Synchronizing Triple Modular Redundant Designs in Dynamic Partial Reconfiguration Applications

Conrado Pilotto José Rodrigo Azambuja Fernanda Lima Kastensmidt

Universidade Federal do Rio Grande do Sul (UFRGS)
Instituto de Informática
Av. Bento Gonçalves 9500, Porto Alegre - RS - Brazil
{cpilotto, jrfazambuja, fglima} @ inf.ufrgs.br

ABSTRACT
This paper presents an innovative method that allows the use of dynamic partial reconfiguration combined with triple modular redundancy (TMR) in SRAM-based FPGAs fault-tolerant designs. The method uses large grain TMR with special voters capable of signalizing the faulty module, and check point states that allow the sequential synchronization of the recovered module. As a result, only the faulty domain is reconfigured, minimizing time and energy spent in the process. In addition, the use of check-point states avoids system downtime, since the synchronization of the recovered module is performed while the others are kept running. Experimental results show that the method has a reduced fault recovery time compared to the standard TMR implementation, maintaining the compatible area overhead and performance.

Categories and Subject Descriptors
B.8.1 [**Integrated Circuits**]: Performance and Reliability - *reliability, testing, and fault-tolerance.*

General Terms
Design, Reliability

Keywords
Dynamic Partial Reconfiguration, FPGA, Fault Tolerance, TMR.

1. INTRODUCTION
The last-decade advances in the semiconductor industry have led to the fabrication of high density chips that integrates an up-rising number of functionalities. However, the same technology that made possible all this progress also brought new challenges. The transistor scaling, reduced voltage operation and higher frequencies utilization have significantly reduced the reliability of integrated circuits by making them more susceptible to faults caused by energized particles [1]. As a consequence, high reliable applications demand fault-tolerant techniques capable of recovering the system from a fault with minimum downtime and with minimum implementation overhead.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SBCCI'08, September 1–4, 2008, Gramado, Brazil.
Copyright 2008 ACM 978-1-60558-231-3/08/09...$5.00.

One of the major concerns is known as soft error, which is defined as a transient effect fault provoked by the interaction of energized particles with the PN junction in the silicon. This upset temporally charges or discharges nodes of the circuit, generating transient voltage pulses that can be interpreted as internal signals, thus provoking an erroneous result [2]. This type of error occurs more often in integrated circuits that contain a large number of memory cells, as a result of the reduced transistor size and high density of this kind of component. Consequently, SRAM-based Field Programmable Gate Arrays (FPGAs) operating in hash environments and in high reliable applications must be protected against this kind of errors [3].

Soft errors have a peculiar effect in designs implemented in SRAM-based FPGAs [3]. When a soft error occurs in a SRAM cell that customizes the logic, it flips the original stored value provoking a persistent error and a system malfunction until the next device configuration. For this reason, designs implemented in such devices are protected by combining two techniques: triple modular redundancy (TMR) and scrubbing [4]. However, there are two drawbacks in using these approaches.

The first one is that a single upset may change a configuration bit that controls the routing, provoking a shortcut between two wires of distinct redundant domains. This would affect more than one module of the TMR design, inducing the majority voters to select the erroneous result, and leading to probable errors in the system's output. In order to reduce the probability of a fault in the routing overcoming the robustness of the TMR, it is necessary to minimize the connections between the redundant domains. If the number of majority voters could be reduced, the connections between the blocks would decrease as well. This way, it would be feasible to place them in a sparser manner, reducing the probability of upsets that affect more than one module.

The second drawback is the time required for scrubbing and the dissipated energy of doing it. As FPGAs are becoming larger, the time to load an entire bitstream in high density devices (more than 11Mbits of customization cells) are reaching the hundreds of milliseconds. According to the application and environment, this time can be too long to avoid multiple upsets caused by a burst of ions. Other than that, most of the customization bits are being continuously loaded with no need, once they have not been flipped by an energized particle. This strategy consumes a great amount of energy in the board, so it would be convenient to restore only small portions of the circuit, and only when they suffer from the effect of upsets.

