

Scrubbing away transients and Jiggling around the permanent: Long survival of FPGA systems through evolutionary self-repair

Miguel Garvie and Adrian Thompson

CCNR, Dept. of Informatics, University of Sussex, Brighton BN1 9QH, UK.

m.m.garvie, adrianth@sussex.ac.uk, +44 (0)1273 872945

Abstract

The Jiggling architecture extending TMR+Scrubbing is shown to mitigate FPGA transient and permanent faults using low overhead. Mission operation is never interrupted. The repair circuitry is sufficiently small that a pair could mutually repair each other. A minimal evolutionary algorithm is used during permanent fault self-repair. Reliability analysis of the studied case shows the system has a 0.99 probability of surviving 17 times the mean time to local permanent fault arrival. Such a system would be 0.99 probable to survive 100 years with one fault every 6 years.

1 Introduction

Reconfigurable hardware devices such as Field Programmable Gate Arrays (FPGA) are being increasingly used in space applications because they allow cheap and fast production of prototypes and final designs in low volume. Buggy designs can be fixed post-deployment, and the same hardware used to perform various tasks – some possibly unforeseen – over the duration of a mission. The use of Commercial Off-The-Shelf (COTS) components such as FPGAs is becoming common-place in space applications and other industries.

SRAM based FPGAs deployed in space are susceptible to radiation hazards, most commonly [14, 3, 10] Single Event Upsets (SEU) not causing permanent damage. Total dose exposure may cause catastrophic damage [10]. The above research does not focus on local permanent damage (LPD).

LPD has not been commonly observed in radiation tested FPGAs, but there are several reasons why it should not be ignored. Cases of Single Event Latch-up, which may cause LPD by inducing high current density, have been reported [9]. Some SEUs cannot be mitigated without a full chip reset which may not be possible for a mission-critical module, thus manifesting themselves as LPD. Radiation testing on earth is not 100% faithful to space conditions and does not last as long as a mission. In fact, no FPGA has been tested for more than 15 years, while NASA plans 100 year missions for deep space exploration. Long device usage could lead to LPD through electromigration or other aging effects. Other dormant faults may only manifest themselves a considerable time after deployment. It would be unwise engineering to assume that LPD to FPGA cells would not occur during long space missions exposed to extreme environmental conditions and radiation. Space missions are not the only deployments that can benefit from strategies dealing with LPD, although they are particularly needy of autonomous onboard repair since communication with earth is low bandwidth and high latency. Radiation and aging effects are also encountered at sea-level and may be a problem for inaccessible systems where component replacement is not feasible.

Triple Module Redundancy (TMR) is currently widely

used to mitigate faults and are considered to have “saved” several space missions. A TMR system has three copies of a module and uses a voting system on their outputs so that the final output is an agreement between at least two modules. A TMR/Simplex defaults to a single module once one module fails, thereby increasing reliability. TMR+Scrubbing [3, 10] provides fault tolerance as above and wipes out SEUs in FPGA configuration data by regular reprogramming. Configuration readback [3] is able to locate configuration errors and fix them by partial reconfiguration. These schemes are only as good as a TMR system in the presence of permanent faults and rely on a *golden* (unbreakable) memory to store configuration data. Latched user data in sequential designs can be protected with state recovery schemes [15].

Lach *et al.* [4] proposed a tile based approach for reconfiguring an FPGA design to avoid LPD. This approach tolerates limited faults per tile, requires a *golden* memory holding precompiled configurations and a *golden* fault location mechanism. The repair mechanism requires tiles to be off-line during reconfiguration, which may rule out repair of mission-critical modules. A similar approach [16] likewise requires a set of *golden* pre-compiled configurations and a *golden* fault diagnosis system hosted on an extra FPGA. The Roving STARS [1] approach proposes a self-testing column and row to shift itself across the FPGA. Fault detection latency is of around 4000 cycles, and it requires constant reconfiguration of the FPGA and is therefore a constant power drain. It relies on a *golden* micro-processor to perform timing analysis, fault location, place and route, and more than 420K of *golden* memory for storage of designs and faults. A final cost is that the system clock is stopped regularly.

Embryonics [7] is a biologically inspired approach with an architecture requiring large amounts of overhead including *golden* copies of chip configuration. Zebulum *et al.* [17] have used a Genetic Algorithm (GA) to repair analog designs on a Field Programmable Transistor Array. They provide results for a restricted fault set and assume a *golden* ‘SABLES’ system composed of a DSP, memory for a full population, and a fitness evaluation mechanism in some cases requiring a healthy copy of the circuit being repaired. [11, 13, 6] apply GAs to repair FPGA designs assuming a *golden* GA module with a full population and fitness evaluation mechanism.

