A NOVEL SRAM-BASED FPGA ARCHITECTURE FOR EFFICIENT TMR FAULT TOLERANCE SUPPORT

Konstantinos Kyriakoulakos[†] *and Dionisios Pnevmatikatos*[†‡]

[†]ECE Department
Technical University of Crete
Chania, Crete, GR73100, GREECE
{kyriakoul, pnevmati}@mhl.tuc.gr

[‡]FORTH-Institute of Computer Science
Computer Architecture and VLSI Laboratory
Heraklion, Crete, GR 71110, GREECE
pnevmati@ics.forth.gr

ABSTRACT

This paper proposes a novel SRAM-based FPGA architecture that is suitable for mapping designs when fault tolerance is desirable. TMR has been successfully applied in FPGAs to mitigate transient faults, which are likely to occur in harsh environments such as in space applications. In addition, fault tolerance techniques gain importance as feature sizes shrink and make circuits less reliable. However, TMR comes at high area penalty, which increases as the TMR grain becomes finer. We propose a slight modification to existing SRAM-based FPGA architectures to support fine grain redundancy at an area cost even less than 3x (1.76 x in average for our benchmark circuits). Our approach also provides accurate fault location and allows smaller and more infrequent reconfigurations saving both reconfiguration time and power.

1. INTRODUCTION

Fault tolerance for semiconductor devices has been important ever since upsets were first experienced in space applications. Space applications must consider the effect energetic particles (radiation) can have on electronic components. Single Event Effects (SEE) occur when charged particles hit the silicon, transferring enough energy in order to provoke a state change and consequently a fault in the system, with potentially serious consequences for the application, including loss of information and functional failure. This phenomenon is a concern in space, but also in other harsh operating environments.

When SEE have a transient effect we refer to their consequences as Single Event Upset (SEU). The main SEU are bit flips in the memory elements. For FPGAs in particular, SEUs may alter the logic-state of any static memory element (latch, flip flop, or RAM cell) or cause transient pulses in combinatorial logic paths. Since the user-programmed functionality of an FPGA depends on the data stored in millions of configuration latches within the device, an SEU in the configuration memory array might have adverse effects on the expected functionality of the user implemented design. Similarly, Single Event Transients (SETs) have a high probability for recognition at flip flop inputs where, if registered, causes a soft-error in the user data.

Static upsets in the configuration memory are not necessarily synonymous with a functional error; however, soft-errors are by definition functional errors, and upsets may or may not have an effect on functionality depending on whether the particular LUT is in use, the fault is masked by some other logical condition, etc. However, accumulation of upsets in the configuration memory eventually leads to errors and then to functional failure.

Several SEU mitigation techniques have been proposed to avoid the effects of faults in digital circuits, including FPGAs. In particular, for SRAM-based FPGAs, SEU mitigation solutions can be classified in architectural and high-level solutions. Architectural level solutions are in fact suggestions of new architectural technologies and topologies such as using hardened memory cells for SRAM-based FPGAs, or using innovative routing structures. High-level techniques do not require changes in the FPGA's architectural structure, suggest user-level circuit design techniques that achieve SEU mitigation and can be easily applied by the user or CAD tools. High-level techniques are more attractive than architectural ones because they can be readily applied to all implementation technologies. They basis for high-level techniques is the use of redundancy, and the most well-known such technique is triple modular redundancy (TMR).

In this work we address the problem of faults in the LUT configuration memories cause by SEUs, usually dealt with using TMR and periodic scrubbing (reprogramming of the entire device). We propose a new architectural-level solution aimed to reduce the cost of TMR. Our approach extends with minor changes the LUT and CLB structure of current FPGAs in order to reduce the cost of mapping fine grain TMR. We opt for fine granularity as this gives the best SEU mitigation results. Our approach exploits structures already available in commercial FPGAs, and with minor additions is able to reduce the space overhead of fine-grain TMR to about 1.76x for our benchmark circuits. Additionally, our proposed approach incurs minor performance overhead compared to a doubling of the latency for traditional fine-grain TMR. In this way, our proposed architecture addresses the main disadvantages of the fine-grain TMR.

