2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines

Reconfiguration Control Networks for TMR Systems with Module-based Recovery

Dimitris Agiakatsikas*, Nguyen T. H. Nguyen*, Zhuoran Zhao*, Tong Wu*,
Ediz Cetin†, Oliver Diessel*, and Lingkan Gong*†
*School of Computer Science and Engineering, UNSW Australia
†School of Electrical Engineering and Telecommunications, UNSW Australia

***Abstract*—Field-Programmable Gate Arrays (FPGAs) provide ideal platforms for meeting the computational requirements of future space-based processing systems. However, FPGAs are susceptible to radiation-induced *Single Event Upsets* (SEUs). Techniques for dynamically reconfiguring corrupted modules of *Triple Modular Redundant* (TMR) components are well known. However, most of these techniques utilize resources that are themselves susceptible to SEUs to transfer reconfiguration requests from the TMR voters to a central reconfiguration controller. This paper evaluates the impact of these *Reconfiguration Control Networks* (RCNs) on the system's reliability and performance. We provide an overview of RCNs reported in the literature and compare them in terms of dependability, scalability and performance. We implemented our designs on a Xilinx Artix-7 FPGA to assess the resulting resource utilization and performance as well as to evaluate their soft error vulnerability using analytical techniques. We show that of the RCN topologies studied, an ICAP-based approach is the most reliable despite having the highest network latency. We also conclude that a module-based recovery approach is less reliable than scrubbing unless the RCN is triplicated and repaired when it suffers configuration memory errors.**

I. Introduction

An FPGA-based TMR system that employs *Module-based configuration memory Error Recovery* (MER) is illustrated in Fig. 1. A *voter* associated with each TMR component identifies which module, if any, is suffering from a persistent fault, and raises a reconfiguration request. Requests from the voters of different TMR components across the device are transmitted through a *Reconfiguration Control Network* (RCN). In Fig. 1, the components of an RCN, which include a *Network Interface* (NI) at each voter, a central *Network Controller* (NC) and an *interconnection network* between them, appear darkly shaded. The RCN identifies which module needs to be reconfigured and sends identifying information to a *Reconfiguration Controller* (RC). The RC fetches the corresponding partial bitstream from *off-chip memory* and reconfigures the faulty module via the configuration fabric, which is accessed through the *Internal Configuration Access Port* (ICAP) present in advanced FPGAs from Xilinx. Analogous hardware is available in Altera FPGAs. After the faulty module has been reconfigured and resynchronized with the remaining two modules of the TMR component, the voter resumes its normal operation.

II. Related work

Several RCN topologies have been described in the literature. These include examples of a point-to-point (P2P) network [1], [2], a bus network [3], a token ring network [4], and an ICAP-based readback approach [5].

This research was supported in part by the Australian Research Council's Linkage (LP140100328) and Discovery (DP150103866) Projects funding schemes.

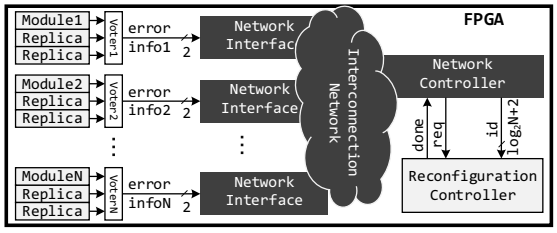

Figure 1. Components of an RCN (darkly shaded) in a TMR system that employs module-based recovery from configuration memory errors.

P2P networks use simple interfaces to connect the voter outputs distributed across the device to the central NC [1], [2]. In P2P networks, the interconnecting wires may need to span the entire device and therefore pass through numerous programmable interconnection resources. This not only increases their susceptibility to SEUs, but also introduces latency.

In [3] the authors implemented a bus network using the *Advanced eXtensible Interface* (AXI) core to transfer the outputs of individual modules to a central voter. While not serving as an RCN, any shared messaging resource, such as this bus, could be used to convey reconfiguration requests from distributed components to a central controller. The use of a shared bus allows new modules to be readily integrated into the system while avoiding the use of the dedicated routing resources found in P2P networks. However, a bus requires more complex interconnection interfaces than P2P networks, which increases soft-error vulnerability, power consumption and latency.

