

TOWARDS BOUNDED ERROR RECOVERY TIME IN FPGA-BASED TMR CIRCUITS USING DYNAMIC PARTIAL RECONFIGURATION

Ediz Cetin¹, Oliver Diessel², Lingkan Gong², Victor Lai²

¹School of Electrical Engineering & Telecommunications

²School of Computer Science & Engineering

University of New South Wales, Sydney, Australia

email: e.cetin@unsw.edu.au, {odiessel, lingkang, victorl}@cse.unsw.edu.au

ABSTRACT

Field-Programmable Gate Array (FPGA) systems are increasingly susceptible to radiation-induced *Single Event Upsets* (SEUs). Application circuits are most commonly protected from SEUs using *Triple Modular Redundancy* (TMR) and scrubbing to eliminate configuration memory errors. This paper focuses on implementing circuits that recover from SEUs within a specified maximum recovery period, a practical requirement not previously addressed. We develop a recovery time model, describe a scalable reconfiguration control network, and investigate the performance of a representative TMR system implemented using our approach. The results demonstrate that modular reconfiguration eliminate configuration errors more responsively and using less energy than scrubbing. However, these benefits are achieved at the cost of additional area, minor speed penalties, and greater design complexity.

1. INTRODUCTION

The main challenge of using *commercial, off-the-shelf* (COTS) FPGAs for space applications is mitigating the effects of radiation-induced *Single Event Upsets* (SEUs). An SEU occurs when deposited charge causes a change of state in dynamic circuit elements. In FPGAs, SEUs can modify the data being processed as well as the implemented circuits by modifying the configuration memory [1]. Detection and mitigation of the effects of SEUs for space-borne FPGA systems is therefore of paramount importance. With ongoing reduction in transistor feature sizes, FPGA-based terrestrial and airborne applications are also increasingly susceptible to SEUs [2, 3], and thus provision for tolerating SEU-induced faults in these designs should be considered as well.

The predominant technique for protecting COTS FPGAs is *Triple Modular Redundancy* (TMR) [4]. However, TMR on its own does not provide a means of correcting a faulty unit. When a triplicated module contains no feedback paths, SEUs are detected as transient errors in the output. However, when an SEU is trapped in a feedback path, or the con-

figuration memory is affected, errors can persist, and some method for eliminating the error is needed. When an SEU is trapped in a feedback path, the fault can be recovered by a simple yet costly system-wide reset, or by voting on the feedback values [5]. For SEUs that affect the configuration memory, the configuration memory can be “scrubbed” by periodically scanning and completely re-writing its contents while preserving the application’s state [1]. However, this method wastes power when errors are not present, and suffers from undue delays in recovering from errors when an error occurs in logic that has just been scrubbed. Between scrub cycles, multiple SEUs can affect more than one of a triplicated component’s modules; the correct state can then be lost, forcing a rollback to a checkpointed state [6] or an even more costly restart. A more selective approach, which shows promise in overcoming these drawbacks of scrubbing, is to partially reconfigure corrupted TMR modules dynamically, when permanent errors are detected and while the rest of the system continues to operate [7, 8].

Employing *Dynamic Partial Reconfiguration* (DPR) requires the designer to partition the circuit into triplicated components by judiciously inserting voters at the partition boundaries. Kastensmidt *et al.* have been concerned with trading off partition size for area overhead and error rates [8], whereas Bolchini *et al.* have developed a framework for exploring this design space allowing the designer to evaluate the costs and performance of alternative partitioning arrangements [9]. As far as DPR for TMR is concerned, reconfiguration control and associated overheads have only been cursorily mentioned in the literature to date.

This paper trades off area overheads with system recovery time. In particular, larger TMR partitions introduce less area overhead and longer module reconfiguration times, thereby increasing the maximum system recovery time. Our goal is to develop methods for partitioning the system into TMR protected components such that the delay in recovering from a fault is less than a specified maximum recovery time, a practical requirement that has not previously been addressed. The gaps in the literature and the needs of our methodology

have motivated us to develop

- A recovery time model, which includes the time for detecting errors, requesting reconfiguration and re-synchronizing module state,
- A scalable reconfiguration scheme, which involves investigating the performance and overheads of an implemented reconfigurable TMR system, and
- A partitioning algorithm based on these elements.

This paper describes the novel results of the first two tasks of our overall project.