2. BACKROUND AND RELATED WORK

The first step of a fault-tolerant scheme is fault detection. Fault detection has two purposes: first to alert a supervising process that action needs to be taken for the system to remain operational and, second to identify which components of the device are defective so that a solution can be determined. These two functions may be addressed simultaneously or by a multi-stage process comprising of different strategies. Usually, on-line fault detection methods use redundancy, while off-line approaches may reuse the test circuitry to detect permanent faults.

Temporal and spatial (hardware) redundancy techniques are used in ASIC [1][3][4]. They range from concurrent error detection (CED) to correction mechanisms. The use of full time or full hardware redundancy permits voting the correct value in the presence of single upsets. Redundancy is also widely used as a method of fault detection in FPGAs. The most well-known high-level fault tolerance technique is the TMR. Many methodologies have been proposed in order to make any kind of circuit more reliable through the TMR [7][8][9][4].

The basic concept of triple redundancy is that a sensitive circuit can be hardened to SEUs by implementing three copies of the same circuit and performing a bit-wise "majority vote" on the output of the triplicate circuit (Figure 1). TMR works under the assumption that at most a single fault will be present in the replicated functions, and hence at most one of the voter's inputs can be incorrect. The cost of TMR is twofold: (i) area cost is increased due to the triplication of the functions, plus the voter cost [4], and (ii) the latency of the circuit is increased by the introduction of the voter in the circuit's critical paths.

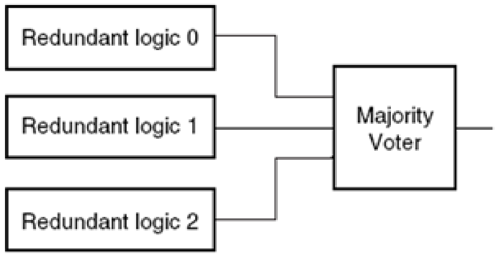

Fig. 1. Basic Tripple Modular Redundancy (TMR).

The TMR technique can be applied at different granularity levels. Fine grain applies the TMR method in small-sized modules and can tolerate more faults. Assuming that faults appear in random locations, when we consider smaller logic modules, the probability of two faults occurring within the 3 copies of the single function is minimal. Coarse grain TMR applies TMR in larger modules in order to minimize the voter area and timing penalty, at the expense of reduced tolerance to multiple faults. The area cost of TMR is (i) the additional two copies of the original logic block(s), plus the voter cost. Depending on the TMR grain this overhead cost ranges from 300% for fine grain TMR at the LUT level, decreasing when the TMR grain is increased asymptotically down to 200%.

Dual Modular Redundancy (DMR) can also be used to detect faults, duplicating all logic blocks and then comparing the two outcomes. Since the error will be equally likely between the two different answers, the discrepancy triggers the recovery action, but this functionality is user/technology specific. Fine grain DMR spatial overhead is 200%, as each logic LUT is doubled and another LUT is used for the comparison.

Redundancy needs not be restricted to only spatial. It is also possible to detect errors using temporal redundancy, trading-off data throughput for reliability. For example, operations can be computed twice by the same circuit, and the two results compared to detect discrepancies [9]. In the second operation, operands are encoded in such a way that they exercise the logic in a different way, and the output is then passed through a suitable decoder and compared to the original. Similarly, temporal redundancy can be used to discover transient faults.

Although most of the work on redundancy has been aimed at detecting and correcting SEUs, there have been publications that apply the techniques to fault detection. DMR is used in [9] to grade the 'fitness' of competing configurations in an evolutionary approach. Parity checking is used in [1] as part of a fault tolerant scheme which is structured so that detection is applied to small regular networks, rather than being bespoke to the function that is implemented. We also discuss DMR in our architecture.

Redundant and data-checking detection systems are generally designed into an FPGA configuration, as they fit around the specific data and control functions that are implemented. In [10], an FPGA structure was considered which has built-in redundancy, so that it is transparent to the user who is designing the configuration.

In addition to TMR, which is an upset mitigation technique, to guarantee system's normal function a recovery technique must also be applied. A periodic reprogramming (called scrubbing) is performed to remove any possible configuration bit flips, with period calculated using the expected upset rate of the device for the given application. Upset rates are calculated from the static cross-section of the device and the charged particle flux the application or mission is expected to endure. A "rule of thumb" is to place this rate one order of magnitude or more above the expected upset rate [6]. Without error detection provided, in reality system should scrub, on average, at least ten times between upsets. With fine grain TMR we slim down the "sensitive" area of a particle strike and so the upset rate as well. This will result in longer period between reconfigurations and thus reduction of power dissipation. For additional information the reader is referred to Stott *et al.* [15], that provides an extensive survey of fault tolerance techniques for FPGAs.