Most FPGA LPD mitigation techniques mentioned so far suffer from the *Repairing the Repairer* dilemma in which a new single point of failure assumed unbreakable is introduced in the mitigation mechanism. This is especially awkward when the mitigation mechanism assumed unbreakable is larger than the breakable system itself. A repair module small and simple enough to be itself repaired, would offer an obvious advantage. This paper will describe such a low overhead fault mitigation mechanism sufficiently small

that a pair could mutually repair each other, thus relying on no golden single point of failure. Section 2 will introduce the TMR+Lazy Scrubbing+Jiggling architecture for transient and permanent fault mitigation, section 3 will lay out a reliability analysis and section 4 will provide some conclusions.

2 TMR + Lazy Scrubbing + Jiggling

The proposed mechanism extends TMR + Scrubbing. TMR provides fault tolerance keeping the system on-line at all times and also votes out SEUs affecting user data. ‘Lazy Scrubbing’ mitigates SEUs to configuration data, and ‘Jiggling’ repairs LPD by using the two healthy modules to repair the faulty one. Once a Jiggling repair is complete three healthy modules are again available (Fig.4). Permanent faults can be repaired until spare resources are exhausted.

After §2.1, the paper deals entirely with permanent fault mitigation of combinational circuits through Jiggling, sequential ones are left for future work.

2.1 Transient SEU Mitigation through Lazy Scrubbing

Traditional Scrubbing reconfigures a whole TMR system on an FPGA regularly from an external memory. To reduce power consumption: a module will only be reconfigured when its output is different to the other two. Instead of requiring a golden memory holding module configuration, the configuration is read from all three modules and a majority vote is taken of this data (taking offsets into account) to reconfigure the faulty module. Lazy Scrubbing requires less power, overhead and single point of failures than traditional Scrubbing. Lazy Scrubbing cannot be used after the first Jiggling repair. Jiggling will still recover from SEUs in FPGA configuration, although with a higher latency.

2.2 Jiggling Architecture Overview

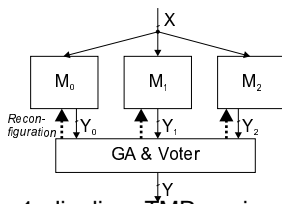


Fig. 1: Jiggling: TMR + minimal GA for repair.

TMR can be applied at many levels. Based on experiments to date, M should be under a thousand gate equivalents to make the repair process feasible. One GA circuit could service many TMR systems, all residing on a single FPGA.

A module is considered to have LPD if it fails to repair after Lazy Scrubbing. At this point, Jiggling is initiated. Jiggling uses the remaining functional modules as a template of desired behaviour to guide a (1+1) Evolutionary Strategy (ES) [8] – the minimal expression of a GA – towards a functional configuration which avoids or exploits the faulty element. The system is kept on-line during this repair process by the two healthy modules driving the majority voter. Spare resources are allocated in each module. Mutations inserted by the GA are single bit flips in the FPGA configuration stream of the faulty module.

Given that permanent faults in FPGAs are not very frequent (indeed completely ignored by Xilinx [5]) we can trade overhead and the knowledge contained therein for time and

allow blind variation and selection find the knowledge required to complete the repair.

Section 2.3 will describe the Jiggling mechanism in greater detail and section 2.4 will cover its hardware implementation. Section 2.5 will describe the simulator used to collect repair time statistics while section 2.6 will present the probability model used for availability analysis.

2.3 Specification

2.3.1 Evolutionary Algorithm used during Repair

A (1+1) ES has one elite individual and one mutated version of it. If the mutant is equal or superior it will replace the elite. Otherwise, another mutant is generated. This strategy has been applied successfully to hardware evolution [12] and has been considered [2] to be an effective strategy for exploring fitness landscapes with high neutrality, such as those of digital circuits. It was mainly chosen here for its simplicity and allowance of a small hardware implementation.

There may not be a single mutation restoring healthy behaviour to a damaged module, so an exhaustive search would blindly iterate over all 2^b configurations, where b is their length. This is not practical even for the small circuit tackled in this work where $b=1058$. A (1+1) ES is a hill-climbing random walk within configuration space moving to a fitter (see §2.3.3) or equivalent configuration at each step. This is not guaranteed to find a solution in time. But faced with a stochastic fault source, no system is capable of guaranteeing survival. The probability of survival for a Jiggling system is studied in §3.