In [4] a token ring network is implemented that spans all voters in a daisy-chained manner. The design uses complex network interfaces that require significantly more logic than the endpoints of the point-to-point connections found in the P2P networks in [1]. However, token ring topologies usually link neighbouring components and therefore utilize a reduced number of global wires for interconnecting them. In contrast, P2P and bus topologies realize mixed distance connections and thus utilize various interconnection resources, including both local and global wires. Usually, SRAM-based FPGAs integrate more local than global wires and therefore token ring networks, which tend to utilize more local wires, are considered to be more scalable than P2P and bus networks. However, in token ring networks, the latency increases with the number of components on the network. A drawback of this topology is that when a link suffers a configuration memory error the ring no longer functions as intended,

978-1-5090-2356-1/16 $31.00 © 2016 IEEE
DOI 10.1109/FCCM.2016.30

whereas the P2P topology is inherently more robust as all links are independent.

A fourth approach that has been described in the literature makes use of the ICAP to read the outputs of the TMR modules [5]. The former uses software to centrally compare the outputs of each module in order to reduce the overheads of distributed voting and to reduce the likelihood of the voter mechanism becoming corrupted. An ICAP-based communications scheme eliminates the need for a soft network and therefore reduces routing pressure, implementation time, and improves reliability. Reliability is enhanced since the built-in hard reconfiguration network is utilized to obtain module or voter outputs. This approach has the potential to be scalable as it does not require user routing resources and utilizes a moderate amount of logic to implement the central controller.

Uniquely, this work compares the prevalent RCN topologies by studying their performance, utilization and reliability when used in a range of synthetic and real TMR systems with a view to aiding the development of systems that are to be deployed in high radiation environments.

III. Reliability Evaluation

In this section, we outline how we model the reliability of a non-triplicated component, the reliability of a TMR component and the reliability of a complete FPGA-based system composed of both non-triplicated and triplicated components. Our analysis is based on the number of critical bits per component for which we use the number of essential bits reported by the vendor's tools as a worst case estimate.

In the following, we assume that the flip of a single essential bit leads to a module failure if the module is not triplicated. With this assumption, the module failure rate λ_m is given by the product of the bit error rate, λ_{bit}, and the number of essential bits in module m. We also assume that the three modules of a TMR component have the same failure rate λ_m. We derived a "high radiation level" of $\lambda_{bit} = 2.7 \times 10^{-10}$ upsets/bit/s for Xilinx 7-series FPGAs [6] in equatorial geosynchronous orbit using the peak 5-min CREME96 model [7] with 2.54 mm aluminium shielding.

We assume that module reliability decreases exponentially over time t as expressed by the function:

$$R_m(t) = e^{-\lambda_m t}, \tag{1}$$

whereby the reliability, $R_m(t)$, of a module at time t denotes the probability that the module operates without any failure in the interval $[0, t]$.

When module m is triplicated, its reliability function becomes:

$$R_m^{TMR}(t) = 3R_m^2(t) - 2R_m^3(t). \tag{2}$$

In order to achieve higher reliability, for a given SEU rate, we employ TMR with MER. The reliability function is then given by [8]:

$$R_{m,r}^{TMR}(t) = \frac{e^{-\frac{1}{2}(at)}\left(a\sinh\left(\frac{bt}{2}\right) + b\cosh\left(\frac{bt}{2}\right)\right)}{b}, \tag{3}$$

where $a = 5\lambda_m + \mu_m$, $b = \sqrt{\lambda_m^2 + 10\lambda_m\mu_m + \mu_m^2}$.

The term μ_m denotes the repair rate of a module, which is the reciprocal of the time needed to recover the faulty module: $\mu_m = \frac{1}{t_{repair}} = \frac{1}{t_d + t_c + t_{sync}} \approx \frac{1}{t_d + t_c}$, where t_d denotes the average error detection time, t_c denotes the error correction time and t_{sync} denotes the synchronization time, which we omit in our case study because it normally only accounts for a small fraction of the recovery time.

The reliability of an FPGA-based system composed of N TMR components that use MER to recover from configuration memory errors and an RCN for aggregating reconfiguration requests can be derived as follows. Depending upon the level of network protection and recovery used, we model the reliability of the RCN $R_{RCN}(t)$ using Eqs (1, 2 or 3). The reliability of the system is then given by the product of the reliability of each individual component, namely the RCN and the N TMR components [9]:

$$R_s^{TMR}(t) = R_{RCN}(t)\prod_{i=1}^{N} R_{i,r}^{TMR}(t). \tag{4}$$

In this derivation, it is assumed that failures follow a Poison distribution and the occurrence of errors in modules or components are statistically independent and uncorrelated. Note that Eq. (4) holds true only if $\mu \gg \lambda$, which ensures repairs are completed independently [9]. Moreover, since we aim to evaluate the impact of various RCN architectures on the total reliability of FPGA designs that incorporate MER, we do not include the reconfiguration controller or the voters in our reliability analysis as these will have the same impact in each case.