3. PROPOSED ARCHITECTURE

In this section we present our proposed FPGA architecture for fault tolerance support. The basic idea is to augment LUTs with TMR functionality reusing the existing structures as much as possible. Providing incremental changes to the LUT and slice structures allows us to achieve low implementation cost of at the hardware level, and even better TMR cost, both for area and latency. Our proposal involves the implementation of the voter circuit in full-custom logic within the LUT/slice structures in order to achieve fine grain redundancy and better fault tolerance.

We present this work using the Xilinx Virtex 5 family architecture. The LUT structure of this family (figure 2) – and any other that uses a similar LUT structure– has some interesting properties. Virtex-5 LUTs can implement any arbitrary six-input Boolean function. Internally, a 6-input LUT is structured as two 5-input LUTs that can be used independently to implement two different Boolean functions as long as these functions use the same five inputs A1-A5. In this case both O5 and O6 outputs are used. The two 5-input LUTs can be combined using the A6 input and an internal multiplexor, to implement an arbitrary six-input function and output the result to O6.

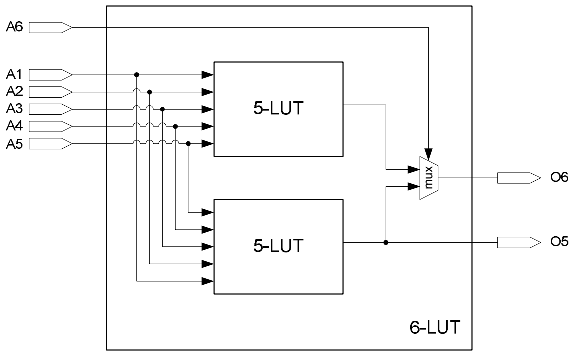

Fig. 2. LUT structure for the Virtex-5 family. Two 5-input LUTs can implement independent functions that share the same inputs. Combined, they implement a 6-input function.

Under certain conditions (for example there is no other function that uses the same 5 inputs with this one), the entire LUT is used to implement a *single* five-input function. In that case only the O5 output is used and one of the two 5-LUTS is not used, i.e. it is redundant. Our approach is to exploit the redundancy and force the design to be mapped and placed only in simple 5-input LUTS so that we "reserve" with other half of each 6-input LUT. Then we will use the reserved LUT capacity in our proposed architecture. In this way, in many (actually most) cases, we will be able to use the new LUT architecture and apply fine grain TMR with an area cost of about 1.76x compared to more than 3x for the traditional TMR.

3.1. DMR Architecture

We explain our technique first for Dual Modular Redundancy support. Having only one of the two 5-input LUTs active to implement the desired function, we can program the other one with the redundant copy of the same function. Figure 3 demonstrates this LUT duplication for a 5-iput function *f*: two copies of the function are mapped in the two halves of the 6-input LUT. Comparison between the two outputs that would require a separate LUT, is done with an additional XOR gate inside the 6-input LUT. Without SEU presence gate's output has value 0, and LUT's output can be either O5 or O6 because they are the same. In DMR mode, the multiplexer selection signal is not useful. When a fault is present, the XOR gate will drive a high signal to the global column fault cable that will alert the system that a fault has been detected somewhere in this column. After this alert no output should be accepted as correct and system must first reconfigure this column. The additional DMR hardware implementation cost is a single xor gate for each 6-input LUT and the global fault presence wire(s). However, an important detail is that there is another form of incurred cost: if DMR mapping is chosen by user, then the user circuit cannot use 6-input LUTs. In cases where a 6-input LUT would be fully used (6-input functions or two 5-input functions with the same inputs), the mapping tool must split them and map them to additional LUTs.

