A Fault-Tolerant Middleware Switch for Space Applications

Sergio Montenegro
Institute of Space Systems
German Aerospace Agency (DLR)
Bremen, Germany
e-mail: sergio.montenegro@dlr.de

Ebrahim Haririan
Institute of Space Systems
German Aerospace Agency (DLR)
Bremen, Germany
e-mail: ebrahim.haririan@dlr.de

Abstract— **Typical data systems for space applications are computer-centric. The central component is a computer to which several devices are attached. The computer handles devices, communication, computation and storage of data. Furthermore, fault-tolerance is an important issue in space systems.**

This paper presents a novel multicast embedded middleware switch which at the first implementation is fully implemented on an FPGA. SRAM-based FPGAs are very susceptible to SEUs stemmed from radiation effects in space, therefore considering fault-tolerance is inevitable. High capability of this switch to handle different interfaces in form of an integrated single system together with its fault-tolerance feature makes it very suitable for space data handling applications.

Keywords: FPGA, SEU, fault-tolerance, middleware switch, TMR, EDAC

I. INTRODUCTION

SRAM-based FPGAs (Field Programmable Gate Arrays) are very susceptible to SEUs (Single-Event Upsets) stemmed from radiation effects in space applications. Considering reliability as a key to success in space missions, need for fault-tolerant design is inevitable. To apply fault tolerance to an embedded multicast switch soft core, a design-level TMR (Triple Module Redundancy) approach was chosen despite its more power consumption and impacts in timing and area.

Due to high increase of area with TMR, a thrifty fault-tolerant (thriFTy) method was applied into the switch core to minimize fault-tolerance overhead. The major concerns in this method are the memory modules, FFs (Flip-Flops) and consequently FSMs (Finite State Machines).

Every flip-flop of the system is replicated and voted according to the TMR approach. This includes all registers and state machines. To implement TMR state machines, states require an explicit encoding scheme. Different state encoding schemes are studied to best fit the requirements of the targeted middleware core, to make it radiation-hardened up to a reasonable level of reliability.

Memory modules are protected against SEUs by means of EDAC (Error Detection And Correction) codes. Error detection and correction is applied to highly-reliable and high-performance applications. The algorithm employed for error detection and correction in the MWS (Middleware Switch) core is the Hamming code. It detects double bit errors and corrects single bit errors anywhere within the memory.

Since data path and routing information are stored in the configuration memory of FPGAs, scrubbing the configuration memory is proposed by the thriFTy method instead of replication of the data path. The thrifty fault-tolerant method, presents more reasonable efficiency results in words of resource usage and maximum frequency of the system, compared to full replication method.

Section II gives a thorough overview on the system under development – the MWS (Middleware Switch) core – in detail. As the first implementation of the MWS core is on FPGAs, radiation effects on FPGAs will be discussed in section III. The paper follows representing fault-tolerance applied to the MWS core in section IV.

At the end, a comparison between the non-FT and thrifty fault-tolerant design represents a moderate increase of resources usage and less decrease of the maximum frequency of the switch logic circuit, compared to the fully replicated approaches.

II. MIDDLEWARE SWITCH

A. Core Avionic Systems

The term ‘core avionic system’ in this context is referred to a special avionic system with a computer as an integral part of the system. This on-board computer performs the whole control of the satellite. Typically this is divided into some subtasks including command handling, data handling, time management, system health monitoring, attitude control, onboard navigation and power and thermal management [3].

Health monitoring is the supervision of all systems and subsystems. An important aspect of health monitoring is memory scrubbing to prevent the accumulation of bit-flips induced by single-event effects.

The data management system on board is similar to many other terrestrial embedded systems but in space, there are very strict constraints and difficulties. Reliability is a very important issue for space applications. This property is extremely important for the data management systems and is the main cause for its high costs in comparison to terrestrial embedded systems.

One of the important aspects of reliability is fault-tolerance. Even if the components are not permanently damaged, malfunctions are to be expected in any engineering

system. The system must be able to recognize such anomalies to correct them before they have wider consequences. On the ground, similar requirements can be found in case of safety-critical systems. Railway, airplane or nuclear reactor control systems are of those examples. Whereas ground applications are protected very well against the cosmic radiation, the core avionic systems, however, are not. This radiation causes a huge amount of data corruptions (bit flips) that leads to quicker ageing of the electronic components in such applications.

B. A Middleware Switch as a Core Avionic System

In 2007, the German Aerospace Center (DLR) planned to develop and operate a DLR-owned Standard Satellite Bus (SSB) suitable for different types of missions and applications. SSB aims to provide necessary facilities for satellite development and satellite operations. The program is in continue of other DLR projects framework referred as the Compact Class describing satellites of approximately 100kg overall mass and dimensions, which allow piggy-back launches.