IV. Experiments and results

We evaluated the performance of the networks discussed in Section II, in terms of resource utilization, latency, operating frequency, power consumption and soft-error vulnerability. All networks were implemented on a Xilinx Artix-7 XC7A200TFBG484-1 FPGA, as hosted by the RUSH payload [4], using the vendor's Vivado 2014.4 implementation tools with default settings. The comparison of the networks is based on data obtained from the implementation tools.

A. Experiments

As mentioned in Section I, an RCN consists of NIs, a central NC and the interconnection network between them. In our experiments the same voter interfaces and RC were used irrespective of the RCN types being tested. Identical NI and NC locations were used for all RCN designs. In a first experiment we studied "synthetic" layouts in which the TMR components, their voters, and thus the NIs were distributed in a checkerboard pattern across the majority of the device area. Furthermore, the NIs and the NC were located in partitions that utilized the same FPGA resources in each implementation. To obtain resource utilization and performance results, we initially implemented designs that only contained the components of the RCNs being tested and constrained the implementation tools to prevent optimizations across the port interfaces of the NIs and the NC. We tested each RCN type for networks comprising 7, 15 and 31 voters. The synthetic layout of a 31-voter design for testing the P2P network topology is shown in Fig. 2(a).

In a second experiment, we investigated the utilization and performance of each RCN when used to collect reconfiguration requests for the RUSH payload [4]. For this case study, we implemented the four network types with

the 9 TMR components comprising the RUSH experiment. These components include a single MAC-based 21-tap *Finite Impulse Response* (FIR) filter with 16-bit signal width, an 8-to-3-bit *Block Adaptive Quantizer* (BAQ), an 8,096-word deep 32-bit *FIFO*, three 32-bit *Shift Registers* (SRs) having different lengths and a range of combinational logic between the stages and three 32-bit *Binary Search Trees* (BSTs) of different heights and a range of combinational logic at each node. A MicroBlaze processor was used to implement the RC and the AXI HWICAP IP was used to reconfigure faulty modules. The layout of this system is depicted in Fig. 2(b).

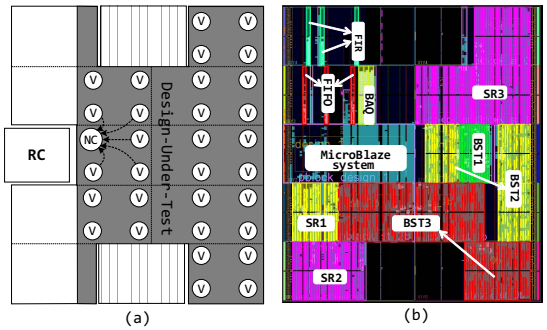

Figure 2. (a) Synthetic layout of a 31-voter design and (b) RUSH floorplan

B. Results

1) Implementation results: Table I presents information extracted from the vendor's implementation tools. Similar designs are grouped together and arranged from left to right with increasing resource utilization. The dynamic power consumption and the number of essential bits follow the same pattern. A relatively small and constant static power consumption was observed, which we believe is due to the relatively few resources (0.2% of total available resources) allocated to the RCNs studied.

The ICAP-based RCN was realized with the fewest resources compared to the other RCN architectures. This is primarily because the ICAP NIs are implemented with just two *Flip Flops* (FFs) and a small amount of support logic mapped to *Look Up Tables* (LUTs). As expected, the number of *Programmable Interconnection Points* (PIPs) and *Switch Matrices* (SMs) used by the ICAP approach is significantly lower than for the other approaches. As a consequence, the ICAP-based RCN has on average 2.7, 3.6 and 6.0 times fewer essential bits than the synthetic layouts of the P2P, bus and ring networks respectively. However, the ICAP-based RCN suffers from high network latency. It requires two to three orders of magnitude more time than the other RCNs to transfer reconfiguration requests to the NC. In contrast, the ring has the lowest latency, since it can achieve a higher operating frequency and only needs 1 clock cycle per node hop. We calculated the average latency for the ICAP, P2P and bus RCNs assuming half the voters are checked before an erroneous one is found. In contrast, we assume a message needs to completely traverse the ring to reach the RC. The latency of the ICAP approach is on average over 175 times that of the ring and the latencies of the P2P and bus networks were about 1.4 times that of the ring over the set of synthetic layouts studied.