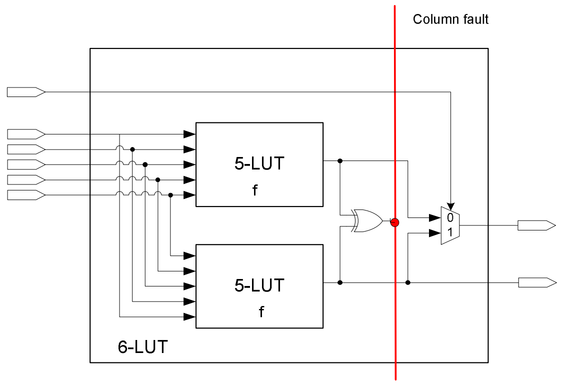

Fig. 3. Dual-Modular Redundancy in our proposed LUT architecture. A single XOR gate compares the two results and signals fault when there is a mismatch.

3.2. TMR Architecture

Following the same strategy we can obtain the TMR architecture by adding the necessary logic. We pose the same restriction, i.e. in TMR mode we allow only 5-input functions and make an important observation: if two of the three copies needed are placed only in a single LUT as in the case of DMR, there is no need to spend an entire 6-input LUT for the third copy of the function. Since the 6-input LUT is actually two 5-input LUTs, they can be used to implement the third copy of two distinct functions. This idea is presented in Figure 4. Thus for 2 functions we need 3 instead of 4 LUTS. As for the DMR, in addition to the LUT implementation cost, there is the cost incurred from restricting the mapping to 5-input LUT only. If this cost is small (as we will show in the next section), then we have significant spatial reduction compared to classic TMR.

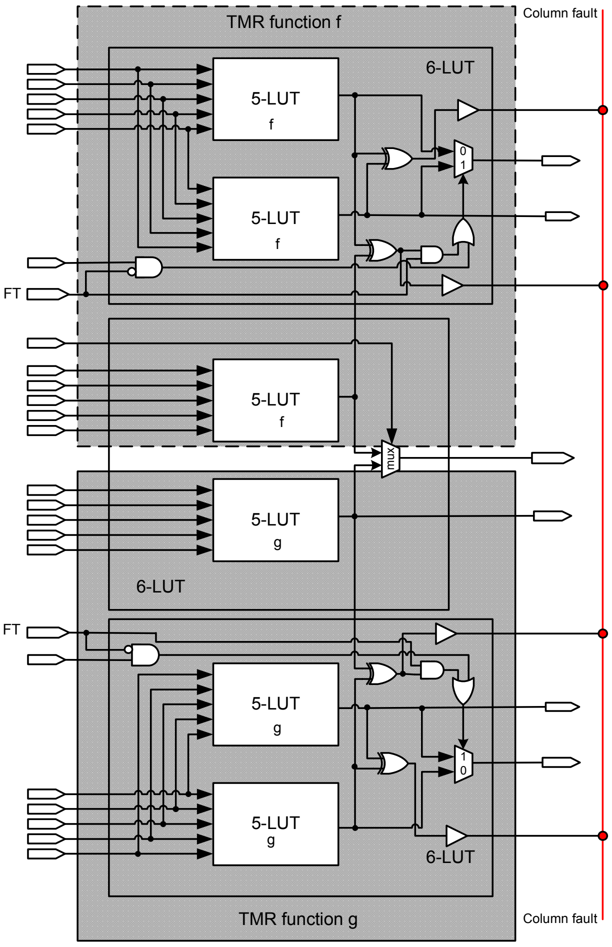

Fig. 4. Tripple-Modular Redundancy in our proposed LUT architecture. An XOR gate compares the first two results and if there is a mismatch the third (guaranteed to be correct by the single fault assumption) result is selected.

The majority voter implementation follows this approach. We take advantage of the multiplexer that exists in the LUT structure to choose one of the two 5-input outputs, and overload its functionality to select the correct TMR output. For function f, assume that we always take the first output (top one in figure) as correct (select 0 in multiplexer). We need to change this choice if a SEU is present in this copy. We determine this condition with the xor gate between 1st and 3rd copy. If the output is one then one of the two copies is wrong. However, we do not need to identify which one has the fault since under the single fault assumption the 2nd copy will be correct. So we select the 2nd output from the multiplexer.

In our system's architecture we want to provide optional TMR support. If TMR is not selected, then multiplexer is controlled directly by the 6th input as it is in the original design (Figure 2). We add an FT signal to select regular or TMR operation for this LUT. We envision this to correspond to a configuration bit. The added gates implement the voter and control the multiplexer, as described above. The three LUTs together behave as a slice that is capable to map: a) three 6-input functions (or up to 6 5-input functions as the original architecture supports), or b) two 5-input functions that use TMR technique.

