A Fault Injection Analysis of Virtex FPGA TMR Design Methodology

F. Lima[1,2], C. Carmichael[1], J. Fabula[1], R. Padovani[1], R. Reis[2], Member IEEE

?

Abstract— This paper presents the meaningful results of a single bit upset fault injection analysis performed in Virtex FPGA Triple Modular Redundancy (TMR) design. Each programmable bit upset able to cause an error in the TMR design has been investigated. Final conclusion using the TMR "golden" comparison method shows that "no errors" were reported by Virtex TMR design implementation in the presence of single bit upsets in the customization logic. The proton radiation ground test has confirmed the results achieved by fault injection.

Index Terms—Field Programmable Gate Arrays, Virtex, fault injection, triple modular redundancy.

I. Introduction

Field Programmable Gate Arrays (FPGAs) are increasingly demanded by spacecraft electronic designers due to their high flexibility to meet multiple requirements such as high performance, attractive cost, no NRE (Non Refundable Engineering) and fast turnaround time. Moreover, because FPGAs are reprogrammable, they offer the additional benefits of allowing on-orbit design changes. Data can be sent after launch to correct errors or to improve system performance.

The Virtex family [1] is a high performance SRAM-based FPGA that supports a wide range of configurable gates from 50k to 1M. The Virtex QPRO family [2] provides an off-the-shelf system-level solution for aerospace and defense customers. It is fabricated on thin-epitaxial silicon wafers using the commercial mask set and the Xilinx 5-layer-metal 0.22 μm CMOS process. The use of epitaxial CMOS process technology has made Virtex Single Event latch-up immune (LET_{th} >120 MeV*cm^2/mg, TID=100Krads(si)).

In order to mitigate Single Event Upsets (SEU) in Virtex FPGAs, the Triple Modular Redundancy (TMR) with voting technique combining with bitstream scrubbing must be applied [3, 4]. The TMR mitigation scheme uses three identical logic circuits performing the same task in tandem with corresponding outputs being compared through a majority vote circuit. If an upset occurs in the customized logic, TMR votes out the error. If the upset occurs in the device configuration, TMR votes out the logic part no longer functioning and reconfiguration (scrubbing) repairs the error before more errors accumulate and overcome the TMR. Bitstream reconfiguration, complete or partial, may occur without interruption of service in the Virtex device.

Previous results from the radiation ground testing presented at [5] showed that using TMR in Virtex FPGAs, the cross section was reduced by 1,000 times compared to using only the scrubbing technique without TMR. But it was still not zero. This investigation started with the objective to justify the errors obtained from the ground testing experiments in the Virtex TMR design. The analysis must explain how a single bit upset in the bitstream of the Virtex FPGA could cause two errors in distinctly redundant logic parts of the TMR design.

This paper presents the complete fault injection analysis of Virtex FPGA TMR design, the error design detection and correction and the final fault injection results showing the high-reliability of the TMR solution. The proton radiation ground test has confirmed the results achieved by fault injection. This work also introduces a more complex TMR design in order to continue evaluating the techniques addressed in [3, 4]. A TMR 8051-like micro-controller was designed and tested by the fault injection tool. Preliminary results from the radiation test of the TMR 8051-like micro-controller are presented in this paper.

II. Triple Modular Redundancy in Virtex FPGA

Aiming at giving a better comprehension of the fault injection process and the error analysis, a brief overview of the Virtex architecture, the device configuration bitstream and the TMR techniques are presented in this section. All topics are explained in more details in the application notes referred in this paper.

A. Virtex Architecture Overview

Virtex devices consist of a flexible and regular architecture composed of an array of configurable logic blocks (CLBs) surrounded by programmable input/output blocks (IOBs), all interconnected by a hierarchy of fast and versatile routing resources, Fig. 1 [1]. The CLBs provide the functional elements for constructing logic while the IOBs provide the interface between the package pins and the CLBs. The CLBs are interconnected through a general routing matrix (GRM) that comprises an array of routing switches located at the intersections of horizontal and vertical routing channel. The Virtex matrix also has dedicated memory blocks called Block SelectRAMs of 4096 bits each, clock DLLs for clock-distribution delay compensation and clock domain control, and two 3-State buffers (BUFTs) associated with each CLB.

The authors are adjuncts to 1 – Xilinx, Inc. (USA) and to 2 – Federal University of Rio Grande do Sul (Brazil).

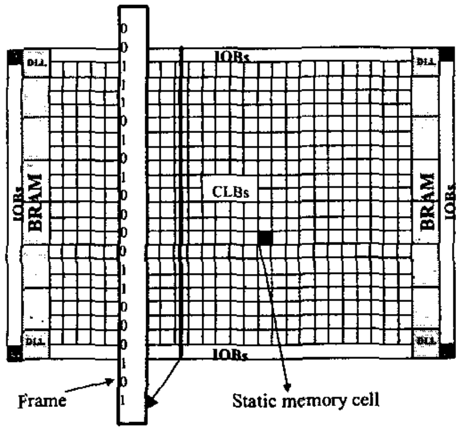


Fig. 1. Virtex Architecture Overview

Virtex devices are quickly programmed by loading a configuration bitstream (collection of configuration bits) into the device. The device functionality can be changed at anytime by loading in a new bitstream. The bitstream is divided into frames and it contains all the information to configure the programmable storage elements in the matrix located in the Look-up tables (LUT) and flip-flops, CLBs configuration cells and interconnections, Fig. 2. All these configuration bits are potentially sensitive to SEU and consequently they were our investigation targets.