A (1+1) ES is brittle in the presence of noise because if a bad mutant gets a lucky high fitness it could replace a superior elite which would then be lost. Noise is present in the evaluation of circuits on an FPGA when they are not constrained to be combinational. Sequential circuits behave differently under changes in input pattern ordering and gate delays which may vary with environmental conditions. To discourage evolution from discarding a good elite, a *History Window* method is used: the last H accepted mutations are stored so that if the current elite’s fitness is lower than the previous one’s, all H mutations are reverted. By rolling back after encountering individuals with noisy fitness, evolution is discouraged from exploring areas of the fitness landscape encoding sequential circuits. If p is the probability of a lucky high noisy evaluation, then the probability a sequential circuit has been lucky H times varies as p^H . Thus, the larger H is, the higher the chance that the circuit reverted to is stable.

2.3.2 Reconfiguration: Hardware Nature of Mutations

Mutations are single bit flips in the configuration stream of the faulty module M_f . Modules are always allocated 2^C addresses of configurable logic blocks (CLB) so that the address of a mutated block can be simply a randomly generated C bit number. If a circuit only requires a fraction of these CLBs, the rest are allocated as spare resources for repair. A mutation may affect any part of the CLB such as a Look-up Table (LUT) or the routing. Ease of partial self-reconfiguration and robustness to random configuration bit flips make some FPGA architectures more suitable than others.

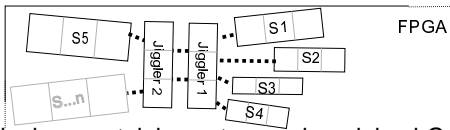


Fig. 2: Jigglers containing voters and a minimal GA are able to service multiple systems including other Jigglers.

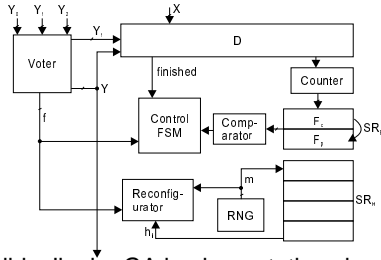


Fig. 3: Possible Jiggler GA implementation showing data flow directed by the Control FSM as in the algorithm in §2.4.1.

2.3.3 Fitness Evaluation

Since normal circuit operation is not interrupted during repair of M_f , circuit fitness evaluation is done on the fly using the inputs being applied during normal mission operation. A score c_{qv} is collected for each output q under every input vector v until all vectors have been encountered. If output q of M_f was ever incorrect under the application of v , then $c_{qv}=0$; otherwise $c_{qv}=1$. Output is considered incorrect if different to the corresponding output at the voter. Fitness score for the current configuration is simply the sum of scores. This fitness score will guide evolution towards configurations whose behaviour is increasingly similar to the healthy modules’.

2.3.4 Repairing the Repairer

Since the repair module is small, it would be feasible to be repaired with the same strategy, once this method has been adapted to sequential circuits. Each repair module could be itself tripled as in Fig.2 and repaired by another repair module. Both repair modules would then be in charge of repairing multiple systems.

2.4 Hardware Implementation

2.4.1 Jiggling Repair Cycle

The (1+1) ES with History Window control loop for repairing faulty module M_f is described below in pseudocode. F_c and F_p are registers holding the current and previous fitness values. D is a bitwise storage holding c_{qv} scores.

1. Set register $F_p = 0$.
2. *Evaluate Elite*: collect c_{qv} scores into D until all input vectors have been encountered. If all scores are 1 then stop repair process.
3. Count number of 1s in D and store it in the F_c register.
4. If $F_c < F_p$ revert all mutations in history shift register SR_H and go to step 1.
5. Shift the value of F_c up into F_p .
6. Insert new random mutation m in M_f .
7. *Evaluate Mutant*: collect scores as in step 2. If all scores are 1 then stop repair.
8. Count number of 1s in D and store it in the F_c register.
9. If $F_c < F_p$ revert mutation m .
10. Else shift the value of F_c up into F_p and push m onto SR_H .
11. Go to step 2.

Control logic can be implemented as a Finite State Machine (FSM) with eight states. If an incorrect configuration gets a lucky full score, repair will be resumed as soon as its behaviour is different from the voter.

2.4.2 Structure

Figure 3 shows a possible hardware implementation for the Jiggler. The voter provides system fault tolerant output Y as the majority of the module outputs Y_0, Y_1, Y_2 . It also provides faulty module index f and output Y_f to the minimal GA. A shift register chain SR_F of size two will hold F_c and F_p . A Counter module will sum vector scores from D into F_c . A Comparator will check if $F_c < F_p$. A shift register chain SR_H of size H will store the mutation history window with the addresses of the last H mutated configuration stream bits. A Random Number Generator (RNG) will generate the random mutation address m . A reconfiguration unit will flip the configuration stream at a particular address and initiate the partial reconfiguration procedure. Its operation will depend on the reconfiguration mechanism of the FPGA architecture chosen.