A dedicated project named *Projekt Kompaktsatellit*[1] was initiated within the DLR's Institute of Space Systems for the development and operation of the desired DLR's standard satellite bus. The *Kompatksatellit* project aims to develop a Standard Satellite Bus called SSB which will be able to handle different missions in the compact satellite class defined above. A major step in this project, is to improve dependability, flexibility and simplicity of the whole core avionic system dealing with different aspects of core avionics development to meet current and future requirements regarding flexibility, availability and reliability of small satellites.

The main factors in a typical avionic system development are complexity, software-hardware interfaces and the difficulties to handle different interfaces in a single system. The new avionic concept targets these problems and aims to provide a very simple integrated solution of software and hardware, hiding the border between them.

The emerging and fast growing FPGA technology makes it possible to implement the biggest part of the avionic system in software, including classical CPU software and FPGA software. The use of fixed hardware is kept to a minimal limit. In this concept, the system's functionality is provided by a network of services. Some of these services are implemented in classical CPU-software, some in FPGA-software and some in hardware devices. To access any service there is no difference in how it is implemented and where it runs. The avionics system is a distributed computer system. No single node is required to be dependable. The computers are connected by a dependable hardware network which is the heart of the system.

In the same way of the hardware network, there is a software network, which interconnects all services, including software tasks running on the same computer, on different computers, FPGA programs and even hardware devices. This global software interconnection network is called 'Middleware'. The middleware is implemented in both CPU-software and FPGA-software. Both implementations use the same communication protocol. This allows to have only one interface type in the whole system: the Middleware Interface. This way, Instead of having many different interfaces, there is only one interface for all communications in the system.

A brilliant idea was to unify software and hardware in an integrated architecture [3]. Doing this, the architecture of the new core avionics system consists of computing nodes, mass memory, a diversity of different sensors and actuators and a dependable multicast middleware switch. All computing nodes, the mass-memory and the peripheral devices are connected to each other via this middleware switch.

C. Architecture

The middleware switch is in fact a publish/subscribe multicast bus. Each port provides a protocol translation layer to connect a device with its own interface and protocol to the middleware bus.

The capabilities of the FPGA emerging technology allows to implement middleware functionality directly in the hardware. The I/O interface - traditionally a UART - will then have in one side the required device interface and in the other side, it will be directly integrated to the middleware protocol.

The Switch is designed to route messages from different senders to several receivers. Up to 32 I/O ports (in the current implementation) in the Middleware Switch enable the switch to deal with many sender and receiver channels. The I/O ports can support different protocols such as JTAG, RS232, USB etc. Basically, it is a Publish/Subscribe multicast switch to route messages from a sender channel to as many receivers that are eligible to get data according to their subscriber list being compared with the Topics list in the message packet.

The message format designed for this architecture is shown in Fig. 1. The maximum length considered for the packet is 2KB. It starts with a 9-bit BOM field that refers to the beginning of message. It continues with two other 9-bit fields containing the list of Topics. Topics are special fields to be compared with the receivers' subscribers list for delivering data to the relevant receiver(s).

The *SenderID* field is to recognize different senders. The next field contains data to be sent through the switch. The data length can be as much that the whole packet does not exceed 2KB. The last field is called EOM representing the end of the message.

The *BOM* and *EOM* fields have a separate fixed 9-bit format. The first bit in each field (c/d) determines if the message is a command or contains data. This criteria was designed for packet synchronization. Each message can be recognized with its *BOM* and *EOM* fields in the beginning and end of the message respectively.

BOM	TopicH	TopicL	SenderID	Data	EOM
c/d1011xxxx	9-bit	9-bit	9-bit	~	c/d1110xxxx

Figure 1: Message format of the MWS core

[1] Project Compact Satellite

Fig. 2 shows an abstract block diagram of the MWS core.

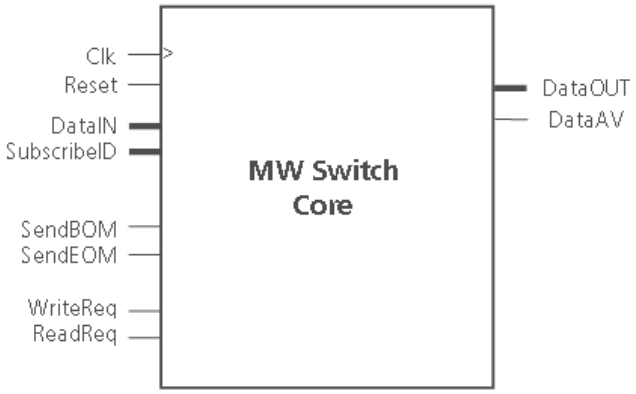

Figure 2: MWS core

DataIn is a 9-bit input port to cover each 9-bit message filed.