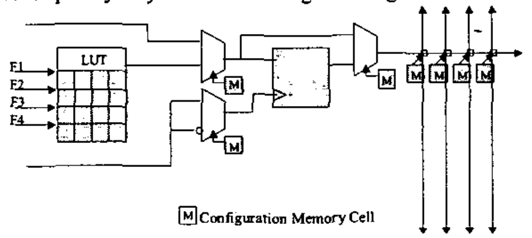


Fig. 2. SEU Sensitive Bits in the CLB Tile Schematic

B. *Virtex Configuration Overview*

The Virtex configuration memory can be visualized as a rectangular array of bits. The bits are grouped into vertical frames that are one-bit wide and extend from the top to the bottom of the array composing a column defined by a major address [6]. Each matrix column is associated to a major address and to a different number of frames according to the nature of the column, shown in table I.

The frames are read and written sequentially with ascending addresses for each operation. The frame size depends on the number of rows in the device. The number of configuration bits in a frame is 18 x (# of CLB rows +2) and is padded with zeros on the right (bottom) to fit a 32-bit word.

The frame organization differs for each type of column. Each frame is located vertically in the device with the front of the frame at the top. Fig. 3(a), 3(b) and 3(c) show the CLB column frame, IOB column frame and Block SelectRAM content organization, respectively. The frame top is showed on the left.

TABLE I - VIRTEX CONFIGURATION COLUMN TYPE

Column Type	# of frames	# per device
center	8	1
CLB	48	# CLB columns
IOB	54	2
Block SelectRAM interconnect	27	# of blocks SelectRAM columns
Block SelectRAM content	64	# of blocks SelectRAM columns

Top 2 IOB	CLB R1	...	CLB Rn	Bottom 2 IOB
18 bits	18 bits	...	18 bits	18 bits
(a) CLB column frame				

Top 3 IOB	3 IOBs	...	3 IOBs	Bottom 3 IOB
18 bits	18 bits	...	18 bits	18 bits
(b) IOB column frame				

PAD	RAM R0	...	RAM RN	PAD
18 bits	72 bits	...	72 bits	18 bits
(c) Block SelectRAM content column frame				

Fig. 3. Frame Organization

The CLB tile is composed of the CLB logic and the surrounding interconnection placed in a determined row and column in the matrix. There are 864 customization bits per CLB tile distributed in 48 frames with 18 bits each, Fig. 4. The bits can be divided in Look-up table bits (7.4%), CLB configuration bits (6.8%), interconnection bits (84.2%) and 3-state buffer configuration bits (1.6%).

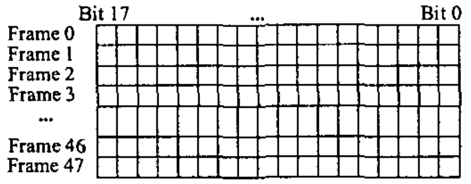


Fig. 4. CLB Tile Map

C. *Virtex TMR Techniques Overview*

The correct implementation of TMR circuitry within the Virtex architecture depends on the type of data structure to be mitigated. The logic may be grouped into four different structure types: Throughput Logic, State-machine Logic, I/O Logic, and Special Features (SelectRAM block, DLLs, etc.). The TMR technique for Virtex is presented in details in [4].

1) Throughput Logic

Throughput logic is a logic module of any size or functionality, synchronous or asynchronous, where all of the logic paths flow from the inputs to the outputs of the module without ever forming a logic loop. In this case, it is necessary to just triplicate the logic, creating three redundant logic parts (0,1 and 2). No voters are required, as the FPGA output will be by default voted later.

2) State Machine

State-machine logic is any structure where a registered output, at any register stage within the module, is fed back into any prior stage within the module, forming a registered logic loop. This structure is used in accumulators, counters, or any custom state-machine or state-sequencer where the given state of the internal registers is dependent on its own previous state. In this case, it is necessary to triplicate the logic and to have majority voters in the outputs. The register can not be locked in a wrong value, for this reason there is a voter for each redundant logic part in the feedback path making the system be able to recover by itself. Fig. 5 shows a general example of this structure.

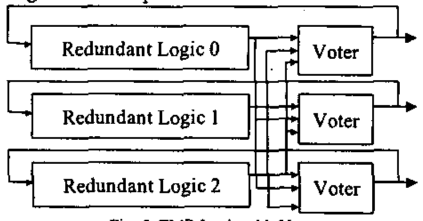


Fig. 5. TMR Logic with Voter

The majority voter, Fig. 6, can be easily implemented by one LUT. For designs constrained by available logic resources, the majority voter can be implemented using the Virtex 3-state buffers instead of LUTs. There are two 3-state buffers per CLB.