2.4.3 Overhead Analysis

Given a circuit has I inputs and Q outputs, we need $2^I \times (1 + Q)$ bits storage for D and $2 \times I$ bits for SR_F . If the the address offset of a configuration bit within a module requires A bits, we need $H \times A$ bits storage for SR_H . The control module can be implemented as an 8 state FSM and might require 3 latches and 10 four-input LUTs. The A bit RNG could be implemented as a linear feedback shift register with roughly $A/18$ LUTs and A latches. The reconfiguration module’s size would depend on the FPGA architecture and could vary between 3 and 30 LUTs and some latches. For this analysis it is assumed it requires 15 LUTs and 15 latches. The voter could be implemented in roughly Q LUTs. Given a CLB offset address within a module needs C bits we need $Q \times C$ bits for the reconfigurable module output addresses.

If $I=5, Q=10, H=8, C=5, A=11$ the overhead is $352+10+88+50=500$ storage bits plus roughly $10+1+15+10=36$ LUTs and $3+11+15=29$ latches. §2.3.4 describes a scheme for mitigating faults in this logic.

For larger circuits, the size of D will dominate the sum because it grows exponentially with the number of inputs. It should be noted that D will usually be smaller than a whole extra module and one Jiggler unit is capable of servicing multiple subsystems thus reducing overhead per subsystem.

2.5 Simulated Jiggling

The Jiggling method was evaluated by collecting repair time statistics from a simulated model.

2.5.1 FPGA model

Various FPGA architectures, some of which have been deployed in space missions, can be simplified to a model where each CLB holds one LUT and one D-Latch. Routing between such CLBs is limited yet can be assumed universal [6] for small circuits such as those dealt with in this work. This first study of the Jiggling approach tackles combinational circuits only, so the FPGA model adopted uses four-input LUTs and no latches. We assume it is not complex to turn off all latch functionality for a given area of an FPGA.

2.5.2 Simulator Characteristics

The simulator used is a simple version of an event driven digital logic simulator in which each logic unit is in charge of its own behaviour when given discrete time-slices and the state of its inputs. Routing is considered unlimited so any unit can be connected to any other allowing recurrent connections inducing sequential behaviour, so care must be taken to update

all units ‘simultaneously’. This is achieved by sending the time-slices to the logic units in two waves: the first to read their inputs and the second to update their outputs. During each evaluation, circuit inputs were kept stable for 30 time-slices and the outputs were read during the 5 last time slices.

Gate delays are simulated in time-slice units and are randomized at the start of each evaluation with a Gaussian distribution with $\sigma = 0.1$ and a μ varying between 3 and 6 thus simulating a probe subjected to a changing environment. μ increments, decrements or keeps its value within the [3, 6] range every t_d simulated generations where t_d is itself taken from a Gaussian distribution ($\mu = 270000, \sigma = 90000$). For the scenario studied in this paper these statistics would translate to the mean of the frequently randomized gate delays changing roughly every minute.

The Stuck-At (SA) fault model was chosen as an industry standard providing a fairly accurate model of the effects of radiation hazards at the gate level. SA faults can be introduced at any of the logic units of the simulator simply by setting its output always to 0 or 1.

2.5.3 FPGA configuration stream encoding

As mentioned earlier, mutations are performed on the section of the FPGA configuration stream which encodes the faulty module. During simulated evolutionary repair, the GA deals with linear bit string genotypes which are equivalent to the simulated FPGA configuration stream. As mentioned in §2.3.2, there are 2^C addresses available. The last I addresses are assigned to circuit inputs while the remaining $2^C - I$ refer to LUT units within the module. C bits are required per address. The first $Q \times C$ configuration bits – where Q is the number of circuit outputs – encode the addresses from which module outputs will be read by the voter. This simulates mutations to the configuration memory controlling routing to the module outputs. The rest of the stream is divided into $2^C - I$ sections, one for each LUT. Each of these sections contains 16 bits encoding the LUT and $4 \times C$ bits encoding where the inputs of this LUT are routed from. LUT addresses are assigned in the order they appear in the configuration stream.

2.5.4 Evaluation Procedure

In order to mimic normal mission operation during circuit evaluation, each test vector applied for the 30 simulated time step presentation is randomized. All analysis in this paper assumes the availability of a fresh input on every clock cycle. Evaluation ends when every test vector has been encountered. For each input vector, the number of circuit output bits that were always correct during its application, is added to the total fitness score.