In this paper, the authors propose an innovative method that focuses on overcoming the two issues mentioned above. The first

requirement for that is to mitigate the probability of single faults that affect the routing of two domains. This is accomplished decreasing the connections between the modules by using large grain TMR, where voters are placed only at modules output. The second is to decrease the consumed energy and the mean time to repair from a fault. To achieve this, the method uses selective dynamic partial reconfiguration to restore only the module affected by a fault. However, a new challenge is introduced when all internal voters are removed: once they are no longer voted, memory elements will have inconsistent values after the reconfiguration. Some related works, such as [5] and [6], have shown dynamic partial reconfiguration schemes in TMR designs, but they did not address the issue of synchronization. The authors in this paper propose a method that introduces the use of check-points inside the circuit's control block, providing a safe state for the reconfigured module to start operation in accordance to the others. The method was developed and validated in a FSM case study circuit.

2. RELATED WORK

When an upset occurs in the combinational logic implemented on a SRAM-based FPGA, it provokes a very peculiar effect not commonly seen on ASICs. The faults induced by this upset can be opens or shortcuts in the routing, or variances in the customization of the CLBs. Their behavior is characterized as a transient event followed by a persistent effect. Usually space redundant techniques are used to protect the system, since they ensure correct outputs even in presence of a persistent upset. In this case, combinational and sequential logic gates are triplicated and voted by majority voters (MAJ voter). These voters must be placed each time there is a user's register. Also, it is recommended to add a feedback path to correct the upset in those registers, as scrubbing cannot correct faults in memory cells. This method is known as XTMR [4] and is illustrated in figure 1. The spatial redundancy ensures that the outputs are correct as long as there are at least two correct modules.

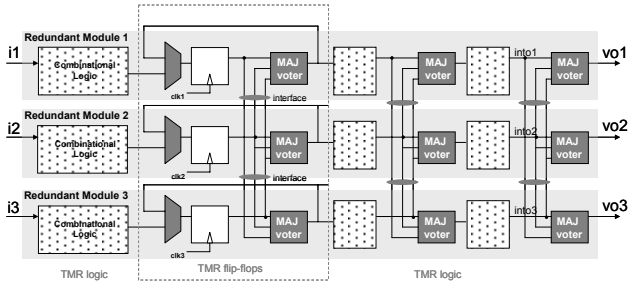
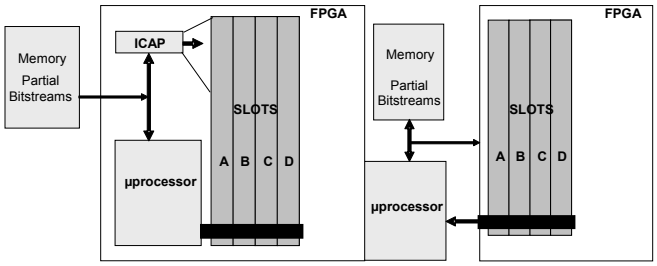


Figure 1 – XTRM approach proposed by Xilinx

Nevertheless, a single upset on a routing cell may provoke a fault that affects two modules at the same time. Previous results [3, 7 and 8] showed that this kind of faults sometimes generate an error in the TMR application output. As 90% of the SRAM cells inside the FPGA are responsible for the routing control [3, 7 and 8], these upsets represent the main concern in fault-tolerant designs. The probability of this kind of soft errors depends on the logic placement for the TMR, which depends directly on the number of majority voters. Each time a new voter is included, new connections between redundant parts must be made, forcing the redundant domains to be placed near each other, hence increasing the probability of upsets generating shortcuts between the interconnections.

Spatial redundancy by itself it is not sufficient to avoid errors in the FPGA. Since faults in the customization bits have a persistent effect, it is necessary to reconfigure the device in order to correct them and restore the system robustness. To achieve that, in most of the times, the complete bitstream is continuously loaded by a process called scrubbing [4]. Scrubbing allows a system to repair bit-flips in the configuration memory without disrupting its operation. This technique can correct upsets in the LUT cells, in the General Matrix Routing (GMR) and in the CLB logic cells. However, it does not refresh the contents of CLB's flip-flops, so the reconfiguration cannot restore user registers, or in other words, the state of the system. This is the reason that the XTMR has a feedback path with majority voters at each flip-flop, as shown in figure 1.