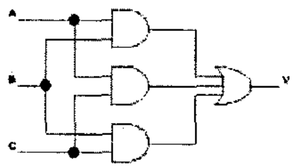


Fig. 6. Three input Majority Voter Schematic [4]

3) I/O Logic

The primary purpose of using a TMR design methodology is to remove all single points of failure from the design. This begins with the FPGA inputs. If a single input was connected to all three redundant logic parts within the FPGA then a failure at that input would cause these errors to propagate through all the redundancies and thus the error would not be mitigated. Therefore, each redundant part of the design that uses FPGA inputs should have its own set of inputs. Thus, if one of the inputs suffers a failure it will only affect one of the redundant logic parts. The outputs are the key to the overall TMR strategy. Since the full triple module redundancy generates every logic path in triplicate there must ultimately be a method for bringing these triple logic paths back to a single path that does not create a single point of failure. This can be accomplished with TMR outputs voters inside the output logic block [4].

4) Special Futures:

The Virtex architecture provides a number of special features, such as BlockRAM, DLLs, etc, which require specialized methods for implementing effective redundancy. A reliable method to TMR the BlockRAM is to constantly refresh the BlockRAM contents, Fig. 7. Since these are dual port memories, one of the ports could be dedicated to error detection and correction. But this also means that the BlockRAM could only be used as single port memories by the rest of the user logic. To refresh the memory contents a counter may be used to cycle through the memory addresses incrementing the address once every four clock cycles. For each address, the data content would be voted and the majority voted value written back into the cells.

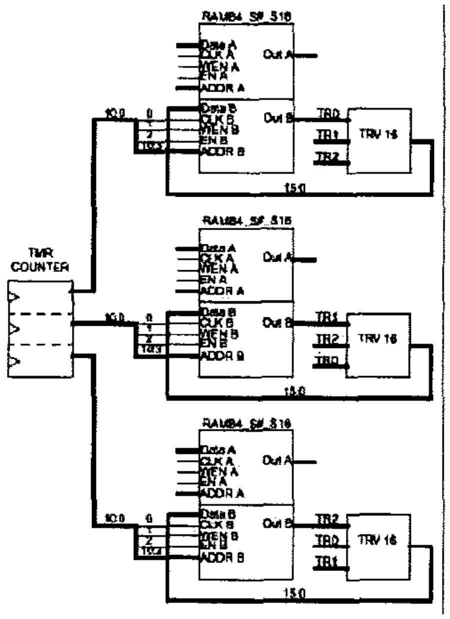


Fig. 7. BlockRAM TMR with Refreshing

5) Avoiding Persistent Errors

A typical FPGA design will be implemented with signals that were resolved to a logic constant (VCC or GND) but could not be entirely optimized out of the design. When the Place and Route (PAR) tools implement the VCC and GND signals, they are implemented in a way that maximizes device resource utilization. This is accomplished by utilizing "Keeper" circuits that exist at the input pins of all CLBs and IOBs. Keepers lie in series with a routing channels and logic block input pins. When the routing channel carries an active signal, the keeper is transparent. But when the channel is unused, the keeper will keep its last known value - which was determined when the device was initially powered-up or re-initialized by activating the FPGA input PROG. When a logic element (i.e. flip-flop) inside a logic block (i.e. CLB or IOB) requires a logical constant, such as a VCC or GND, this logical constant may be obtained from the keeper circuit of an unused pin of the logic block. Its polarity may be selected by programmable inversion within the logic block.

An SEU may upset, or alter, the state of a keeper circuit either by direct ionization, or indirectly by momentarily connecting an active routing channel to the input of the keeper. In either case, the result is a functional disturbance that cannot be detected by readback nor corrected by partial reconfiguration. Therefore, this type of error is known as a "persistent error". And it can only be corrected by completely re-initializing the FPGA. Schematic designers should be careful to examine the primitive implementation of all library macros that are likely to contain registers, before using them in their design. Even if the macro provides clock enable and reset pins at the top level, the primitive implementation may be different than expected. Similarly, if a VHDL user describes a synchronous process without specifying a clock-enable or initialization function, the synthesis tool will implement this function by using primitives and connecting all unused pins to the correct logical constant, thus creating VCC and GND. In order to avoid persistent errors, user VCCs, user GNDs and user clock enables for each redundant logic part must be created in the design as inputs.

6) Scrubbing

The scrubbing [4] allows a system to repair SEUs in the configuration memory without disrupting its operations. The scrubbing is performed through the Virtex? SelectMAP interface. When the FPGA is in this mode, an external oscillator generates the configuration clock that drives the PROM and the FPGA. At each clock cycle new data are available on the PROM data pins. One example is the Flash-PROM XQR18V04 that provides a parallel frequency up to 264 Mbps at 33 MHz.