2. PROPOSED SEU DETECTION AND MITIGATION FRAMEWORK

2.1. Recovery Time for Acyclic Components

In a TMR-protected acyclic component, a single transient error will clear itself after a single erroneous cycle since the data processed in the immediately following cycle won't have been affected by the SEU and there is no feedback path for the original error to be stored or circulated. Two or more successive errors occurring in one module in this type of circuit either indicates multiple SEUs occurring on successive clock cycles, or that some damage has been caused to the circuit's configuration. As the latter cause is significantly more likely in FPGA-based circuits, such an event can be used to trigger reconfiguration of the module. The maximum time to detect an error occurs when an error presents itself in the input stage of the component. If the latency or pipeline depth of the component is N and the clock frequency of the component is f_c , then the maximum time to detect an error is $t_{D_max} = N/f_c$.

A request to reconfigure a module must be communicated to the *Reconfiguration Controller* (RC) before it can be acted upon. The RC also needs to inform the component's control circuit when reconfiguration has completed. The total communication delay, t_{C_tot} , depends upon the architecture and layout of the design. In Section 2.3 we describe a reconfiguration control network that we have designed for this purpose.

The time needed to reconfigure the module, t_R , is proportional to its size. This time includes some constant overheads as well as the time needed to read each frame from off-chip memory, check the frame contents for errors, and write it to the destination configuration memory.

The newly reconfigured module cannot be used for checking the correctness of its siblings until it produces the correct outputs. In an acyclic module, simply waiting until the inputs to the newly reconfigured module have been completely processed by the circuit suffices. Using this approach, the time to re-synchronize the module is determined by the maximum number of clock cycles an input needs to emerge at an output – the latency of the component, t_{D_max} .

Including the time needed to re-synchronize the module, the total time needed to detect and recover from a persistent error in an acyclic component therefore is at most $2 \times t_{D_max} + t_{C_tot} + t_R$.

2.2. Recovery Time for Cyclic Components

If a component contains cycles it is not possible to determine whether a persistent fault lies in the datapath registers of the configured circuit or in the configuration memory of the FPGA. Furthermore, if a module contains cycles, the correct state cannot be established simply by allowing inputs to flow through the newly reconfigured circuit. For this reason it is crucial that the circuit to be implemented be partitioned so that *all feedback edges are cut* [5, 10]. Cut feedback edges give rise to an output from and an input to an otherwise acyclic component. That is, the feedback output is passed through the voter to determine whether it is correct or not, and feedback inputs are supplied from the checked outputs of voter circuits. As the resulting component is acyclic, the same recovery time formula applies.

2.3. Reconfiguration Control Network Design

When a large circuit is to be protected in high radiation environments, the circuit may need to be partitioned into a large number of TMR components. A *Reconfiguration Control Network* (RCN) is introduced to manage the requests for reconfiguration that could arise from any voter.

To keep the resources used by this network to a minimum, and to ensure high performance, the RCN is based upon a token ring architecture that connects all voters with the RC in a daisy-chained manner. This architecture ensures a constant port size irrespective of the number of components and ensures few resources are allocated to a supporting function that is not heavily utilized. Only the voter that possesses the token may request a reconfiguration. After issuing the reconfiguration request, the voter holds on to the token until the RC responds that the requested reconfiguration has been completed. If the voter does not require any of its modules to be reconfigured, or when a request it has made has been completed, it simply passes the token on to the next voter around the ring. A voter that does not currently hold the token passes any messages it receives to the next voter around the ring.

The RC responds to reconfiguration requests by determining the off-chip memory address and length of the bitstream associated with the module-ID contained in the request. It then passes the request to load the bitstream to an *Internal Configuration Access Port* (ICAP) controller, which fetches, checks the *Error Correction Code* (ECC) if required, and writes the bitstream to the ICAP port.

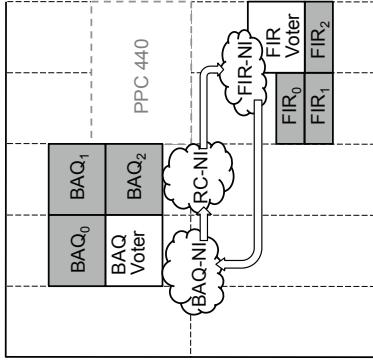


Fig. 1. System layout, shown spanning bottom 10 clock regions of a Xilinx Virtex-5 XC5VFX70T FPGA.

2.4. Partitioning Algorithm

Available FPGA design tools do not readily support the partitioning method, approach to voter insertion, and system layout needed to minimize reconfiguration delays for modules as advocated above. We are therefore currently investigating the design of partitioning algorithms.