In addition to the voter gates, an important change in the requirement for independent input signals for the two halves of the middle LUT. This is because they implement copies of different functions. There are two implementation alternatives for this change: (i) extend the interface to add an additional set of (five) inputs, and use the routing logic to connect them to the appropriate signals, and (ii) keep the outer interface unchanged and use two multiplexers to decide if the middle LUT will be used as in TMR or not (Figure 5). Option (i) results in smaller LUT cost but increased wiring and option (ii) leaves the interconnect unchanged but increases the area cost and the LUT latency. As in the DMR case, we use gates to discover the existence of faults and signal the fault's presence at a column granularity. Since the column fault is a high capacity wire, latching buffers should be used to decouple the LUT operation from the (slow) fault flag.

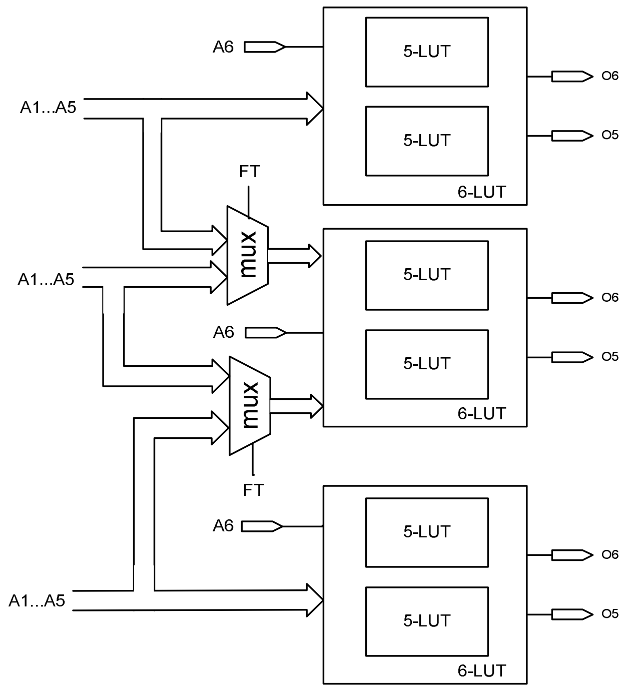

Fig. 5. Slice block diagram. Under TMR the middle LUT implements two copies of independent functions. The MUXes select the independent inputs for regular and copies of the top and bottom inputs for TMR operaion.

The column fault signal can be used in to determine the recovery action. For FPGAs the commonly used technique is a preventive complete reprogramming of the FPGA in regular intervals (scrubbing). The fault detection signal can be used to initiate on demand the reconfiguration, avoiding in this way unnecessary reconfigurations. Even better, the column fault signal can be exploited to initiate *partial* reconfiguration (supported in recent devices) instead of a complete scrubbing, saving additional time and energy.

4. EVALUATION

To evaluate the benefits and cost of our proposed architecture we applied the TMR technique to a set of benchmark circuits. The International Test Conference ITC'99 benchmarks were developed in the CAD Group at Politecnico di Torino as a set of circuits with characteristics typical of synthesized circuits. We used the VHDL descriptions and made the appropriate transforms in order to derive the equivalent TMR circuit.

Our proposed architecture is motivated by the Vitrex-5 LUT structure, so we used this FPGA family in our experiments. To implement TMR we followed Xilinx's TMR design guide [5] that outlines the recommended design methodology for constructing and implementing TMR logic for the Virtex architecture. More specifically, we followed the TMR State-Machines section.

We used Xilinx ISE 10.1 for synthesis and mapping of the VHDL code to a Virtex5 XC5VLX30 target device was the–one of the smallest FPGAs of this family. We synthesized the circuits first using the original source, and then using our TMR version.

Besides triplicating the logic as outlined earlier, the TMR methodology inserts majority voters after flip-flops to vote the state of the FSM. Another voter is needed for each output to choose one of the three copies of the output signals. This overhead logic is the reason why very small designs have a large relative overhead (b01, b02, b10). For larger designs the final TMR area increases by a factor that ranges in between 3 and 4 times. The detailed circuit implementation costs are shown in Table 1.