2.5.5 Collecting Repair Time Statistics

Repair time information is required to analyse the reliability of the Jiggling approach. Since all modules are equal repair statistics from a single module convey the information required. The time taken to repair the first fault, when all spares are available, may be different to that of repairing the n^{th} when n faults are present in the module and $n-1$ successful repairs have already taken place probably allocating at least $n-1$ spares.

To collect repair time information, W random fault sequences of length s – the initial number of spare LUTs – are generated making sure no LUT is failed twice. For each of these W sequences, faults are inserted into the simulated

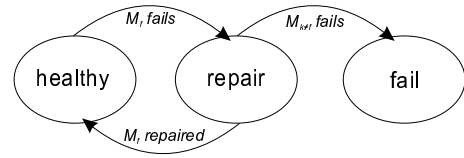


Fig. 4: Markov model of the Jiggling life cycle.

module in order and the number of generations of evolution required to arrive at a fully functional configuration is recorded. H consecutive generations with a fully functional elite must elapse before the module is considered repaired to avoid lucky solutions. The repair time of the i^{th} fault in the j^{th} sequence will be referred to as r_{ij} . If r_{ij} exceeds 4.32 million (M) simulated generations the fault sequence is aborted and the repair time of all unrepaired faults in the sequence is set to ∞ , so $\forall a \geq i. r_{aj} = \infty$. This limits the amount of CPU time used for simulation and will significantly skew the statistics pessimistically since long-term missions may have months to repair a permanent fault.

2.6 Probability Model

Figure 4 shows a Markov model with the three states the Jiggling system can be in. It begins its life fault-free in the healthy state. When a fault arrives at module M_f it moves to the repair state. If during repair another fault arrives in M_f it will stay in this state. If during repair a fault arrives in one of the other modules it reaches the fail state. The system will only be considered to go back to the healthy state when M_f is repaired before a fault arrives at $M_{k \neq f}$.

The probability of moving from healthy to repair is dictated by the permanent fault rate which may be affected by such factors as usage, age and environmental stress. The probability of moving from repair back to healthy will depend on the permanent fault rate and on time to repair which is likely to increase as faults accumulate and spares get used.

Faults affecting the voter and GA module could be dealt with as mentioned in §2.3.4.

2.6.1 Availability Analysis

The Jiggling system is considered to fail when, during repair of M_f a fault arrives at $M_{k \neq f}$. Consider P_i the probability of the i^{th} repair within the whole system (after the i^{th} fault at any module) succeeding. This is the probability that, given $i-1$ system repairs were successful, the latest fault at M_f will be repaired before a fault arrives at $M_{k \neq f}$. Given a fault rate λ per unit area and a total area A for the three modules, the Poisson distribution of permanent local fault arrival during a time period t has parameter λAt . The probability that the system has not failed before time t , ie. its reliability is:

$$R(t) = \sum_{n=0}^{\infty} \left(\frac{(\lambda At)^n e^{-\lambda At}}{n!} \prod_{i=1}^n P_i \right) \quad (1)$$

The outer sum considers all possible fault counts n during t . For each n the probability of n faults occurring during t is multiplied by the probability of all n system repairs succeeding. If p_i is the probability of the i^{th} repair on a single module succeeding, the probability P_i of the i^{th} system repair succeeding can be calculated:

$$P_i = \sum_{r=0}^{i-1} p_{r+1} \left(\frac{1}{3} \right)^r \left(\frac{2}{3} \right)^{k-r} k C_r \quad (2)$$

Table 1: Time r_{ij} in thousands of generations taken to repair the i^{th} fault for each of the $j=1\dots W$ fault sequences.

$j \setminus i$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	>15
1	0	0	0	0	0	0	229	197	942	1413	0	∞	∞	∞	∞
2	0	0	0	11	43	60	2059	97	2	387	606	1463	4226	∞	∞
3	0	0	0	316	1922	409	320	319	62	223	2968	∞	∞	∞	∞
4	0	1616	1	37	79	0	1376	1160	657	3097	0	681	394	∞	∞
5	0	0	5	2	25	3	0	79	147	1136	80	935	2908	1661	∞
6	15	78	139	319	1246	131	151	53	1009	124	1986	1456	2751	∞	∞
7	0	0	324	30	328	473	41	1153	228	355	77	59	∞	∞	∞
8	0	752	25	322	0	329	779	29	132	96	0	298	15	∞	∞
9	0	0	0	0	0	193	228	319	451	185	770	424	3689	4107	∞
10	684	0	303	78	58	0	1311	417	199	193	2137	∞	∞	∞	∞

where $P_0=1$. The sum considers all possible previous fault counts r at M_f . For each r the probability of the $(r+1)^{\text{th}}$ single module repair succeeding is multiplied by the probability of r faults occurring at M_f out of $i-1$ system faults.