The MWS core consists of three main modules; Sender, Arbiter and Receiver modules (Fig. 3).

Each module is explained as follows:

1) Sender Module

The Sender module (Fig. 4) is to get data from an input port and deliver to the receiver(s) after checking if data is valid. As different senders may be connected to the Middleware Switch, the Arbiter module determines the sender to be granted the bus. Different senders are recognized with their *SenderID* field in the messages.

The Sender module contains different units. These units check message availability, store data, request to get the data bus from the Arbiter and read out data in case bus was given to that Sender.

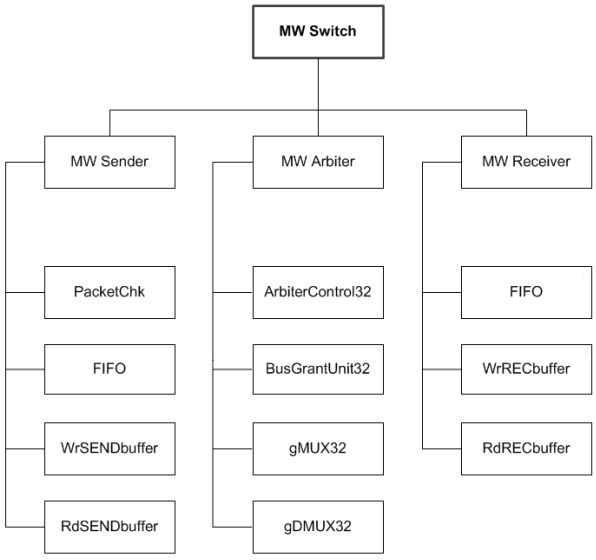

Figure 3: Design tree of the MW Switch core

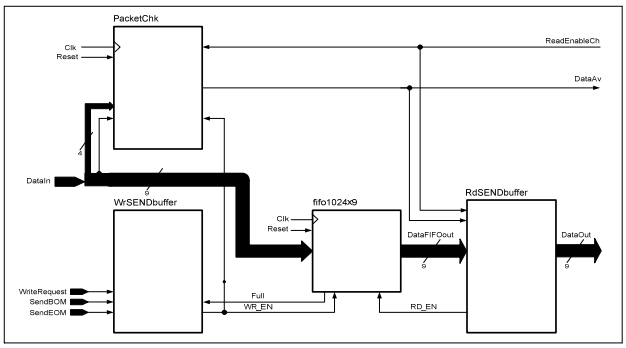

Figure 4: Sender module of the MWS core

2) Arbiter module

The Arbiter module (Fig. 5) grants the bus to different senders based on a round-robin scheme. Senders apply for getting the bus with setting their Data Available (*DataAv*) signal. Then the Arbiter decides constantly to give the bus to the senders one by one.

After a selected sender puts data on the bus, the Arbiter module distributes data among all receivers. That's why this switch is in the multicast switch categories. In the Receiver module, only those receivers that already have subscribed to receive data, will get the received messages. The desired data is in the receivers Topics list.

Fig .5 shows the architecture of the Arbiter module of the MWS core.

3) Receiver Module

All Receiver modules receive the same messages from the Sender module. They check if the Topic fields of the received messages match their Subscriber list, containing topics they have subscribed before. If so, they put the messages into their corresponding FIFO. Messages are read out when the Read Enable (*RD_EN*) signal is set. Fig. 6 presents the Receiver module of the MWS core.