2.5. Layout

Xilinx advocates use of PlanAhead to floorplan reconfigurable designs and assist with the production of partial bitstreams. However, PlanAhead is not well suited to the layout of potentially large numbers of reconfigurable modules so as to minimize reconfiguration delays. Our research has only explored manual layouts of small numbers of modules to date. Further study is needed to facilitate the automated implementation of large systems using our method.

3. EVALUATION

The evaluation aimed to gauge the performance and overheads of the proposed approach with representative satellite-based signal processing circuits. To that end, we studied a 16-bit, 21-tap single accumulator *Finite Impulse Response* (FIR) filter and an 8-to-3-bit, 256-sample *Block Adaptive Quantizer* (BAQ) [11] circuit interconnected with the RCN outlined in Section 2.3. The layout of the system, including the arrangement of the reconfigurable modules (shaded), voters, *network interfaces* (NIs) (clouds), and links (arrows) is sketched in Figure 1.

The signal processing circuits were hand coded in Verilog and implemented with ISE and PlanAhead 12.4 tools targeted at a Xilinx Virtex-5 XC5VFX70T speed grade -2 FPGA. We also tested our implementation on an ML507 board with a slower -1 grade FPGA by halving the clock frequencies of individual modules.

Table 1. Area, speed and bitstream length results of mapping FIR and BAQ designs to Xilinx XC5VFX70T-2 device.

Design Name	FB Stages	Utilization			Freq (MHz)/Thrput Msps)	Bit. Len (Wd)	Lat. (CC)
		LUT	FF	DSP			
FIR-1	2	37	84	4	421/16.8		
FIR-3	2	182	225	12	407/16.3		
FIR-3R	2	804	588	12	390/15.6	4100	525
BAQ-1	1	372	261		405/405		
BAQ-3	1	1143	783		384/384		
BAQ-3R	1	1678	1013		333/333	7380	512
RCNode		295	228		125/-		
FlashC		577	598				
TOTAL		3354	2427	12			

3.1. Implementation Results

The FIR and BAQ circuits were implemented without further partitioning. However, the feedback edges were cut to re-synchronize reconfigured modules. To overcome delays in the feedback paths, we added pipeline registers to these and compensated for the additional latency by computing prefix sums of the inputs that used the feedback. In Table 1, results of implementing FIR and BAQ are grouped into sets of 3 rows. In each set, the number of feedback stages and resource utilizations for an optimized, single-module base component (design named with -1 suffix), the conventional triplicated TMR component with associated voter (-3 suffix), and our proposed partially reconfigurable triplicated component, including the associated voter, reconfiguration trigger, re-synchronisation counter and NI (-3R suffix), are listed. Also listed is the clock frequency in MHz and throughput in Msamples/s of each design. The bitstream length in 32-bit words and the latency of the component in clock cycles are listed for the partially reconfigurable implementations. The final set of 3 rows lists the resource utilization for the reconfiguration controller and its NI (RCNode), the Xilinx XPS MCH EMC flash controller (FlashC) used to retrieve stored partial bitstreams, and the total for the system comprising single instances of FIR-3R, BAQ-3R, RCNode and FlashC, as illustrated in Figure 1.

3.2. Reconfiguration Control Network Performance

We simulated the error recovery performance of the system and compared the results with measurements obtained from an instrumented ML507 implementation (see Table 2).

The first 3 rows of the table record the clock frequencies used for the 3 “nodes” of the networked system: the FIR and BAQ components, and the RC. The next 3 rows list the simulated and actual transmission delays for a 7-bit flit on each link of the ring network. Then the estimated and actual delays to recover from a permanent FIR module error are presented.

Table 2. Simulated and implemented reconfiguration control network performance results.

Parameter	Simulation	Implementation
FIR-3R node clock frequency	400 MHz	200 MHz
BAQ-3R node clock frequency	333 MHz	170 MHz
RCNode clock frequency	100 MHz	62.5 MHz
FIR-BAQ flit transmission time	278 ns	535 ns
BAQ-RC flit transmission time	541 ns	1,206 ns
RC-FIR flit transmission time	591 ns	1,104 ns
Initiate reconfiguration request time	1,195 ns	3,340 ns
Transmit reconfiguration request msg.	1,901 ns	4,153 ns
Retrieve bitstream & reconfigure	42,030 ns	1,918,864 ns
Transmit reconfiguration done msg.	591 ns	1,104 ns
Re-synchronization delay	1,311 ns	2,625 ns
TOTAL FIR error recovery time	47 μs	1,930 μs
TOTAL communications time	3.7 μs	8.6 μs

In the third section of the table we first report the time taken to detect an error and to initiate a reconfiguration request after waiting for the token to arrive. Next, the time to transmit and receive a 3-flit reconfiguration request at the RC is given. Messages are wormhole routed, and so this time corresponds to the delay for the first flit to arrive plus the delay of transmitting the remaining flits over the slowest link in the ring. The time to check the validity of the request (just 3 clock cycles) and the time to retrieve and write the module’s bitstream to the ICAP port is then given. Note that the implemented controller is not optimized, while the simulated controller assumes a bitstream word can be retrieved and written each clock cycle. As can be seen, the total error recovery time is dominated by the reconfiguration delay, and that the communications delay of the RCN is relatively small in this case.