2.6.2 Calculating p_i

Recall p_i is the probability of the i^{th} repair on a single module finishing before a fault arrives at one of the other two modules. Given that fault arrival is modelled as a Poisson process, fault interarrival time T follows an exponential distribution, such that the probability of the next fault in any of two modules occurring later than time t_r is $P(T \geq t_r) = e^{-2A_M \lambda t_r}$, where $A_M = \frac{A}{3}$ is module area. Given a set of W single module repair times for the i^{th} repair r_{ij} , the probability p_{ij} of each of these repairs succeeding is $P(T \geq r_{ij})$. Provided with a limited sample of W values of r_{ij} for each i the best estimate of p_i using the frequency interpretation of probability will be: $p_i = \frac{1}{W} \sum_{j=1}^W p_{ij}$. Given the equations above, the experimental data r_{ij} collected as in §2.5.5 can be used to calculate reliability $R(t)$.

2.6.3 TMR/Simplex

The Jiggling system reliability under permanent faults will be compared to TMR/Simplex. In this papers' experiments 180% spares were allocated over mission LUTs so the Jiggling module area A_M is 2.8 times as large as the TMR/Simplex module area A_{TM} . The reliability of a

TMR/Simplex system is: $R_{TMR}(t) = \frac{3e^{-\frac{A_{TM}\lambda t}{3}}}{2} - \frac{e^{-A_{TM}\lambda t}}{2}$. This ignores faults at the voter (a single point of failure), making the comparison conservative.

3 Results

Repair time statistics were collected as in §2.5.5 for the cm42a combinational benchmark from the MCNC '91 test suite. This circuit has four inputs and ten outputs requiring 10 four-input LUTs. The number of bits per address C was chosen to be 5 so 32 addresses were available, four of which were allocated to circuit inputs (§2.5.3). Thus 18 spare LUTs were available in the module being repaired. History window size H was set to 8. The number of generated fault sequences W was chosen to be 10. An average 53.6 random input test vectors must be applied during evaluation until all 16 possible four-input vectors have been encountered. 20 clock cycles would be sufficient between evaluations to count fitness, swap a bit in the configuration stream and perform the required control procedures. At a very modest clock speed of 1MHz both module configurations (individuals) in a generation could be evaluated in $\frac{73.6 \times 2}{10^6} = 1.472 \times 10^{-4}$ seconds and 6793 generations could be completed per second. With a timeout of 4.32M generations the simulator gave less than 11 minutes for circuits to repair from permanent damage.

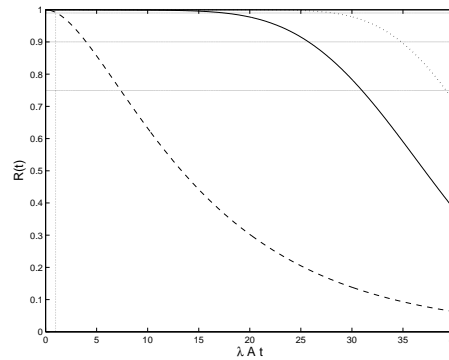


Fig. 5: Reliability vs. time for TMR/Simplex (dashed), Jiggling (solid) and Jiggling with every repair successful (dotted).

Once r_{ij} were collected (Table 1), p_{ij} , p_i and P_i were calculated, providing $R(t)$ through the equations in §2.6. Repair times were short requiring on average under two minutes of simulated time. 94 of the 104 evolutionary searches starting from a misfunctional configuration arrived at a healthy one within the simulated 11 minutes. In the 10 cases of unsuccessful repair there was on average 14 faults present in the module under repair. Under the stochastic fault source and tight time constraints, the repair mechanism exploited on average up to 78% of spare resources.

Figure 5 compares the resulting reliability of the Jiggling system against a TMR/Simplex approach whose modules are almost 3 times smaller. Since repair times are either necessarily low or ∞ due to the 4.32M timeout, the repair clock speed does not influence reliability and the analysis only depends on the fault rate. The time scale is in terms of λA which is the local permanent fault arrival rate at any module in the Jiggling system. The fault rate at any module in the TMR/Simplex system is $\frac{\lambda A}{2.8}$. The mean time between fault arrivals $t=(\lambda A)^{-1}$ is marked together with the 0.75, 0.9 and 0.99 $R(t)$ levels. With a λA of 1 fault a day the TMR/Simplex system has a reliability lower than 0.75 after a week yet the Jiggling system still has 0.99 reliability after 17 days and is still above 0.75 up to a 31 day month. With a fault rate of $1/t$ (mean time between arrivals t) the Jiggling system will be 0.99 reliable after $17t$, so a fault rate of 1 every 6 years would be tolerated for 100 years with 0.99 probability.