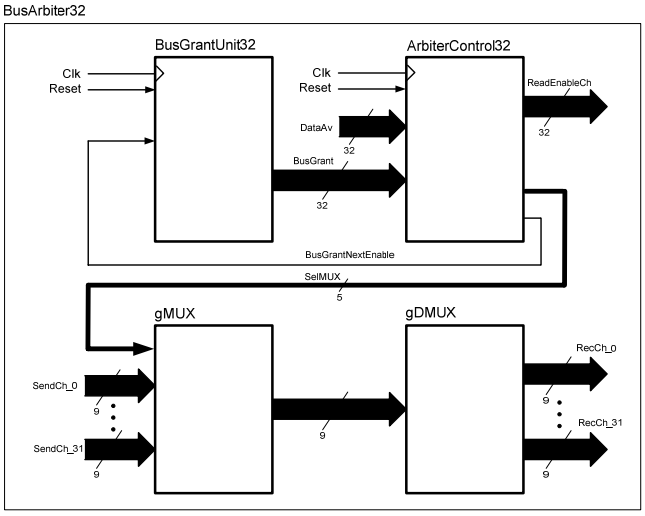

Figure 5: Arbiter module of the MWS core

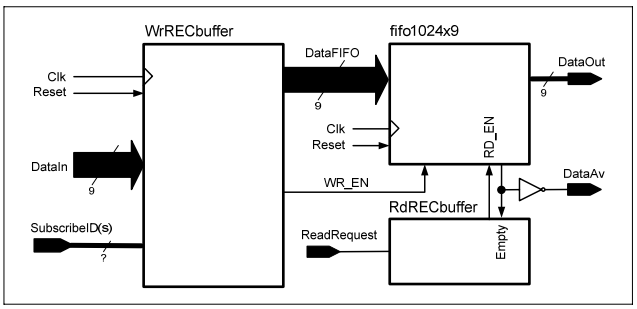

Figure 6: Receiver module of the MWS core

D. Functionality

The MW switch, in its current implementation supports up to 32 senders/receivers. Although each time only one sender is granted the bus, messages are published among all the receivers. The switch acts in a Publish/Subscribe manner. It means that the receivers may accept or ignore the arrived messages according to their interest to the received message. This interest is based on their prior subscription to receive desired messages. These message topics are stored in an exclusive Topics List in each sender. When a receiver receives a message, it compares the Topics field of the message with its SubscriberID list. The receiver accepts a message only if a match occurs in this comparison.

For a detailed functionality overview, let's assume that a message packet arrives in the *DataIn* port of the Sender. In each clock cycle, a 9-bit data enters the Sender module. The *PacketChk* unit of the Sender module checks if the data is ready and if so, sets *DataAv* signal. When *WriteRequest* signal is on, *WrSENDbuffer* unit generates *Wr_EN* signal to let data go into the FIFO and be queued there.

DataAv signal set by *PacketChk* together with *ReadEnableCh* signal coming from the Arbiter module, cause *RdSENDbuffer* unit, send out *RD_EN* signal to let data inside FIFO, come out and pass through the *RdSENDbuffer* unit to go to the next module.

The next module is *BusArbiter32* – a unit to grant the bus to one of the senders according to a round-robin scheme. Up to 32 senders are able to request for the bus. The bus is given to a sender if its *DataAv* signal is set. This happens with a counter in *BusGrantUnit32* unit in collaboration with a state machine in *ArbiterControl32* unit. When bus is granted to a sender, *ArbiterControl32* unit sets the relevant *ReadEnableCh* signal for that sender. This signal is the one that goes to the *RdSENDbuffer* unit in Sender module mentioned before, to let data come out of that Sender.

As only one sender's data is to be let go through the Arbiter module to the relevant receiver, data coming from the eligible sender should be selected. This comes true via the *gMUX* unit. It allows only data coming from one sender determined by the *ArbiterControl32* unit, to be distributed among the receivers. *gDMUX* is responsible for distribution of the selected data into the receivers.

All receivers receive data and compare it with a list of topics (at the moment is a fixed 10 numbers of 16-bit topics). If data matches with one of these topics, it is saved in the FIFO. The saved data could be sent out when the *RD_EN* signal is set by the *RdRECbuffer* unit. *RdRECbuffer* unit generates this signal when its *ReadRequest* input signal is set. Also a *DataAv* signal comes out of the receiver when data is ready in the FIFO to go out.

The middleware switch, at its first implementation is fully implemented on an FPGA. SRAM-based FPGAs are very susceptible to SEUs in space applications. This is discussed in more detail in section III. Robustness is a key to success in space missions therefore need for fault-tolerant design is inevitable. Section IV is dedicated to the fault-tolerant implementation of the MWS core.

III. RADIATION EFEFCTS ON FPGAS

As the MW Switch implemented in FPGAs will be used in satellite systems, a special care on radiation effects in space should be emphasised.

Electronic components can be damaged in the space environment through single-event effects (SEE). Space environment possesses high energy electrons, protons and heavy ions due to cosmic rays and other natural radiation resources. These particles may cause damaging effects on electronic components.

Radiation in space environment makes space applications different than terrestrial systems. Radiation can cause bit flips (upsets) in memory modules and hence failure in semiconductor devices. Therefore high-reliability plays an important role in such systems.

Reprogrammable logic chips do support reconfiguration feature. The use of reconfiguration in space applications allows designers to apply changes and update on-board hardware by replacing faulty designs with correct data. Moreover, it is useful to take advantage of developing and debugging the hardware on ordinary FPGAs and applying necessary modifications in the design as much as needed.

Re-programmable devices make it possible to implement different tasks in a single chip instead of putting them into different dedicated parts. This feature causes a reduction of overall system power as well as the area. This high level of integration and flexibility shows the high potential of reprogrammable FPGAs and makes them very suitable for space applications.