The scrubbing cycle time depends on the configuration clock frequency and on the readback bitstream size. For the XQVR300, it is necessary to utilize 207,972 clock cycles in order to perform the full scrubbing load (Scrub cycle = # clock cycles x clock period). The scrubbing rate describes how often a scrubbing cycle must occur. It is determined by the expected upset rate of the device for the given application. Upset rates are calculated from the static bit cross-section of the device and the charged particle flux the application or mission is expected to endure. The scrubbing rate should be set such that any SEU on the configuration memory will be fixed before the next upset will occur. In reality the scrubbing rate is minimized to be equal to the scrubbing cycle. In this way configuration logic is always being refreshed. The implemented design can also have influence in the selection of the scrubbing rate. A good "rule of thumb" is to place the scrubbing rate one order of magnitude or more above the expected upset rate. In other words, the system should scrub, on the average, at least ten times between upsets.

III. Fault Injection Analysis

Fault injection is an attractive technique for the evaluation of design characteristics such as reliability, safety and fault coverage, due to its high flexibility in terms of spatial and temporal information. The process involves inserting faults into particular targets in a system at a determined time in the process and monitoring the results to determine its behavior in response to a fault. It has also a reduced turnaround time and evaluation cost compared to traditional radiation ground testing, helping designers in the development process of SEU hardened digital circuits.

The fault injection in SRAM-based FPGAs is defined as a bit flip in all bits of the configuration bitstream. In this way it is possible to evaluate the effects of an upset in all sensitive areas of the programmable matrix. Some of these bits are directly related to the user's design combinational and sequential logic and some of them are related to the FPGA architecture and design implementation.

The fault injection analysis was executed in four main steps. First, the fault injection tool developed by Los Alamos National Laboratory was used to catalogue all the configuration bit locations that caused a dynamic error in the TMR design. Then all the reported bits were identified in the general FPGA matrix in terms of row, column and functionality. Based on this information it was possible to identify those bits in the FPGA IC schematics. The third step identified the correlation between the bit location in the FPGA IC schematic and its location in the design under test in the FPGA editor tool. The last step was the characterization of the error.

A. Test Design Methodology

The TMR test design methodology used to analyze the SEU in the Virtex FPGA consists of a TMR counter design replicated in the circuit in order to fill the resources of the device (XQVR300). All of the CLBs were used to implement eight TMR 32-bit counters with pipeline design. The design can be divided in three groups: the redundant logic part 0, redundant logic part 1 and redundant logic part 2. Each redundant group is composed of eight 32-bit counters.

In order to detect an error in one of the 32-bit counters, the eight 32-bit counters located in the same redundant logic group are compared against each other. There is one comparator for each group. Comparators 0, 1 and 2 report an error in the redundant part 0, 1 and 2 respectively.

The three redundant logic groups are finally compared in the minority voter located in the output logic block. The error flag, a result of the minority voter, reports if there is an error in two or more redundant parts. A schematic of this approach is illustrated in Fig. 8.

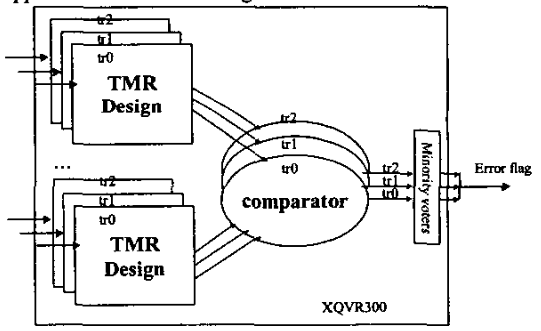


Fig. 8. TMR Design Methodology

B. Fault Injection Tool

The fault injection tool developed in Los Alamos National Laboratory is able to corrupt all the bits of Virtex bitstream in a sequential way, or individually by choosing a specific bit location. The objective of this tool is to analyze the effect of a single bit upset in a TMR design implemented in the Virtex architecture. All single bit upsets able to cause an error in the TMR design were cataloged for investigation.

In principle, no single bit upset in the bitstream should cause an error in the TMR design if a single upset error affects only one redundant part of the design. By TMR definition, if one redundant part is corrupted by an upset, the majority voters continue voting the correct value from the two other redundant uncorrupted parts.

The fault injection tool can upset a single bit in the bitstream sequentially starting from a user defined major address and frame, or it can upset one specific bit when the user defines the major address, frame, frame byte and bit. The fault injection is performed in three steps, presented in table II. The three-step method guarantees no double upsets for any short period of time.

TABLE II - VIRTEX CONFIGURATION COLUMN TYPE

Fault injection steps	Bitstream example
Read the bitstream:	... 0110010101010...
Corrupt one bit and load the bitstream:	... 1110010101010...
Correct the previous bit and load the bitstream:	... 0110010101010...
Reset the flip-flops	... 0110010101010...

Each time an error is reported by the test design comparator, the fault injection tool shows the location of the upset bit that caused the error. The tool reports the major address, frame, frame byte and bit location of the bit. Using this information it is possible to know exactly the location of the bit in the bitstream and as consequence in the FPGA matrix.