Using the SelectMAP interface [9], the same used for scrubbing, it's also possible to perform a partial reconfiguration of the device. This kind of reconfiguration allows the device to be reconfigured only at the desired columns, leaving the rest of the device customization intact. If this process is performed while the system is operational and without interrupting the rest of the application, it is called Dynamic Partial Reconfiguration [10]. As a consequence, it's possible to partition portions of the FPGA in slots that will accommodate different modules of the system. For system-on-chip (SoC) platforms, the Hardware Internal Configuration Access Port (HWICAP) module can also be used to perform dynamic partial reconfiguration from inside the FPGA. To accomplish this task, it's necessary to store partial bitstreams (i.e. bitstreams corresponding only to the module being reconfigured) in an external memory. Dynamic partial reconfiguration is a solution to reconfigure the matrix in small portions or slots, reducing the reconfiguration time. Figure 2 represents the partial reconfiguration scheme from inside and outside the FPGA. Each slot (A, B, C and D) can be reconfigured separately when needed.



(a) From inside the FPGA (b) From outside the FPGA

Figure 2 – General schemes for partial reconfiguration

3. PROPOSED TECHNIQUE

The main objectives of the proposed method are: to reduce the probability of upsets affecting more than one redundant module in the TMR and to reduce time and energy used to recover from a fault. To achieve the first one, a large grain TMR can be used to minimize the number of connections between the redundant domains. To achieve the second, dynamic partial reconfiguration is used to reconfigure only the module affected by an upset. The proposed redundant scheme is illustrated in figure 3, where each module has been replicated, and majority voters capable of signalizing the fault module are placed only at outputs.

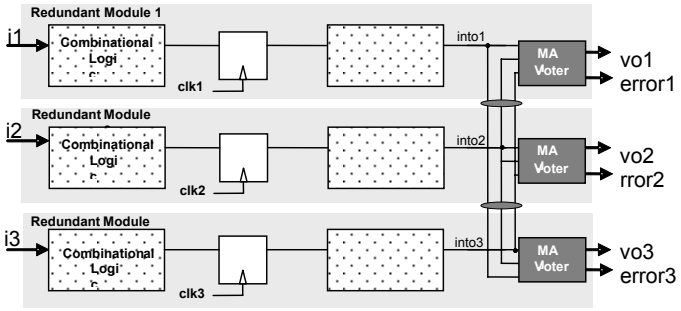


Figure 3 – The large grain TMR proposed

The challenge of this approach lies on synchronizing the recovered module with the correct ones, after the partial reconfiguration is realized. Synchronizing a reconfigured module is one of the main issues found is this kind of methodology. Current applications using this technique on other scenarios [10] restore the state of the reconfigured module by reading it from an external memory. This is possible when the modules are not always present in the system. Once a module is not being used, it can be offloaded from the FPGA and have its state saved. However, in TMR scenarios, the modules are constantly running, so there is no context saving. Synchronizing modules represents having to set parts of the system on hold until the updating is done. The amount of clock cycles needed for the reconfiguration depends on the size of the area to be reconfigured. This can range from few to hundreds of milliseconds. Having the system's output on hold for a period of time may be unacceptable for some applications, such as real-time.

In order to avoid the system to stop, the proposed method is based on predicting a future state to which the system will soon converge (check-point state), and presetting the reconfigured module to it. This means that only the reconfigured module will be set on hold while the other two continue their tasks. Once the other two modules reach that check-point state, the reconfigured block can be released and the three redundant modules start to operate in phase again. Analyzing the functional waveform of the proposed method in figure 4, it is possible to identify three important and distinct time frames.

The first is the time between the fault occurrence and its identification by the voters. This window is variable, since the circuit may take a while to exercise the part affected by the upset and generate a wrong result. The second is the amount of time spent to perform the dynamic partial reconfiguration of the fault module (T_{pr}). This time depends on the size of the module being reconfigured (number of columns it occupies in the FPGA), so it's a constant for identical modules. And the last is the time for the correct modules to reach the check-point state (T_{rs}). T_{rs} depends on the logical distance between the application state, after the reconfiguration process is over, and the checkpoint in which the reconfigured module will stand on hold.

Note that the voted outputs (vo1, vo2 and vo3) continue to be valid during the process, as two modules continue to operate correctly.