Table 1. Number of LUTs for the regular and TMR circuit versions.

	no TMR	TMR	TMR overhead
b01	5	30	6x
b02	4	24	6x
b03	38	120	3,16x
b05	174	585	3,36x
b06	8	51	6,38x
b08	22	84	3,82x
b10	35	168	4,8x
b11	105	315	3x
b12	263	885	3,37x
b15	1963	6684	3,40x
Total	**2617**	**8946**	**3,42x**

Finding the space overhead when restricting the mapping to 5-input functions is not straightforward. Unfortunately no Xilinx FPGA with 5-input LUTs is available: families before Virtex 5 use 4-input LUTs. To compute this overhead we used the following technique: first we synthesized each benchmark for a Virtex 4 FPGA that uses 4-input LUTs. Then the derived netlist (ngc file) was imported and implemented for a Virtex 5. The ISE tool offers a mapping option that controls the number of inputs of mapped function. We employed this method three times for four-, five- and six-input LUTs respectively. Table 2 presents the different spatial costs. To verify that this procedure leads to reliable results, we compared results of 4- and 6-input mapping with normal implementation of same designs directly for Virtex 4 and Virtex 5 devices, and found this approach to be accurate.

As we can observe in Table 2, the area cost of restricting the mapping from 6- to 5-input LUTs in most cases is between 10% and 20%, and is always smaller than 40%. These results are similar to the ones reported in the Xilinx Retargeting Guidelines for Virtex-5 white paper [14]. The area cost is also shown in Figure 6 scaled to the cost of 4-input LUTs. The reduction in the bars corresponds to the benefits from using larger LUTs.

Table 2. Circuit cost mapping to 4-, 5- and 6-input LUTs.

	4-input	5-input	6-input	Cost of using 5-input LUTs
b01	12	5	5	0%
b02	4	4	4	0%
b03	81	66	51	29%
b05	250	224	189	19%
b06	9	8	8	0%
b08	37	28	25	12%
b10	52	48	35	37%
b11	151	133	108	23%
b12	412	360	308	17%
b15	3005	2682	2295	17%
Total	**4013**	**3558**	**3028**	**17.5%**

Using these results we can compute the cost of DMR and TMR for our architecture. On average, restricting the mapping to 5-input LUTs increases the average area cost by about 17.5%. Hence, this would be the cost of supporting DMR in our proposed architecture. To find the expected TMR cost, we must compute how many more LUTs we need. As described earlier, implementing TMR for 2 (5-input) LUTs we use a third one to have triple redundancy. So the overall area cost is 1.5 times the cost of the circuit using 5-input LUTs, or 1.5 * 1.175 times the original 6-input mapped design. Therefore supporting fine grain TMR in our proposed architecture increases the area by a factor of 1.76 times compared to at factor of at least 3 in traditional TMR even for coarse granularity. These results are summarized in Table 3.

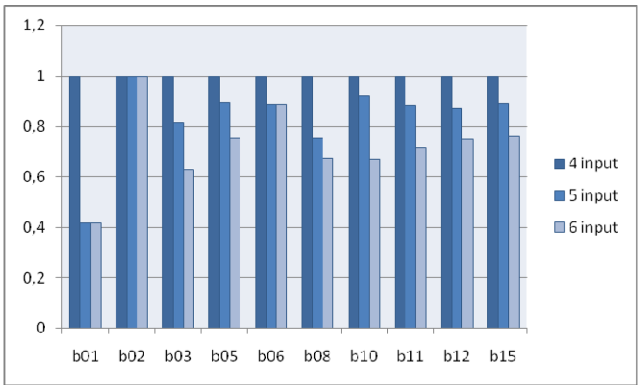

Fig. 6. Area cost for 4-, 5- and 6-input LUTs, scaled to the area for 4-input LUTs.

Table 3. Comparison between high-level and proposed architecture spatial cost.

No fault tolerance	High-level DMR	Our DMR	High-level TMR	Our TMR
1x	2x	1.175x	3.42x	1.76x

5. CONCLUSIONS

In this paper we propose a new LUT structure for efficient TMR support in Virtex-5 like LUT structures. We reuse as much as possible the existing LUT circuitry and augment it to provide TMR support at reduced cost. Our technique achieves significant cost savings for TMR operation at the cost of a few gates overhead per LUT. Since these modifications would be done in full-custom logic, the overhead is very small.