The column bit information is obtained by the major address. The major address begins with '0' for the center column and alternates between the right and left halves of the device for all the CLB columns, then IOB columns, and finally block SelectRAM interconnect columns. The frame and bit data give the upset bit row location by using the equation (1).

```
Row = (Byte Frame x 8 + bit) / 18          (1)
```

The frame, frame byte and bit data are used to obtain the exact bit location in the CLB tile by using the equation (2).

```
CLB tile bit = 17 - [Byte Frame x          (2)
8 - Floor((Byte Frame x 8 + Bit) /
18), 18) + Bit]
```

Each bit of the CLB tile has been identified in the FPGA IC schematic and therefore in the design floorplanning by using some internal Xilinx tools. In this way it was possible to build a design flow from the upset bit information coming from the fault injection tool (major address, frame, frame byte and bit) and the final design floorplanning bit location.

The fault injection test platform, shown in Fig. 9, is made from two AFX V300PQ240-100 daughter cards, a MultiLINX^{TM} cable used as an interface to a host PC, and a control panel. The system can operate stand-alone or in conjunction with a host PC and test software. The control panel communicates directly with the control chip to specify the mode of operation. Configuration of the DUT may be controlled by either the control chip or the test software via the MultiLINX Cable. The control chip also controls the dynamic operation of the DUT and dynamic error detection.

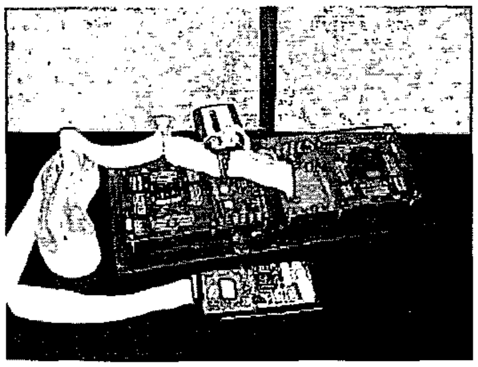
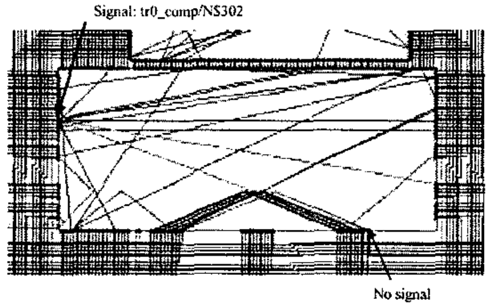


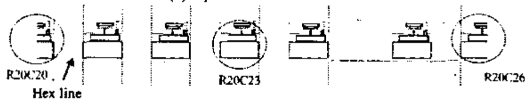
Fig. 9. SEU Test platform

C. Fault Injection Results and Conclusions

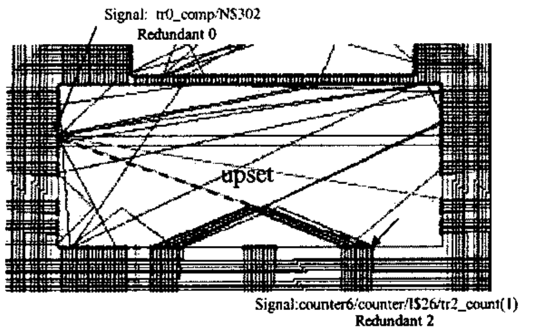
The fault injection was performed in the TMR test design running at 10 and 20 MHz. The report showed that 224 upset bits of 1,663,200 bits in the XQVR300 bitstream had caused a dynamic error in the TMR design. Analyzing the upset bits in the design floorplanning, we observed that a single upset in the routing matrix (GRM) could provoke an undesirable connection between two different signals placed in distinct parts of the FPGA. An example of upset in the GRM that was able to cause an error in the output of the TMR design is located in the major address: 10, frame: 35, frame byte: 46, bit: 5 of the bitstream. Using the equation presented in sub-section III.B, the upset bit in the floorplanning is placed at CLB row 20, column 20 and CLB bit tile: 4. The upset was identified in the IC schematic as I_c_singles.Iw6he1.I377 that means a connection between the single line west 6 and hex line 1, represented in Fig. 10(a). Apparently this upset can not generate an error because it connects a signal from the comparator of the redundant part 0 to "no" signal. However, the hex line connects the CLB R20C20 to two others CLBs as displayed in Fig. 10(b). Analyzing the CLB R20C23, for example, we noticed that actually there is a signal connected to this hex line. The signal is from one of the counters of the redundant part 2, as shown in Fig. 10(c).



(a) Upset in CLB R20C20



(b) CLBs connected by the hex line



(c) Undesirable connection detected in CLB R20C20

Fig. 10. SEU example in the GRM user's design floorplanning

The analyzed upset bit was characterized by an undesired connection between one bit of the 32-bit counter in a redundant module and a signal from the comparator logic of another redundant module. In this case, both comparators 0 and 2 are going to report an error producing "one" in the error flag. This kind of error would never been occurred if the comparators were placed out of the chip.