3.1 Discussion

The reliability of the Jiggling system studied in this paper has been skewed negatively by the following factors. A fault in one of the modules not being repaired does not necessarily lead to system failure. Firstly, and specially relevant when spare count exceeds mission logic count, faults may arrive at spares of these modules. Secondly, two modules each carrying a fault may produce erroneous outputs for disjoint sets of inputs (ie. they never produce a false output at the same time). In this case there is enough information at the voter output to repair *both* modules, returning to the healthy state. Finally the 4.32M generation limit equates to under 11 minutes repair time at 1MHz and under 0.66 seconds at 1GHz of repair time and thus ignores a huge amount of possibilities for repair. Since permanent damage is most likely to happen every 6 months at worst, the timeout heavily skewed statistics negatively. With an extremely high permanent fault rate of one per day there would be time to complete 587M generations at 1Mhz within the mean fault interarrival time. Informal investigations suggest that with such a limit, repair would

succeed at nearly every opportunity and the system would repair up to the number of spares available making reliability more like the dotted line in fig. 5. Thus each spare added to a module would lengthen its life by roughly the fault interarrival time. Adding spares gives diminishing returns because of the increased area, but it remains possible to compute the number of spares required to provide a desired reliability at a specific point in the mission.

This work assumes the circuit being repaired is being used for normal operation and its input is effectively randomised at every clock cycle. If the circuit were not in use then it could be supplied inputs artificially, possibly from the RNG or the counter. This is also necessary if the full set of input vectors is not frequently encountered during normal operation. If the normal mission input pattern is not uniformly random and its characteristics are known, these could be used during simulated repair to generate the appropriate reliability figures.

Once the first permanent local fault is repaired through Jiggling, Lazy Scrubbing §2.1 must be disabled since one of the module configurations has changed. However transient faults to configuration bits could still be repaired through Jiggling by a mutation hitting the configuration stream bit affected by the SEU. This could be accelerated with minimum extra hardware by using a counter to flip each bit in the configuration stream in turn, until the SEU was undone. Further study is required to analyze the repair statistics under transient faults combined with permanent faults potentially doing away with the need for Scrubbing altogether.

This preliminary case study evaluated the Jiggling system repairing a design smaller than itself and increasing its reliability. The high probability of repair before spares are exhausted suggests the method could be applied to larger benchmarks. Unpublished related experiments suggest that evolutionary algorithms are capable of finding solutions in spaces as large as 2^{10000} , although with a significantly longer search time. Given that the average repair time for the studied benchmark is under two minutes and that permanent local fault interarrival time is likely to be over 6 months, a larger benchmark could have repair times several orders of magnitude larger and still achieve similar reliability figures. Each of the W fault sequences tested for repairing the cm42a benchmark took about 1 day on a 1.4GHz processor. The amount of processing power required will vary exponentially with benchmark size due to increased simulation time per configuration and longer evolutionary search.

As Jiggling is evaluated for larger benchmarks and as the fault model used is made more realistic, the possibility of using real hardware is more attractive. The Jiggling system could be implemented on a Xilinx FPGA subjected to radiation. This would give us accurate overhead measures as well as more accurate reliability figures since all cases mentioned above would be taken into account in the real system and generations could be truly performed at millions a second.

4 Conclusion

The TMR+Lazy Scrubbing+Jiggling approach to FPGA transient and permanent fault mitigation has been introduced. Lazy Scrubbing mitigates transients with less power and overhead than traditional Scrubbing and does not depend on a *golden* memory with configuration data. Jiggling self-repairs a system subjected to permanent faults requiring no *golden* memory for holding pre-compiled configurations nor an external *golden* microprocessor. The Jiggling repair module is implemented on the same FPGA and for the case studied re-

quires 500 storage bits, 36 LUTs and 29 latches. This is small enough to be itself repairable by another Jiggler thus removing all single points of failure, provided the architecture is extended to sequential circuits. This repair module would also service multiple systems, amortising the overhead.

Reliability analysis for a small benchmark shows the Jiggling system using 2.8 times the overhead per module can survive with 0.99 probability: 17 times longer than a TMR/Simplex approach. This analysis takes into account both the stochastic natures of evolution and of the fault source. It is shown how the number of spares in such a system can be adapted to reach desired reliability guarantees for specific times during a mission.