A. Radiation and SEUs

Logic circuits in turbulent environments may encounter the following problems [4]:

- Single-Event Upsets (SEU) or bit flips in memory or flip-flops, known as soft errors as well. If SEUs put the circuit into an undefined state, a Single-Event Functional Interrupts (SEFI) may happen.
- Single-Event Latchup (SEL) which is shorted junctions inside a semiconductor.
- Single-Event Transient (SET) is a transient signal inside the circuit.

This paper covers only SEU effects on the MWS implemented on an FPGA. FPGAs as semiconductor devices are very susceptible to Single Event Upsets. There are different approaches to handle SEUs in digital logic. As mentioned above, SEUs affect the registers and memory

elements. The paper addresses the sequential logic parts of the MWS core to make these parts tolerable against Single Event Upsets.

The functionality information of an FPGA is stored in memory cells. Also the internal connections of the FPGA are based on data stored in SRAM cells or so called Look-Up Tables of FPGAs. An upset in these memory cells could cause the device to malfunction. For this reason SEUs are a major cause of concern.

The probability of having multiple errors within one clock cycle is low [1] thus considering only single events does not hurt the reliability issue in the targeted MWS core. Here follows an overview of SEU mitigation approaches for sequential logic on FPGAs.

B. SEU Mittigation Techniques

Mitigation techniques against SEUs on FPGAs could be applied in physical level, system level and logic level.
Physical techniques include several methods such as shielding the package against radiation, choosing special substrate which gives higher tolerance to faults or special considerations in manufacturing step.

Highly dependable space systems are good examples for applying radiation protection with design techniques where the existing protection in physical level is not enough. A dependable design should accept SEUs, correct them and reconfigure the affected circuit part automatically. All these activities should happen without affecting the behavior of the overall system.

For protection in system level, a common SEU mitigation technique is to apply Triple Modular Redundancy (TMR). TMR brings tolerance against faults using redundant components in system level together with a voter circuit to perform votes among redundant systems. This concept can also be implemented by replicating the logic in the design. A single bit may be replaced with three bits and a separate voting logic for each bit to determine its result in each clock cycle. Furthermore, error detection and correction codes (EDACs) can be used to check for errors in memory modules. Reading out the data, checking for errors and writing back corrected data into memory can be another suitable method to prevent damages in memory contents.

In logic level TMR approach, each flip-flop is replicated three times and voted by a majority voter to determine the true state of that flip-flop. TMR can be applied to a complete design or even to part of it.

A thorough overview about applying TMR method on FPGAs is presented in [2]. The problem with this approach is its more power consumption due to redundancy together with its impacts on the circuit timing. Moreover, it only covers the registers and does not cover multiple upsets protection. Furthermore, the effects of SEUs are not limited to flip-flops. Combinatorial logic is also sensible to SEUs for which there are several protection schemes proposed.

Several EDAC methods are also available that can be used for SEU protection as a complement to TMR. These methods are specially useful to protect on-chip memory modules from probable upsets.

IV. FAULT-TOLERANT MIDDLEWARE SWITCH

This section is about fault-tolerant implementation of the embedded Middleware Switch core on an FPGA. In section II, the architecture and functionality of the MWS core was explained in detail. As mentioned before, fault-tolerance plays an important role in space applications. The Middleware Switch core is supposed to be radiation-hardened so that it keeps working properly even in case SEUs would happen. The following parts cover the methods employed to achieve this goal.

Fault-tolerance in general is to make the system to operate satisfactorily in erroneous conditions. In case of space systems, the erroneous conditions arise due to the radiation and is ought to be mitigated in a right way. An important point here is to focus on the ‘right way’ statement to achieve this goal. By this statement, we mean to determine the level of fault-tolerance needed for the system as well as a suitable technique to apply fault-tolerance to the system.

The degree of fault-tolerance is determined by the system requirements [1]. It is the expected behaviour of the system upon presence of faults that defines this level. For instance, it should be cleared that if the errors in the system, are only to be detected or corrected as well. Or for another example, it should be noted that if the system is susceptible to more than one error per clock cycle. This kind of questions, refers directly to the system specifications. While considering these specifications, it is possible to decide on the degree of fault-tolerance that the system needs to be equipped.

After determining the required level of fault-tolerance, it’s time to find a suitable tolerance technique against faults. This technique is correlated to the corresponding level of fault-tolerance.

Soft-error mitigation techniques usually address only the latches within the circuit. For the MWS core, a thrifty fault-tolerance method is applied. The thrifty fault-tolerant method targets only the parts with major concerns to be mitigated against SEUs. These major concerns are the memory modules, FFs and consequently FSMs.

The thrifty fault-tolerance is different than that discussed in [17]. Here, thrifty means the tolerance has been applied only to the parts with major concerns saving used resources and area, causing minimal timing effects and offering a reasonable level of fault-tolerance.