In order to avoid upset connections between the test design and the comparator test circuitry, a new TMR design based on the "golden" chip approach was implemented in the Virtex device, where the DUT output signal is compared to the golden design placed outside the chip, Fig. 11. In this case, if a single bit upset in the DUT routing matrix provokes an undesirable connection between two signals from different redundant parts of the design, the TMR will always vote the correct signal to the storage elements and to the output. A bit flip in the customization logic will only be able to generate an error if it upsets the exact same bit in two distinct redundant logic parts, which has an extremely low probability to occur. Moreover, this type of error can be totally avoided with a structured floorplanning of the design placement.

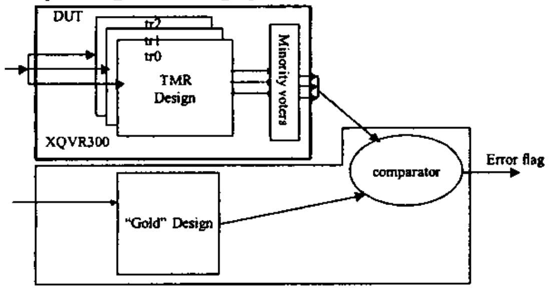


Fig. 11. "Golden" Chip Method

The fault injection experiment using the "golden" chip method was performed in the TMR design running at 25 MHz. The tool has reported "no errors" for all the bits in the bitstream. The result has finally confirmed the efficacy of the TMR structure to recover an error in the FPGA architecture. The radiation characterization results [7] performed at the proton facility in UC Davis show that the *Virtex* FPGA has presented the same reliable results achieved by the fault injection experiment.

IV. Designing and Testing a TMR 8051-like Micro-controller

Micro-controllers implemented in programmable logic platforms are becoming more and more advantageous in order to integrate systems on a single chip (SOC) improving performance, flexibility and time to market. When a micro-controller is implemented in a SRAM based FPGA, not only the registers and memories are sensitive to SEU but also all the programmable logic defined by the FPGA architecture such as the Lookup Tables, routing switches, flip-flops and memories. The previous experiment has shown that the TMR is a successful technique to protect designs against SEU in SRAM-based FPGAs platforms, for this reason it has been applied in the micro-controller architecture too. In addition, a fast time-to-market using off-the-shelf micro-controller architecture for space applications can be achieved by protecting the micro-controller core description and implementing in Virtex QPRO FPGA.

In this direction a micro-controller VHDL description developed at UFRGS (Federal University of RGS) and presented at [8, 9] was re-used to implement the SEU hardened micro-controller into Virtex XQVR300 FPGA using the TMR techniques proposed in [3, 4]. The 8051-like VHDL description is divided into six main blocks as illustrated in Fig. 12. The Finite State Machine (FSM) block implements a counter that generates 24 clock cycles to guide the instruction execution. The Control unit generates all the enable signals for the registers and Arithmetic unit located in the Datapath. The Instruction unit generates the microcode word for each instruction. The datapath includes an Arithmetic Logic Unit (ALU) and many registers. There are two 256 bytes internal memories, one for the data and the other for the application program.

The 8051 micro-controller runs an application based on two 6x6 matrix multiplication at a frequency of 10 MHz. This application performs the multiplication by shifter register and addition. This allows an intensive use of the available memory and internal registers since the operators are read and written many times and both operators and result are stored in the internal memory.

An extra logic circuit was designed to be able to analyze the results of the 8051 after a bit is upset. This block is able to read all the memory data and to send serially the data to an output pin. This output data is compared to the "golden" chip located in a different FPGA device. If the data do not match to each other, the comparator circuit sends a flag error to the fault injection tool. Each corrupted bit able to cause an error in the TMR design is reported in a file.

In order to protect the VHDL description against SEU, each logic block has been triplicated and voters were inserted in all register loops. The datapath, control and instruction logic blocks are mainly throughput logic and consequently they were just triplicated. The registers in these blocks are constantly been written avoiding being locked in a wrong state. The vector signals were replaced by an array of 3 vectors (0, 1 and 2) representing the vector signal for each redundant logic part.