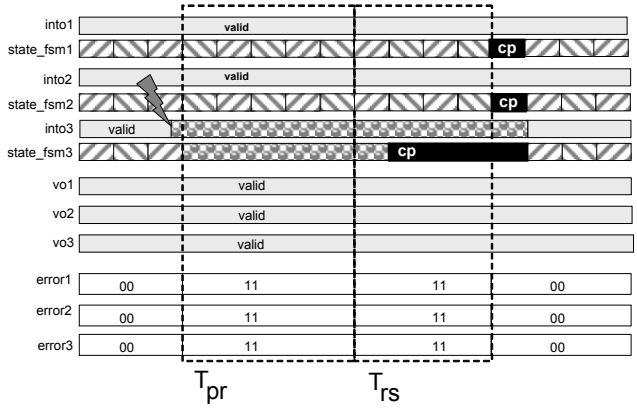
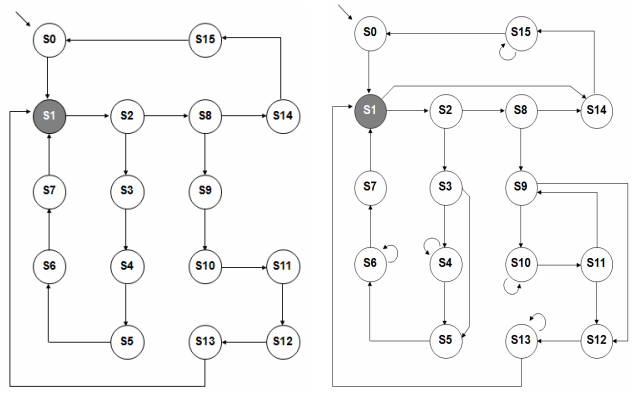


Figure 4 – Functional waveform for the proposed method

In this paper, the method is validated in a Finite State Machine (FSM). FSMs are great case-study circuits because they have registers, to store the internal state; combinational logic, to drive the next state and the outputs; and a behavior that has branches and loops, as a general algorithm implemented in hardware.

Theoretically, any state of the system can be used as a check-point state, but in practice, there are some constraints that must be analyzed. First, it should be a state that is frequently reached during the application. This minimizes the time necessary for the other modules to reach the check-point (T_{rs}). Second, in case there are loops and branches in the FSM, it is necessary to ensure that the application will pass through that state in a near future. Third, a higher number of check-points increase the width of the check-point signal shared by the voters. Therefore, it's necessary to perform a comprehensive statistical study of the circuit behavior in order to determine the best check-points and to reduce the T_{rs}.

Let one start by analyzing an example. Figure 5 shows two case-studies FSMs with 16 states each. The FSM presented in fig. 5(a) has few loops, while the FSM in fig. 5(b) has some branches and some loops.



(a) FSM A **(b) FSM B**

Figure 5 – Case-study Finite State Machines

The FSM A has 3 minimal loops paths: (1) S1, S2, S8, S14, S15, S0, S1; (2) S1, S2, S3, S4, S5, S6, S7, S1 and (3) S1, S2, S8, S9, S10, S11, S12, S13, S1. All of them have state S1 as a common state, meaning all loops converge to it at some time during the application. For that reason S1 is a good check-point candidate. For FSM A, the worst case in terms of T_{rs} is when the correct machines are at state S2. When that happens, it is necessary to

wait 5 clock cycles to reach check-point S1 if the machines are on loop (1), 6 clock cycles if the machines are on loop (2) and 7 clock cycles if the machines are on loop (3). In order to reduce the T_{rs} time, it is possible to add more check-points, breaking the loops in half, for instance. So, loop path (1) would have S1 and S14 as check-points, reducing the T_{rs} time from 5 clock cycles to 3 clock cycles. Loop path (2) would have S1 and S5 as check-points, also reducing T_{rs} time to 3 clock cycles. And loop path (3) would have S1 and S10, reducing T_{rs} time to 4 clock cycles.

However, by adding extra check-points, it is also necessary to increase the width of check-point signals, so they are able to drive the correct amount of check-points. In addition, it is necessary to have a circuit to predict which would be the best check-point state to preset the reconfigured module to. This circuit can be as simple as a memory table or can be complex if it has multiple parameters to evaluate; all depends on the behavior of the application. When using more than one check-point, the overhead of extra signals increases by the function of 2×Ceil (Log2 (#check-points+1)), where Ceil(x) returns the lowest integer equal or higher than x.