A more thorough evaluation of this architecture would involve larger benchmarks and more accurate statistics. The latter may be achieved by a less pessimistic probability model and by allowing more time for simulated repairs. As the size of benchmarks renders simulation cost prohibitive, the Jiggling system may be implemented onto a real FPGA allowing a more accurate overhead analysis and its true evaluation in the presence of radiation. The reliability of Jiggling to mitigate transient as well as permanent faults should also be studied with the possibility of doing away with Scrubbing.

Further developments to the architecture are required to allow repair of sequential modules. The fitness evaluation procedure for sequential systems is necessarily more complex, and remains the key area for future work. For larger benchmarks, routing restrictions may be introduced as well as a more comprehensive fault model.

References

- [1] M. Abramovici, J. Emmert, and C. Stroud. Roving STARS: An integrated approach to on-line testing, diagnosis, and fault tolerance for FPGAs in adaptive computing systems. *3rd NASA/DoD W. on Evolvable Hardw.*, page 73, 2001.
- [2] L. Barnett. Nectrawling – optimal evolutionary search with neutral networks. In *Proc. Cong. on Evol. Comp. (CEC)*, pages 30–37. IEEE, 2001.
- [3] E. Fuller, M. Caffrey, A. Salazar, C. Carmichael, and J. Fabula. Radiation characterization, and SEU mitigation, of the Virtex FPGA for space-based reconfigurable computing. *Xilinx Application Note*, 2000.
- [4] J. Lach, W. Mangione-Smith, and M. Potkonjak. Low overhead fault-tolerant FPGA systems, 1998.
- [5] A. Lesesa and P. Alfke. A thousand years between single-event. *Xilinx TechXclusives*, March 2003.
- [6] J. Lohn, G. Larchev, and R. DeMara. A genetic representation for evolutionary fault recovery in Virtex FPGAs. *Evolvable Systems: From Biology to Hardware*, 5th Intl. Conf. (ICES 2003), LNCS2606:47–56, 2003.
- [7] D. Mange, M. Goeke, D. Madon, A. Stauffer, G. Tempesti, and S. Durand. Embryonics: A new family of coarse-grained FPGA with self-repair and self-reproducing properties. In E. Sanchez and M. Tomassini, editors, *Towards Evolv. Hardw.: The evol. eng. approach*, volume 1062 of LNCS, pages 197–220. 1996.
- [8] H.P. Schwefel and G. Rudolph. Contemporary evolution strategies. In F. Morán, A. Moreno, J. J. Merelo, and P. Chacon, editors, *Adv. in Artificial Life: Proc. 3rd Eur. Conf. on Art. Life*, volume 929 of LNAI, pages 893–907, 1995.
- [9] S. Straulino. Results of a beam test at GSI on radiation damage for FPGAs Quick-Logic QL12x16BL and Actel 54SX32.
- [10] F. Sturesson and S. Mattsson. Radiation pre-evaluation of Xilinx FPGA XQVR300. *European Space Agency Contract Report*, September 2001.
- [11] A. Thompson. Evolving fault tolerant systems. In *Proc. 1st IEE/IEEE Int. Conf. on Genetic Algorithms in Eng. Systems: Innovations and Applications*, pages 524–529. IEE Conf. Publication No. 414, 1995.
- [12] A. Thompson and P. Layzell. Evolution of robustness in an electronics design. In J. Miller, A. Thompson, P. Thomson, and T. Fogarty, editors, *Proc. 3rd Int. Conf. on Evolvable Systems*, volume 1801 of LNCS, pages 218–228, 2000.
- [13] S. Vigander. *Evolutionary Fault Repair of Electronics in Space Applications*. Dissertation, Norwegian Univ. of Sci.&Tech., Trondheim, Norway, February 2001.
- [14] J. J. Wang, R. B. Katz, J. S. Sun, B. E. Cronquist, J. L. McCollum, T. M. Speers, and W. C. Plants. SRAM based re-programmable FPGA for space applications. *IEEE Transactions on Nuclear Science*, 46(6):1728–1735, December 1999.
- [15] S. Yu and E. McCluskey. On-line testing and recovery in TMR systems for real-time applications. In *Proc. IEEE Intl. Test Conference*, pages 240–249, 2001.
- [16] S. Yu and E. McCluskey. Permanent fault repair for FPGAs with limited redundant area. *IEEE Intl. Symp. on Defect & Fault Tol. in VLSI Sys.*, page 125, 2001.
- [17] R. S. Zebulum, D. Keymeulen, V. Duong, X. Guo, M. I. Ferguson, and A. Stoica. Experimental results in evolutionary fault-recovery for field programmable analog arrays. *2003 NASA/DoD Conf. on Evolvable Hardw.*, pages 182–186, 2003.