahmood, A.; McCluskey, E.J. "Concurrent error detection using watchdog processors-a survey" Computers, IEEE Transactions on , Volume: 37 Issue: 2, Feb. 1988 Page(s): 160 -174

[8] Saxena, N.R.; McCluskey, E.J. "Parallel signature analysis design with bounds on aliasing" Computers, IEEE Transactions on , Volume: 46 Issue: 4 ,April 1997 Page(s): 425–438

[9] www.dinigroup.com