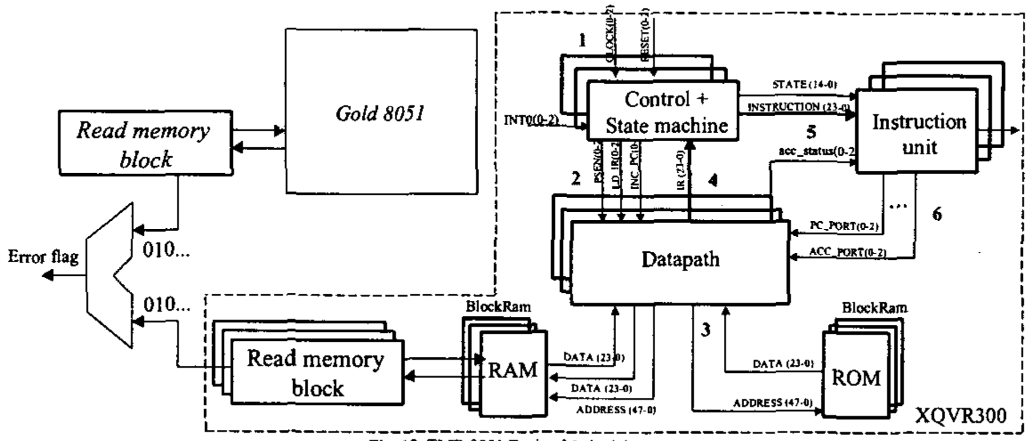


Fig. 12. TMR 8051 Design Methodology

All persistent errors (caused by 'weak-keepers') were avoided by using user ground input and user global clock enable. The registered loops located in the state machine and in the counters were protected by TMR with a major voter in each redundant feedback path. All voters were implemented using LUTs. The internal memories were replaced by the TMR BlockRAM component presented previously in Fig. 7. In the DATA memory there is a circuitry able to detect write conflicts in the memory when refreshing. The micro-controller always has the write priority. In the program memory there is no conflict because it is a read only memory from the micro-controller point of view.

Table III shows a summary of the TMR design logic overhead in the 8051-like micro-controller. The number of flip-flops in the TMR design has increased in 3.6 times. The ratio exceeds 3 because of the extra counters located in the BlockRAM scrubbing logic. The TMR design contains 3 times the number of BlockRAM and each one of them has an extra logic of flip-flops and LUTs for voters, counters and logic analyzes. The number of LUTs in the TMR design is approximately 3.6 times bigger than in the standard design. This proportional also exceeds 3 because of the voters and the scrubbing logic. Three of the four available global clock buffers in device are being used for the system clock.

TABLE III - TMR LOGIC OVERHEAD IN THE 8051 (XQVR300)

Item	Standard 8051	TMR 8051
FDCE	127	459
BlockRAM	2 of 16	6 of 16
TMR BlockRAM extra logic	-	36 FDCE 87 LUTs
Inputs	2	12
Outputs	1	3
BUFG	1 of 4	3 of 4
LUTs	757 (12%)	2778 (45%)

The fault injection experiment was performed in the test board introduced in section III.B. at 10 MHz. Bit flips were inserted in all 1,663,200 bits of the XQVR300 bitstream. Each fault has remained in the bitstream enough time to run many cycles through the application in the micro-controller. The fault injection tool has reported "no errors" by the "golden" system method output in a presence of single upsets in the TMR bitstream.

V. RADIATION TEST RESULTS

The test was performed at Crocker Nuclear Laboratory at UC Davis. The proton energy and fluxes were measured as incident on the DUT package. All tests were performed at room temperature. The beam energy was set to 63.3 MeV. The proton flux was varied from 8.54E+08 to 1.70E+09 protons/sec-cm^2, in order to ensure a scrubbing rate higher than the error rate. The TMR 8051 design was tested in the dynamic mode and compared to the non-protected design. The tested part was XQVR300 (0.22um, 2.5V). The cumulative limit of TID achieved in this test was 116 krads(Si).

The experiment compared the TMR design with and without scrubbing in order to demonstrate the benefits of TMR combined with scrubbing. Results are presented in fig. 13.

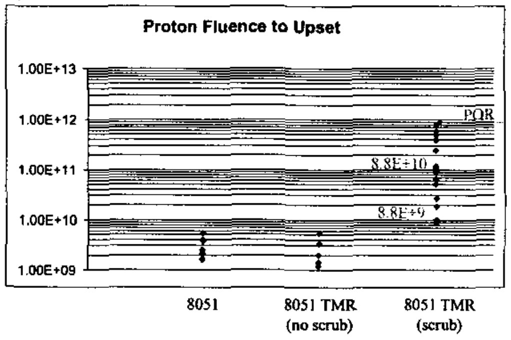


Fig. 13. Testing Platform

Each test measured the fluence to failure. The no-TMR and TMR designs were tested with and without scrubbing. Table IV presents the TMR 8051 cross-section average for the observed fluence to upset collected in the second experiment. The fluence to upset was measured while the PROM was continually scrubbing the configuration bits. The experiment frequency was set at 9 MHz. The same clock was provided to the scrubbing PROM. The BRAM refreshing performed inside the DUT used the same clock divided by 8. It takes 4 ms to entirely run the two 6x6 matrix multiplications (application) and the internal memory read. The scrubbing takes 22 ms to refresh the whole matrix. And the BRAM refreshing takes 0.2275 ms to refresh all addresses.

TABLE IV - TMR 8051 CROSS-SECTION IN VIRTEX^7

Upset	Hit	Cross-section (cm^2)
Bit	18	6.93E-12
Persistent	2	1.91E-10
POR	0	
Average		2.54E-11

In summary the application runs 4 times during a scrubbing cycle and the BRAM is refreshed 17 times per application cycle. The application re-starts with a reset in the micro-controller coming from the read memory logic. In general, bits from the BRAM and the CLB flip-flops (user logic upsets) have the highest refresh rate. The LUTs, customization and routing bits (configuration upsets) are refreshed by the scrubbing rate.