The first implementation of the MWS core is fully on an FPGA. In SRAM-based FPGAs the routing information is saved in the configuration memory of the FPGAs. To make sure that there is no more than one upset in the FPGA at a given time and correct it, configuration memory scrubbing mechanism is employed [5]. This mechanism is described in section IV part B.

The only remaining part is I/O ports. Considering that the MW Switch core is designed as a soft core, to be connected to other cores such as protocol converters, the I/O ports are not of too much concern. Applying fault-tolerant techniques to the selected modules mentioned above together with configuration memory scrubbing as a complement provides a reasonable level of fault-tolerance for the MWS core with this thrifty approach.

A. *Memory Modules*

One of the parts to be mitigated against SEUs is Memory modules. The common mitigation technique for SEUs in RAMs is by means of error detection and correction codes.

Error detection and correction is applied to many highly-reliable and high- performance applications. A suitable algorithm for error detection and correction is Hamming code. It detects double bit errors and corrects single bit errors anywhere within the system.

B. *FSMs*

State machines are generally used as controllers in logic designs. A state machine constitutes of several FFs to hold its current state value. This registered value is then fed back into a prior state and forms a registered logic loop to control the sequence of digital logic. Therefore if there would be a fault in the current state, the next state would be faulty too. Such a sequential logic is to be replicated for fault-tolerance. In the thriFTy method applied to the MWS core, replication of the FSMs is managed with TMR method.

As mentioned above, state machines are typically used in controller units of logic designs. In case an error happens in a state machine the whole design may malfunction.

1) *TMR and State Encoding*

To implement TMR in state machines, an explicit encoding scheme is required (Fig. 7). The type of encoding determines the susceptibility of the state machine to radiation [12].

The idea behind the state machine encoding is to assign codes to the states which are represented symbolically, to be able to present them in a register. This way, each state is recognized with its assigned code. Researches [12] show that Hamming-3 encoding has the best tolerance against faults, showing no errors in fault injection tests. In Hamming-3 encoding, states are different by 3 bits. Thus, three bits should be changed in any state in order for the state machine, to malfunction. But in change, it requires the most resources, and is the slowest compared to other common encoding methods. Hamming-2 encoding (states are different in 2 bits) has less errors than binary or one-hot encodings that are two other alternatives (Fig. 7).

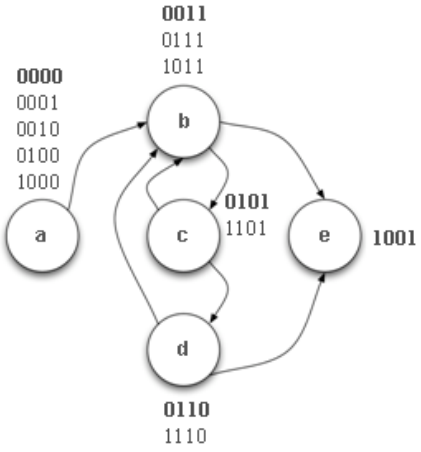

Figure 7: Sample state machine with Hamming-2 state encoding

Hamming-2 seems to be the best choice for fault tolerant designs in terms of size and speed [12]. Hamming-3 encoding could be considered for those applications that require very high reliability. For state machines with a large number of states, a Hamming-n state encoding forces a big redundancy to the switch core. Following the thrifty policy of the thriFTy method, for mitigation of the state machines, Triple Module Redundancy (TMR) method with One-hot state encoding is preferred.

2) *Automated TMR*

The basic concept of TMR is that a fault-sensitive component can be hardened to SEUs by implementing three copies of the same component and performing a bit-wise majority voter on the output of the triplicate circuit. This component can be a single flip-flop or a logic circuit. The function of the majority voter is to output the logic value that corresponds to the majority (at least two) of its inputs.

Several projects have shown that implementing TMR within FPGAs improve reliability. For example, TMR was used on LEON3-FT [6]. A detailed description on using TMR within the FPGAs can be found in [7].

Significant improvements in design reliability by applying TMR have encouraged vendors to develop several tools for automating the process. Xilinx TMRTool and Gaisler's FTMR are of those cases [8][9]. Although effectiveness of TMR circuits produced by these tools has been verified in radiation and with fault injection [10], there are some discussions on easiness or generalization of using these tools to design fault-tolerant circuits. Gaisler's Research technical report [9] states that FTMR's increase in on-chip resource usage for protection is a factor of between 4,5 and 7,5 for the demonstration application. Moreover, a performance decrease of about 50%, could limit the usability of that method.