Table II
RESULTS OF MAPPING 9 TMR COMPONENTS TO XILINX ARTIX-7 XC7A200TFBG-484

Design	Essential Bits ($\mathbf{n_e}$)	Failure rate ($\lambda_\mathbf{m}$) (upsets/s/module)	$\mathbf{n_f}$	$\mathbf{t_c}$ (ms)
FIR	12,042 (0.02%)	3.25×10^{-6}	65	1.2
FIFO	41,842 (0.07%)	1.13×10^{-5}	192	3.5
BAQ	48,963 (0.08%)	1.32×10^{-5}	73	1.3
BST1	281,604 (0.46%)	7.60×10^{-5}	145	2.6
SR1	285,914 (0.46%)	7.72×10^{-5}	378	6.8
SR2	515,904 (0.84%)	1.39×10^{-5}	474	8.5
BST2	793,534 (1.30%)	2.14×10^{-4}	610	11.0
SR3	1,403,647 (2.30%)	3.79×10^{-4}	1,090	19.6
BST3	1,833,235 (3.00%)	4.95×10^{-4}	1,483	26.7

We investigated an optimization of the ICAP RCN that entails constraining the registers of those groups of NIs that are located within each clock region. These registers are forced to be placed into a single configuration frame so that they can be accessed in a single frame read. With reference to Fig. 2(a), which depicts 4 voters per clock region (10 grey rectangles), this optimization resulted in the creation of horizontal wires leading from each voter to a frame that was centrally located in each clock region. Instead of requiring 31 separate frame reads to check all voters, this approach reduced the number of frame reads needed to 8 in total — one for each clock region used by the design. The results of this implementation are reported in Table I in the ICAP column headed L1*. As can be seen, this optimization reduced the latency of the ICAP approach by a factor of 4 while increasing the number of essential bits used over the unoptimized 31-voter ICAP design by 32%.

2) RUSH case study results: Table II presents the number of essential bits (n_e), the failure rate of each module assuming $\lambda_{bit} = 2.7 \times 10^{-10}$ upsets/bit/s, the number of frames (n_f) and the correction time (t_c) of each TMR module. Note that in our design, since we used the AXI HWICAP, the ICAP throughput using the MicroBlaze was limited to 10 MB/s, considerably less than the maximum possible throughput of 400 MB/s. The reduction in ICAP bandwidth also affected the latency for checking a voter using the ICAP to 60 us per voter, and we observed a much higher network latency.

The system reliability for each RCN type and the 9 RUSH application circuits using Eq. (4) is compared with the reliability of a blind scrub implemented on the same system in Fig. 3. The MicroBlaze RC and off-chip flash configuration storage used by the RUSH payload supports a random FPGA configuration frame read latency of 60 us and a sustained frame write period of 18 us per frame. Blind scrubbing, which entails rewriting each configuration frame of the device, therefore takes 430 ms on the used Artix-7 XC7A200TFBG-484 (24,060 configuration frames), and errors are recovered by scrubbing after 215 ms on average. Please note that in Fig. 3, the scrub plots only account for the 9 application components; they specifically exclude an RCN component, which is not needed for blind scrubbing.

Fig. 3(a) assumes that the four RCNs are implemented as non-triplicated components. While the ICAP RCN results in the best reliability for MER, all 4 RCNs reduce the reliability of the system since they are single points of failure.

Table I
RESULTS OF MAPPING FOUR RCNS TO XILINX ARTIX-7 XC7A200TFBG-484

Type	**ICAP**					**P2P**				**BUS**				**RING**			
Layout	**L1**			**L1***	**L2**	**L1**			**L2**	**L1**			**L2**	**L1**			**L2**
# NIs	7	15	31	31	9	7	15	31	9	7	15	31	9	7	15	31	9
Slices	7	15	31	31	9	12	29	50	18	21	33	60	21	30	50	141	35
LUTs	0	0	0	0	0	14	27	30	16	28	50	108	33	54	130	279	87
FFs	14	30	62	62	18	26	44	77	32	35	61	110	43	61	134	295	87
PIPs	440	889	1,770	1,858	557	1,101	1,996	3,513	1,243	1,341	2,553	4,625	1,729	2,057	3,894	7,986	2,724
SMs	38	62	102	181	55	277	453	792	274	351	616	1074	466	426	496	861	426
Freq. (MHz)	100					112	109	107	126	109	107	104	114	132	203	186	145
Clocks / Hop	230					2				2				1			
# hops	7	15	31	8	9	7	15	31	9	7	15	31	9	8	16	32	10
Latency (us)	8.05	17.25	35.65	9.20	300	0.06	0.14	0.29	0.5	0.06	0.14	0.30	0.5	0.07	0.08	0.18	0.5
Static (mW)	138	138	138	138	138	138	138	138	138	138	138	138	138	138	138	138	138
Dynamic (mW)	3	4	5	5	3	4	7	7	4	6	7	9	5	4	5	8	4
Ess. bits (Kb)	3.42	5.6	10.4	13.7	4.1	9.4	15.8	26.0	10.1	11.9	20.1	38.6	14.9	18.3	33.7	69.4	24.3