3.3. Discussion

Without partitioning our test circuits, we were able to achieve a recovery time of 1.93ms for the FIR circuit (a delay of around 3.4ms could be expected for the BAQ circuit). These delays are substantially less than the maximum error rate of about 1 per second expected for an FPGA approximately twice the size (XC4VLX200), albeit produced with a 90nm rather than a 65nm process, deployed in a geosynchronous orbit. We therefore believe we can meet the practical challenges of recovering from SEUs in space. Significantly, the partial bitstream sizes are respectively about 0.5% and 0.9% of the size of a complete bitstream for an XC5VFX70T device, and therefore require a similar fraction of time and energy to load under ideal circumstances. This potential saving is directly due to the use of partial reconfiguration rather than scrubbing to clear configuration memory errors.

It is also apparent that circuit speed is affected by the use of partial reconfigurability. In our experiments a 5% slowdown was observed for FIR and about 13% for BAQ relative to static TMR for our pipelined implementations.

4. CONCLUSION

We have outlined a framework for implementing FPGA circuits that can recover from configuration memory errors within a desired maximum recovery period.

Our experiments have allowed us to gauge the impact of our framework on the area and speed of two representative user circuits, and to assess the feasibility of our reconfiguration control network. We found that enabling DPR required significant additional area beyond that needed by TMR, as implemented with scrubbing. Small speed penalties were observed but these could be reduced with further development. The overheads of the RCN are tolerable.

Our future work is focussed on developing and assessing partitioning and layout tools to support our framework.

5. REFERENCES

- [1] C. Carmichael, M. Caffrey, and A. Salazar, *Correcting single-event upsets through Virtex partial configuration*, Jun 2000, XAPP216 (v1.0).
- [2] E. Normand, “Single-event effects in avionics,” *Nuclear Science, IEEE Trans on*, vol. 43, no. 2, pp. 461–474, 1996.
- [3] ———, “Single event upset at ground level,” *Nuclear Science, IEEE Trans on*, vol. 43, no. 6, pp. 2742–2750, 1996.
- [4] R. Lyons and W. Vanderkulk, “The use of triple-modular redundancy to improve computer reliability,” *IBM Journal of Research and Development*, vol. 6, no. 2, pp. 200–209, 1962.
- [5] C. Carmichael, “Triple module redundancy design techniques for Virtex FPGAs,” *Xilinx App. Note XAPP197*, vol. 1, 2001.
- [6] A. Sari and M. Psarakis, “Scrubbing-based SEU mitigation approach for systems-on-programmable-chips,” in *Field-Programmable Technology (FPT), 2011 International Conference on*. IEEE, 2011, pp. 1–8.
- [7] C. Bolchini, A. Miele, and D. M. Santambrogio, “TMR and partial dynamic reconfiguration to mitigate SEU faults in FPGAs,” in *IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT’07)*, Sep 2007, pp. 87–95.
- [8] C. Pilotto, J. R. Azambuja, and L. F. Kastensmidt, “Synchronizing triple modular redundant designs in dynamic partial reconfiguration applications,” in *ACM 21st annual symposium on Integrated circuits and system design (SBCCI’08)*, 2008, pp. 199–204.
- [9] C. Bolchini, A. Miele, and C. Sandionigi, “A novel design methodology for implementing reliability-aware systems on SRAM-based FPGAs,” *Computers, IEEE Transactions on*, vol. 60, no. 12, pp. 1744–1758, 2011.
- [10] E. Johnson and M. Wirthlin, “Voter insertion algorithms for FPGA designs using Triple Modular Redundancy,” in *18th annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA’10)*, 2010, pp. 249–258.
- [11] U. Benz, K. Strodl, and A. Moreira, “A comparison of several algorithms for SAR raw data compression,” *IEEE Transactions on Geoscience and Remote Sensing*, vol. 33, no. 5, pp. 1266 – 1276, Sep 1995.