On the other hand, FSM B of figure 5(b) has 12 minimal loop paths: (1) S1, S2, S8, S14, S15, S0, S1; (2) S1, S14, S15, S0, S1; (3) S15; (4) S1, S2, S3, S4, S5, S6, S7, S1; (5) S1, S2, S3, S5, S6, S7, S1; (6) S4; (7) S6; (8) S1, S2, S8, S9, S10, S11, S12, S13, S1; (9) S1, S2, S8, S9, S12, S13, S1; (10) S9, S10, S11, S9; (11) S10 and (12) S13. Note that S1 is present in all loop paths. Also, if it is considered that the smaller loops (3), (6), (7), (10), (11) and (12) are part of larger loops that pass through state S1, then the state S1 is present in all branches, so it can be also be considered a good candidate for a check-point state in FSM B. However, now it is not possible to estimate the T_{rs}, because there are many internal loops, whose behavior depends on the application inputs. By analyzing the application functionally and the probability of inputs, it is possible to ensure that machine will pass by the check-point, but it is not always possible to calculate the exactly number of cycles to reach it. In this case, extra check-points can be added according to the smaller loops in order to reduce the T_{rs}. For instance, check-point states can be also S15, S4, S6, S9, S10 and S13.

Since T_{pr} time is about few milliseconds and T_{rs} time is about few clock cycles, the difference between them can be more than 3 orders of magnitude in designs operating at MHz. So, for many applications the use of only one check-point is already enough and multiple check-point states will represent minimal improvement in the final time for fault recovery.

The proposed method is now presented in figure 6. The original FSM was modified so it can be preset to a check-point state through the signal (rst_cp). Also, it was added a signal (cp_out) that outputs if the machine is at a check-point state. The MAJ voters were also modified, so they can handle the check-points scheme and signalize which module presents the erroneous result. Comparing to the traditional MAJ voter, the proposed voter has the following extra signals: the check-point state status (cp_out) as an input, a warning signal that indicates which module is faulty (error) as an output, and the preset for the check-point state (rst_cp) also as an output. The width of signal into (m) depends on the standard FSM design, the error signal width is always 2 ("00"- no error, "01" - error in module 1, "10" - error in module 2 and "11" - error in module 3), and the rst_cp and cp_out width (n) depend on the number of check point states.

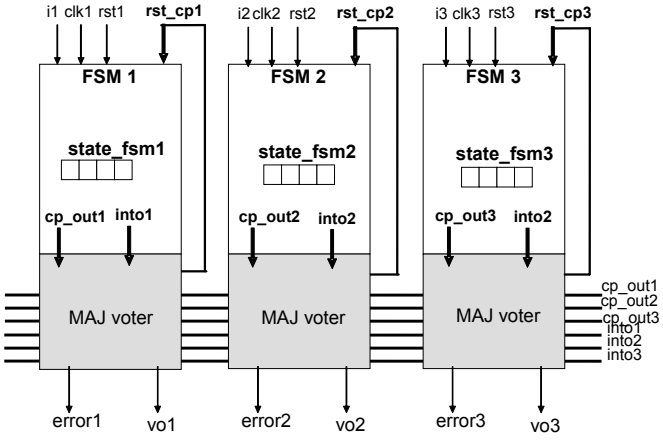


Figure 6 – Proposed Scheme with FSMs + MAJ Voters

For example, in a FSM with 1 check-point state and 1 bit output, if the TMR design outputs (into1 = '0', into2 = '0' and into3 = '1') , this indicates that module 3 is the one in disagreement. The voted outputs are (vo1 = '0', vo2 = '0' and vo3 = '0') and the error outputs (error1, error2 and error3) are all equal to "11", which indicates that module 3 is the faulty one. The module 3 is reconfigured while the others continue to process the data. The rst_cp3 is set to '1' until cp_out1 = cp_out2 = to cp_out3. When that happens, rst_cp3 is set to '0' and all the three modules continue to operate in phase.

Figure 7 shows the simplified functional flowchart for the Majority voter. The MAJ voter logic can be easily modeled in a truth table and implemented by a set of LUTs in the FPGA.