3) *TMR and Configuration Memory*

Proper functionality of TMR in FPGAs depends on the fact that there should be no more than one upset in the configuration memory of an FPGA at each clock cycle. More than one upset may cause the majority voters to malfunction. To avoid this, configuration memory scrubbing is employed to periodically remove upsets stemmed from the radiation environment. Scrubbing is repeatedly correcting upsets in the configuration memory of an FPGA. Several methods have been suggested for doing this [11]. If scrubbing process would be done fast enough it can ensure that there would be no more than one upset in the FPGA's configuration memory at a given time.

4) *TMR and the Synthesis Problem*

After validation of the design behaviour, it should be turned into an implementation in terms of logic gates. This process is called Logic Synthesis. One important aspect of synthesis tools is logic optimization in words of area and timing. To optimize the design, synthesis algorithms may remove the replicated parts added into the design for fault-tolerance. Therefore, the purpose is to ensure that intentional replications (result of tripled FFs) are not removed by the synthesis tool in optimization process.

For this, one way is to apply necessary directives in the synthesis tool for preserving redundant logic. Although this seems to work, the MWS core is to be designed as a soft core. That means it should be independent of the synthesis tools to be synthesized in any FPGA or ASIC in future. To achieve this goal, the state variable of the FFs were put into a(n) array/vector with three cells to accommodate three redundant copies of the same value as required for TMR (Fig. 8).
This way, the synthesis tool considers each cell of the array as a separate state and does not realize the repeated states as redundant values.

5) TMR Voter Implementation

To implement a majority voter in FPGAs, there are two options. One way is to use Look-Up Tables (LUTs) that are logic resources to implement any boolean function inside FPGAs. An alternative is dedicated hardware resources available in FPGAs. For instance, Xilinx offers internal 3-state buffers in its Virtex series (Virtex library primitive BUFT). These dedicated resources are useful when the available logic resources are limited. Implementing the voter logic by means of dedicated resources saves area used in the chip. For this reason, in the MW Switch core, the dedicated hardware resources were chosen. The structure of a majority voter circuit using the BUFT library primitive is presented in [7].

C. Routing and Data Path

In FPGAs, logic paths are not hard-wired as in ASICs. Data path and routing information are stored in configuration memory of FPGAs. Configuration Memory Scrubbing ensures that there will not be any SEU in the configuration memory of an FPGA, so routing information are kept safe and there is no need to apply replication to routing and data path in the thrifty fault-tolerant approach applied to the MWS core. During scrubbing, the design is not interrupted. After scrubbing, a Readback process is recommended to immediately follow to ensure that SEUs were corrected in each clock cycle. Readback and scrubbing are mechanisms to detect and correct SEUs in the configuration memory of FPGAs without interrupting their operations.

Instead of replicating the routing paths, the thriFTy method suggests the Configuration Memory Scrubbing for SEU mitigation of the data path information in the MWS core. Since the Configuration Memory Scrubbing does not interrupt the functionality of the switch, timing effects are not matter of concern. [14] and [15] provide necessary information regarding configuration and readback operations in a sample Xilinx Virtex-4 device.

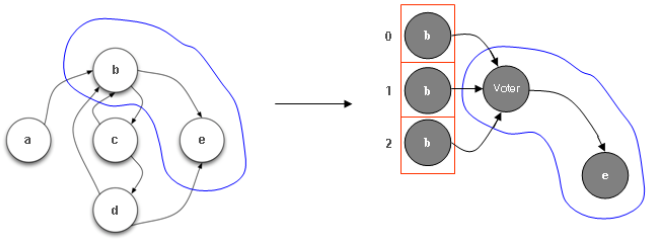

Figure 8: Solution for the synthesis problem in the state replication

V. SIMULATION AND TEST

There are several techniques to evaluate SEU mitigation approaches in FPGA designs. Common techniques are Radiation Test , Fault Injection and Simulation of faults in different subcomponents as well as the whole system.
Radiation Test is an accurate way of determining SEU sensitivity of a design. In this technique, high-energy particles are applied to the design to measure sensitivity of the device to SEUs. Although this is an accurate method, it is very expensive and time consuming too.

Another way of measuring SEU sensitivity is to inject artificial upsets into different parts of the design. Studies show a comparable result with radiation tests [16]. However, it can not simulate upsets to the configuration logic itself [13].

For the MW Switch core, a complete radiation test has been planned before making it operational. Up to then, the design is to be validated to ensure if the system specifications are met. In the absence of faults, the synthesized fault-tolerant core should behave similar to the non-fault-tolerant one. Besides this, the behaviour of the Switch core against intentional faults fed to different parts was simulated and studied very carefully. Simulation results showed that the fault-tolerant core is capable of handling faults in different situations. Moreover, different post-synthesis I/O tests have successfully proved the system functionality, achieving expected results out of the applied inputs.

Development of the MW Switch core is still ongoing, adding further capabilities such as covering different peripherals through a special translation layer. The FT concept together with the introduced methods remains the same and applicable to any further parts being added during the whole development phase.