L1: Synthetic layout L2: RUSH layout L1*: optimized ICAP layout

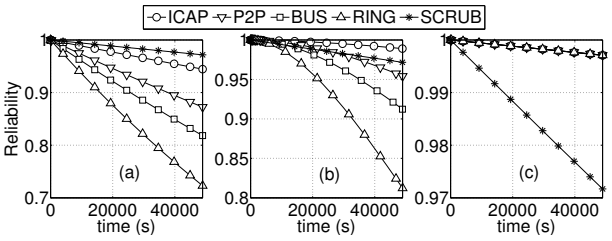

Figure 3. a) Unprotected RCN b) TMR triplicated c) TMR triplicated with recovery

Fig. 3(b) assumes that the RCNs are implemented as triplicated components, and that the errors that occur in this component are not repaired. This provides some limited error mitigation. Only the ICAP outperforms scrubbing over the time period shown. However, eventually (when $t > 165,000$ s) even this approach succumbs to errors that remain unrepaired and scrubbing once again dominates.

In Fig. 3(c) we assume that the device is partially reconfigured in its entirety when an error in the triplicated RCN component is detected. This error recovery period is longer than desired, but the approach ensures any error in the network is corrected. Despite the long recovery time (equivalent to reconfiguring the complete device), the reliability is not significantly affected because errors occur infrequently in the relatively small RCN components.

V. CONCLUDING REMARKS AND FUTURE WORK

In this paper, we have compared four RCN types in terms of reliability, scalability, resource utilization, power consumption and sensitivity to configuration memory errors. The utilization and performance of these RCNs were assessed for networks with 7, 15 and 31 voters. The results demonstrate that the ICAP-based readback approach, which uses the built-in reconfiguration mechanism available in FPGAs, requires the least resources of those networks studied.

The results of a case study that was implemented on the RUSH payload indicate that the ICAP-based readback approach has the highest system reliability but it also has a relatively high latency. This higher latency may not be too problematic except when radiation levels become much higher than the high error rate assumed in our work. We have shown that the latency of the ICAP approach can be reduced by clustering the registers that are to be read from a clock region into a single frame. This optimization does not significantly increase the number of essential bits of the design. We have also determined that for the reliability of MER to be competitive with scrubbing in a real system, the RCN must also be triplicated and be repaired when errors affect it.

One direction for further study is to consider the order in which TMR components are checked. Further work is also envisaged to derive more comprehensive reliability models for complete FPGA-based TMR systems with MER. This work is just the first step in this direction.

REFERENCES

[1] C. Bolchini et al., "A novel design methodology for implementing reliability-aware systems on SRAM-based FPGAs," *IEEE Trans. on Computers*, vol. 60, no. 12, pp. 1744–1758, 2011.

[2] M. Straka et al., "Fault tolerant system design and SEU injection based testing," *Microprocessors and Microsystems, vol. 37, no. 2*, pp. 155–173, 2013.

[3] B. Navas et al., "The upset-fault-observer: a concept for self-healing adaptive fault tolerance," in *AHS*, 2014, pp. 89–96.

[4] E. Cetin et al., "Overview and investigation of SEU detection and recovery approaches for FPGA-based heterogeneous systems," in *FPGAs and Parallel Architectures for Aerospace Applications*, 2016, pp. 33–46.

[5] F. Veljkovic, T. Riesgo, and E. de la Torre, "Adaptive reconfigurable voting for enhanced reliability in medium-grained fault tolerant architectures," in *AHS*, June 2015, pp. 1–8.

[6] D. Hiemstra et al., "Single event upset characterization of the Kintex-7 Field Programmable Gate Array using proton irradiation," in *REDW*, July 2014, pp. 1–4.

[7] A. Tylka et al., "CREME96: A revision of the cosmic ray effects on micro-electronics code," *IEEE Trans. on Nuclear Science*, vol. 44, no. 6, pp. 2150–2160, Dec 1997.

[8] D. McMurtrey et al., "Estimating TMR reliability on FPGAs using Markov models," 2008. [Online]. Available: http://scholarsarchive.byu.edu/facpub/149

[9] M. L. Shooman, *Reliability of Computer Systems and Networks: Fault Tolerance, Analysis and Design*. New York, NY, USA: John Wiley & Sons, Inc., 2002.