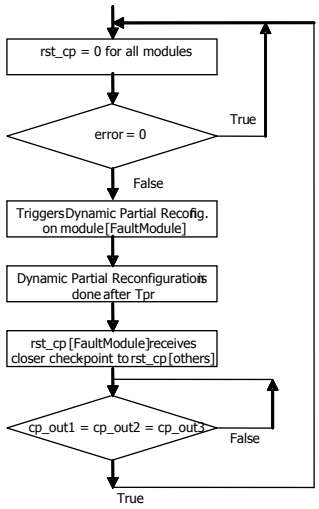


Figure7 – Functional Flowchart for Modified MAJ Voter

4. EXPERIMENTAL RESULTS

The techniques for soft-errors mitigation here discussed were designed, simulated and implemented on a Virtex-II Pro (2vp30ff896-7) platform [9] in order to confirm the results based on theoretical speculation. This FPGA part has a bitstream of 11,589,920 bits. The proposed method and the XTMR are compared in relation to area resources, maximum operation frequency and quantity of shared signals. Also, it is presented for

both approaches the experimental times needed by the system to restore from a possible bit-flip error.

The system architecture used to implement the technique consists of a PowerPC core, an OPB ICAP controller, an OPB SysAce controller and a RS232 UART core. The design was conceived using ISE and EDK tools version 8.1, and partial reconfiguration was achieved using the Modular Flow described in [9].The system was validated by injecting faults in the modules to simulate a soft error.

The implemented circuit is the FSM B of figure 5(b) with one check-point at S1 (n = 1). Recalling, it has 16 general states, 4-bit input and 2-bit output (m = 2). Modifications in the voter, enabling it to handle the check-point signals, presented minor overhead in terms of device resources used and signal sharing. The results are shown in table1.

Table 1 – Voter overhead analysis

Voter	Flip-Flops	LUTs	Shared Signals
Standard	0	2	Width (m) = 2
Proposed	1	7	Width (m + n) = 3

Two different methodologies were used in the internal state codification in order to compare the gain and loss of our method in different cases. First the circuit was codified using standard binary codification, and then was encoded with one-hot technique. Although it doesn't make a substantial difference in terms of application, this shows the implications of having a higher number of registers in the TMR circuit; both in terms of resources used and signal sharing. Table 2 presents the resource utilization and maximum operation frequency for the FSM using binary encoding, the most traditional form of state representation.

Table 2 – Area and performance using binary encoding

Design	Flip-Flops	LUTs	IOs	Max. Freq.
Standard FSM	4	24	8	364.884 MHz
XTMR	12	96	24	348.693 MHz
Proposed TMR	15	135	30	284.910 MHz

As a consequence of the feedback path of the signal rst_cp, and the necessity of having it registered, the proposed method suffered from a loss of 18% in the maximum frequency allowed. This may not represent a drastic drawback depending on the device operation frequency. Also, different applications may have less impact on the frequency loss, as will be shown in the other example.

Table 3 presents the amount of shared signals between the redundant independent modules of the TMR application referent to the Binary Encoded FSM. These numbers are highly dependent on the number of voters that were added to the original circuit. Due to the reduction of the voter quantity, the method proposed in this paper has reduced by a factor of 2 the interconnections between the independent domains.

Table 3 – Shared Resources using Binary Encoding

	XTMR	Proposed TMR
Register Voters	12 (Standard)	-
Output Voters	3 (Standard)	3 (Proposed)
Shared Signals	**18**	**9**

Since different applications may have a higher number of registers, because of different forms of implementation or design constrains, the FSM was also encoded with One-Hot scheme. This implies in 16 registers for a 16-state FSM. Table 4 shows the experimental results for such implementation.

Table4 – Area and Performance using one-hot encoding

Design	Flip-Flops	LUTs	IOs	Max. Freq.
Standard FSM	16	24	8	713.216 Mhz
XTMR	48	279	24	332.309 Mhz
Proposed TMR	51	99	30	310.264 Mhz

This time the frequency loss compared to the XMTR was lower than with binary encoding. Even so, both approaches have a real significant loss compared to the standard non-protected FSM. On the other hand, the LUT utilization rate is now 3 times smaller. This result it is due to the significant reduction in the number of shared signals by voters, which minimizes the routing between independent domains and demonstrates the gain of the proposed method with the increasing number of internal registers. Table 5 presents the results for signal sharing with one-hot encoding.