Our approach offers several advantages: it allows the use of DMR with just 17.5% and TMR with 76.5% area overhead compared to 100% and 242% for the traditional implementations. It also provides a fault detection mechanism and column-based fault location, minimizing the number of required reconfigurations, saving both reconfiguration time and energy. It also allows the extensive use of fine grain TMR that offers the best fault tolerance and resolution.

Our work concentrated on LUT configuration faults only. A considerable portion of the FPGAs programming bits refer to the interconnection network, for which this approach is not possible. We are investigating fault manifestation in FPGA interconnects and how we can apply fault tolerant techniques to it.

REFERENCES

[1] M. Alderighi, F. Casini, S. D'Angelo; D. Salvi, G, Sechi, "A fault-tolerant FPGA-based multi-stage interconnection network for space applications", IEEE nt. Workshop on Electronic Design, Test and Applications, pp. 302-6, 2002.

[2] L. Anghel, D. Alexandrescu, M. Nicolaidis, "Evaluation of a Soft Error Tolerance Technique based on Time and/or Hardware Redundancy," Proc. of IEEE Integrated Circuits and Systems Design (SBCCI), Sept. 2000, pp. 37-242.

[3] L. Anghel, M. Nicolaidis, "Cost Reduction and Evaluation of a Temporary Faults Detecting Technique," Proc. 2000 Design Automation and Test in Europe Conference (DATE 00), ACM Press, New York, 2000, pp. 591-598.

[4] M. Berg, "Fault tolerance implementation within SRAM based FPGA designs based upon the increased level of single event upset susceptibility", Int. On-Line Testing Symposium, 2006.

[5] C. Carmichael, "Triple Module Redundancy Design Techniques for Virtex FPGAs", Xilinx Application Note XAPP197, July, 2006. http://www.xilinx.com/support /documentation/application_notes/xapp197.pdf

[6] C. Carmichael, M. Caffrey, A. Salazar, "Correcting Single-Event Upsets Through Virtex Partial Configuration", Xilinx Application Note XAPP216, June, 2000. http://www.xilinx.com/support/documentation/white_papers /wp248.pdf

[7] S. D'Angelo, C. Metra, S. Pastore, A. Pogutz, G. Sechi, "Fault-tolerant voting mechanism and recovery scheme for TMR FPGA-based systems", Int. Symposium on Defect and Fault Tolerance in VLSI Systems, pp. 233-40, 1998.

[8] S. D'Angelo, C. Metra, G. Sechi,, "Transient and permanent fault diagnosis for FPGA-based TMR systems", IEEE Int. Symposium on Defect and Fault Tolerance in VLSI Systems, pp. 330-8, 1999.

[9] R. DeMara, Kening Zhang , "Autonomous FPGA Fault Handling through Competitive Runtime Reconfiguration", ASA/DoD Conference of Evolution Hardware, 2005.

[10] D. Dupont, M. Nicolaidis, P. Rohr, "Embedded Robustness IPs for Transient-Error-Free ICs", IEEE Design and Test of Computers, May-June, 2002, pp. 56-70.

[11] S. Durand, C. Piguet, "FPGA with self-repair capabilities", Int. Workshop on Field Programmable Gate Arrays, p.1-6, 1994.

[12] F. Lima, L. Carro, R. Reis, "Designing fault tolerant systems into SRAM-based FPGAs", Design Automation Conference, pp. 50-655, 2003.

[13] G. Mojoli, D. Salvi, M. Sami, G. Sechi, R. Stefanelli, "KITE: A behavioural approach to fault-tolerance in FPGA-based systems", International Workshop on Defect and Fault Tolerance in VLSI Systems, p 327-334, 1996.

[14] B. Philofsky, "Retargeting Guidelines for Virtex-5 FPGAs", Xilinx White Paper WP248, July 31, 2007. http://www.xilinx.com/support/documentation/white_papers /wp248.pdf

[15] E. Stott, P. Sedcole, P. Cheung, "Fault tolerant methods for reliability in FPGAs," International Conference on Field Programmable Logic and Applications (FPL), 2008, vol., no., pp.415-420, 8-10 Sept. 2008.