An error can just occur in the design functionality if the number of accumulated upsets is enough to overcome the TMR. For example, if an upset in the routing occurs in the first application execution time of the scrubbing cycle, 2 out of 3 TMR parts should be able to vote the correct value. However this undesirable connection or disconnection may affect different parts of the design generating upsets that can be stored in different redundant parts. All of these upset cells must be refreshed with their original values. If the refreshing rate is such that one can not avoid the accumulation of upsets, errors are going to be observed in the output. It is important to analyze how the upsets can propagate inside the architecture. In each application cycle, the CLB flip-flops are reset, however the BRAMs are never reset, they have always been refreshed by voting their own values. If the refreshing in the BRAMs is not fast enough to avoid accumulation of upsets, a failure can be observed in the output.

The average of TMR 8051 error rate = 17 bits/upset (6e-2 upsets/bit/s = 190 upsets/bit/day) and the average of scrub rate is 45. This means that in average 0.4 upset per bits scrubbing. This rate could be insufficient, besides the point that the flux is not always constant (2 or more upsets can occur during a scrubbing cycle) and the upsets can propagate in the architecture generating more upsets. In real applications the scrubbing rate should be at least 2 orders of magnitude higher than the error rate.

In order to improve the results, two solutions can be evaluated. The first option is to set the proton flux in the radiation facility one order of magnitude or more lower in order to be sure that there is only one upset per scrubbing cycle. In space the flux is much, much lower than the test 99% of the time. However a very low flux would take a long time to observe each error. The other solution is to speed up the scrubbing frequency. However the PROM used can only achieve up to 16 MHz with reliable performance. Additional tests will be performed with a faster scrubbing rate and memory refreshing.

VI. Conclusions

This work has validated a test methodology to evaluate the effects of a single bit upset in the Virtex architecture. Using the steps mentioned in the paper, it is possible to find the specific location of an upset bit in the FPGA IC schematic and consequently in the user's design floorplanning. This methodology was useful to invalidate the previous test design methodology and to show the new results from the "golden" chip approach.

The fault injection analysis results show that the "golden" TMR method has presented "no errors" in presence of single bit upset in the Virtex bitstream. This report demonstrates the efficiency of the TMR SEU mitigation technique for SRAM-based FPGAs. Radiation ground testing performed in the "golden" chip TMR methodology in the counter design confirmed the results achieved by the fault injection experiment.

The fault injection results in the TMR 8051-like micro-controller were meaningful to continue the validation of this technique and it has confirmed the reliability of the Virtex TMR design techniques. The radiation test of the TMR 8051 micro-controller has shown the efficiency of the TMR mitigation technique to a large set of space applications.

VII. Acknowledgments

The authors wish to thank Jeff Lindholm, Scott Frake and Steve Trimberger of Xilinx and Anthony Salazar of Los Alamos National Laboratory for their support of this work and professor Luigi Carro from Federal University of RGS in Brazil (UFRGS) for his contribution of the original 8051-like micro-controller description.

VIII. References

[1] XILINX, INC. Virtex™ 2.5 V Field Programmable Gate Arrays, Xilinx Datasheet DS003, v2.4, Oct. 2000.

[2] XILINX, INC. QPRO™Virtex™ 2.5V Radiation Hardened FPGAs, Xilinx Application Notes 151, v1.3, Feb. 2000.

[3] CARMICHAEL, C., CAFFREY, M., SALAZAR, A., Correcting Single-Event Upsets Through Virtex Partial Configuration, Xilinx Application Notes 216, v1.0, Jun. 2000.

[4] CARMICHAEL, C., Triple Module Redundancy Design Techniques for Virtex Series FPGA, In: Xilinx Application Notes 197, v1.0, Mar. 2001.

[5] FULLER, E., CAFFREY, M., SALAZAR, A., CARMICHAEL, C., FABULA, J., "Radiation Testing Update, SEU Mitigation, and Availability Analysis of the *Virtex* FPGA for Space Re-configurable Computing", NSREC, Jul. 2000.

[6] XILINX, INC. Virtex Series Configuration Architecture User Guide, Xilinx Application Notes 151, v1.3, Feb. 2000.

[7] CARMICHAEL, C., FULLER, E., FABULA, J., LIMA, F. Proton Testing of SEU Mitigation Methods for the *Virtex* FPGA. In: MAPLD, 2001.

[8] CARRO, L.; PEREIRA, G.; SUZIN, A. Prototyping and Reengineering of Microcontroller-Based Systems. In: IEEE Rapid Systems Prototyping Workshop. Proceedings... June 1996.

[9] SILVA, L.; LIMA, F.; CARRO, L.; REIS, R. Synthesis of the FPGA Version of 8051, UFRGS Microelectronics Seminar (12), June 6-7: Porto Alegre, BRAZIL, pp. 115-120, 1997.