Table 5 – Shared Resources using one-hot encoding

	XTMR	Proposed TMR
Register Voters	48 (Standard)	-
Output Voters	3 (Standard)	3 (Proposed)
Shared Signals	**54**	**9**

In presence of a fault signalized by the modified voter (error signal), a software in the embedded processor loads the partial bitstream from the external memory and writes it into the ICAP port. The time consumed to reconfigure the modules was measured by the same software using the XTime.h library. The time taken by the scrubbing process is estimated in accord to [6] and [9], and refers to the time spent to reconfigure the entire device, spanning one column at a time. This time is valid for comparison, since in the worse case, the fault will occur in the column that was just refreshed; therefore, the latency for being reconfigured again will be the entire device reconfiguration time. Table 6 contains the times necessary for both reconfiguration methods.

Table 6 –Reconfiguration Time

Design	Reconfig. Time
XTMR	940.00 ms (full)
Proposed TMR Binary Encoded	13.62 ms (partial)
Proposed TMR One-Hot Encoded	13.10 ms (partial)

5. CONCLUSIONS AND FUTURE WORK

In this paper, the authors have presented a new approach to deal with faults affecting redundant designs and reconfigurable systems. The method here discussed reduces the number of shared signals proportional to the number of internal registers, decreasing the probability of a single fault overcoming the TMR robustness. Also, it has been demonstrated how the concept of check-points can be used to synchronize the reconfigured modules preventing the complete application from freezing. Moreover, using dynamic partial reconfiguration, the reconfiguration process can be up to 70x times faster as the traditional approach. As a result, energy

dissipation and the overall fault recovery time are highly improved.

Future works include the implementation of the proposed method in a real case scenario, such as a TCP/IP algorithm, and the tradeoff analysis of multiple check-point states in the fault recovery time.

6. REFERENCES

[1] International Technology Roadmap for Semiconductors: 2005 Edition, Chapter Design, 2005, pp. 6-7.

[2] P. E. Dodd, L. W. Massengill, "Basic Mechanism and Modeling of Single-Event Upset in Digital Microelectronics", IEEE Transactions on Nuclear Science, vol. 50, 2003, pp. 583-602.

[3] F. Kastensmidt, G. Neuberger, C. Carro, R. Reis, R. Hentschke, "Designing Fault- Techniques for SRAM-based FPGAs", IEEE: Design and Test of Computers (D&T), v.21, n.6, Dec., 2004.

[4] C. Carmichael. "Triple Module Redundancy Design Techniques for Virtex FPGAs". Xilinx Application Notes 197, 2006.

[5] K. Paulsson, M. Hübner, M. Jung, J. Becker, "Methods for Run-time Failure Recognition and Recovery in dynamic and partial Reconfigurable Systems Based on Xilinx Virtex-II Pro FPGAs", Proceedings of the 2006 Emerging VLSI Technologies and Architectures, ISVLSI, 2006.

[6] C. Bolchini, A. Miele, M. D. Santambrogio, "TMR and Partial Dynamic Reconguration to mitigate SEU faults in FPGAs", 22nd IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, DFT, 2007.

[7] F. Kastensmidt, L. Sterpone, L. Carro, M. Sonza Reorda, "On the Optimal Design of Triple Modular Redundancy Logic for SRAM-based FPGAs", in the Proceedings of Design Automation and Test in Europe (DATE), IEEE, 2005.

[8] L. Sterpone, M. Reorda, M. Violante, "RoRA: a reliability-oriented place and route algorithm for SRAM-based FPGAs", Research in Microelectronics and Electronics, 2005, Volume 1, 2005. p.173 – 176.

[9] Xilinx Inc. ISE and EDK Manuals, Datasheets and User's guide Manuals for Virtex-II Pro. http://www.xilinx.com.

[10] J. Becker, M. Hubner, G. Hettich, R. Constapel, J. Eisenmann, J. Luka, "Dynamic and Partial FPGA Exploitation", Proceedings of the IEEE, Volume 95, Issue 2, Feb. 2007, p. 438 – 452.