VI. RESULTS

Table I. provides a comparison between the FT and Non-FT Switch core in terms of resource usage and maximum frequency of the whole system predicted by the synthesizer. The synthesis results shown in Table I. were achieved using the Xilinx ISE development software. The prototype design was implemented on a Xilinx Virtex-4 FX12 FPGA on ML403 development board.

Furthermore, power consumption is a major issue for space systems and should be carefully analyzed.

TABLE I. COMPARISON OF FT AND NON-FT MWS CORE

Logic Utilization	Available	Non-FT version	FT version	Change
Slice FFs	10944	1184	1663	+40.4%
LUTs	10944	2238	2809	+25.5%
FIFOs	36	16	16	0%
Max. freq	--	101.890MHz	73.659MHz	- 27.7%

Table II. represents increase of power consumption after applying fault-tolerance to the MWS core. It is necessary to note that, power consumption analysis depends on the different operation modes of a system. The results in Table II. are based on considering active operating mode for the most of the life cycle of the design (worst case). Lower time considerations for the active operating mode gives less increase in power consumption compared to the worst case conditions.

TABLE II. POWER CONSUMPTION IN FT AND NON-FT MWS CORE

Power Consumption	Non-FT version	FT version	Change
Static	12.72 mW	12.72 mW	0 mW
Dynamic	37.538 mW	90.118 mW	52.58 mW
Total	50.258 mW	102.838 mW	52.58 mW
Battery Life[1]	34.313 h	16.435 h	-18.878 h

Compared to the full TMR method that in some cases, causes a performance decrease of 50% [9], the fault-tolerant MWS core, represents more reasonable results considering resources usage and maximum frequency of the design.

REFERENCES

[1] Melanie Berg, A Simplified Approach to Fault Tolerant State Machine, Design for Single Event Upsets, Ball Aerospace & Technologies Corp.

[2] R. Katz et al., SEU Hardening of Field Programmable Gate Arrays (FPGAs) For Space Applications and Device Characterization, 31st Annual Nuclear and Space Radiation Effects Conference, 1994 NSREC, Tucson, USA

[3] Sergio Montenegro, Jan-Thimo Grundmann, Bobby Kazeminejad, Peter Spietz, The new DLR Standard Satellite Bus series (SSB), Small Satellites Systems and Services - The 4S Symposium, German Aerospace Center, 2008

[4] Single Event Effects in Avionics, Boing Radiations Effect Lab., a presentation by E. Normand to C17 Avionics group, Dec 1998

[5] Keith S. Morgan, Daniel L. McMurtrey, Brian H. Pratt, and Michael J. Wirthlin, Fault-Tolerant Design Techniques for FPGAs, IEEE transactions on nuclear science, 2007

[6] http://www.gaisler.com

[7] C. Carmichael, Triple Module Redundancy Design Techniques for Virtex FPGA, " Tech. Rep. Xilinx Corporation, 2001, vol. 1.0, xAPP197

[8] Xilinx XTMR Tool Available at: http://www.xilinx.com

[9] Sandi Habinc, Functional Triple Modular Redundancy (FTMR), Design and Assessment Report, Gaisler Research, 2002

[10] Keith S. Morgan, Daniel L. McMurtrey, Brian H. Pratt, and Michael J. Wirthlin, A Comparison of TMR With Alternative Fault-Tolerant Design Techniques for FPGAs, IEEE transactions on nuclear science, 2007

[11] C. Carmichael, M. Caffrey, and A. Salazar, Correcting Single-Event Upsets Through Virtex Partial Configuration, Tech. Rep. Xilinx Corporation, 2000, vol. 1.0, xAPP216

[12] Gary Burke and Stephanie Taft, Fault Tolerant State Machines, Jet Propulsion Laboratory

[13] Carl Carmichael, and Chen Wei Tseng, Correcting Single-Event Upsets in Virtex-4 Platform FPGA Configuration Memory, Xilinx Application Note XAPP988

[14] Virtex-4 FPGA Configuration User Guide UG071, Xilinx Corporation

[15] Brendan Bridgford, Carl Carmichael, and Chen Wei Tseng, Single-Event Upset Mitigation Selection Guide

[16] M. Rebaudengo, M. Sonza Reorda, M. Violante, Simulation-Based Analysis of SEU Effects on SRAM-based FPGAs, FPL2002, International Conference on Field Programmable Logic and Application, 2002

[17] Praveen Kumar Samudrala, Jeremy Ramos, and Srinivas Katkoori, Selective Triple Modular Redundancy for SEU Mitigation in FPGAs, University of South Florida, Tampa, FL-Honeywell Space Systems Inc., Clearwater, FL

[1] This battery life was considered for a sample battery capacity of 1000 mAh